CS569 – Final Report

Xu Zheng

Onid: zhengxu

Student ID: 932-509-227

Jun 6, 2016

1. Introduction

As we know, Random Testing is a Black-box testing method. Instead of analyzing the source code, generating random inputs to test the software is the purpose of random testing. The disadvantage of random testing is that there will be lots of useless test cases generated. So that it is very important to have efficient algorithms to generate its test cases. To this end, there are many researchers developed enhanced random testing such as "Feedback-Directed Random Test Generation" [1], "Feedback-Controlled Random Test Generation" [2] and "Adaptive Random Testing" [3]. Their common idea, which is also my idea for this project, is selecting the test cases to improve the effectiveness of testing.

My original idea was to develop a test case generation algorithm that based on "Feedback-Controlled Random Test Generation" and "Adaptive Random Testing". Feedback-directed random test generation is a widely used technique to generate random method sequences. It can use the feedback to guide the generation. According to the research, limiting the amount of feedback can improve diversity and effectiveness of generated tests. And the main difference between feedback-directed random testing is that it makes the use of the feedback from the previously generated inputs to guide the generation. It is convincing me that feedback-directed random testing is effectiveness with the respect to the fact it has been used widely in industries filed. According to their experiments result, feedback-controlled random test generation is highly effective.

After I understand how to use TSTL API, I think it is hard for me to implement my idea using TSTL API. So I made some changes to my original algorithm. The original idea was that to use multiple pools concurrently to generate the inputs, instead of using a single pool. To be more specific, each pool has different contents, so that different pools will guide the generation toward different direction. By using different pools concurrently, the algorithm can dynamic determine in which direction to continue generate inputs by analyzing the feedback information.

Consider that using multiple pools concurrently is hard to implement, I changed it to use two pools in order. It means the second pool is generated from the first pool. And there is a filter used to select inputs from previous pool to current pool. My new idea is inspired by coverage tester algorithm that divides test cases generating into different phase based on threshold coverage.

2. Project Implementation

My tester can read several arguments from command line: timeout, seed, depth, width, faults, coverage and running. To implement this, I used a python library called "argparse" to parse the command line arguments. And I have set defaults for all the arguments mentioned above.

To be more specific, user can set the running time limit, which refers to "timeout" argument, for the tester. And user can set the seed of Random.random() function for random number generation in the code. User also can choose the depth and width of the test. In addition to this, my tester can display how many faults have been encountered in the test, the coverage information and running information by setting the correspond arguments to "1" instead of "0".

There are two pools will be created to store the test cases generated. And the time for each one is half the total run time. It means that the first pool will be created during the first half time. And at the rest time, the tester will generate new test cases based on the first pool. The results I got show that my tester has better performance than random tester.

3. How to run the tester

To run the finaltester.py, you need to input all seven arguments. For example, if I input "python finaltester.py 30 1 100 1 0 1 1" in the command line. The first argument "30" means the total run time of the tester is 30 seconds. In this case, the time of constructing the first pool and second pool are all 15 seconds. In order to prevent the running time exceed the expected time, the tester will check the current running time every time it finish a generation. The second argument "1" represents the seed to generate the random number. The third argument "100" and the fourth argument "1" represent the depth and width of the test generation respectively. In this case, the max depth of test generation is 100. The fifth argument "0" means do not check faults. Conversely, the tester will check the faults if this argument is set to "1". The sixth argument "1" means to produce a coverage report. The tester won't produce a coverage report if this argument is set to "0". And the last argument "1" means to produce running information on branch. The running information won't be produced if this argument is set to "0".

To run the mytester.py, you do not need to input all arguments mentioned above. This tester is more flexible in taking the arguments. To be more specific, you just need to input "python mytester.py –t 3 –s 1 –d 100" to do the same work as you input "python finaltester.py 30 1 100 1 0 1 1". If you just input "python mytester.py", then the tester will run with default configuration. You can see the help information with the input arguments by inputting "python mytester.py -h".

4. Results

The results of my tester running on avl tree are as follows:

	Milestone1	Milestone2
Running Count	209	209
Brach Count	209	224
Statement Count	155	166

As you can see, after I made some changes to my algorithm, the tester has a better performance than before. The interesting thing I found is that if my tester stop after created the first pool, the results would be the same as the tester1. It turns out my algorithm do improve the test cases generation.

5. Future work

Since I have applied my algorithm to the tester, I want to figure out how to improve it. The key point is how to decide which test cases should be selected. In another word, I will work on improving the filters to make the tester to generate test cases effectively. Instead of choosing coverage as the threshold to filter the inputs, maybe

I could find another value to determine which inputs should be selected to next pool. And maybe the number of the pools affects the performance of the tester as well. I want to do more experiments to test my idea. I will focus my work on all mentioned above in the future work. I believe there is still some room to improve my algorithm.

References

[1] C. Pacheco, S. K. Lahiri, M. D. Ernst, and T. Ball. Feedback-directed random test generation. In *Processing of the 29th international conference on Software Engineering*, ICSE'07, pages 75-87, 2007

[2] K. Yatoh, K. Sakamoto, F. Ishikawa, and S. Honiden. Feedback-controlled random test generation. In *Processing of the 2015 international symposium on Software Testing and Analysis*, ISSTA'2015, pages 316-326, 2015

[3] T. Y. Chen, F. C. Kuo, R. G. Merkel, and T. H. Tse. Adaptive random testing. In *Processing of the 9th Asian Computing Science Conference*, pages 320-329, 2004