CS 569 - Part 3 – Prof. Alex Groce

Eman Almadhoun – 932909951

Introduction:

It seems that creating test generation for testing the Software Under Test is a bit hard with misleading a perfect technique. Not every test generation can cover all the statements and branches of the SUTs. Especially when you want to get to the narrow paths for finding any missed bugs on them. With using a good algorithm or some knowledge of machine learning, the test generation might be better to find most failures and cover most the code. It is very important to create a test generation with a minimum cost. As Prof. Alex explain that in the machine learning there is a technique that we can explore thing which looks at an interesting data to collect and exploit which means we get to this data using some statistics to find more bugs. I am working on exploring and exploit method for finding the failures in the Software Under Test and put a lot of affording to maximize code coverage. In this project, I will introduce a method that using code coverage to measure the distance in different input types [1].

Algorithm Explanation:

This project aims to maximize the coverage on the Software Under Test which will make the test generation is more effective for finding bugs. Branch coverage is my goal which I will focus on my test generation. The algorithm which I used is focusing on a part of guiding the test cases using coverage data. In the test generation, firstly, I run pure random tests which generate random tests with random actions. After that, I look up for any failure while the tests cases are generated. Failures mean the actions in TSTL are not ok and have errors and the properties are not ok. TSTL is Template Scripting Language which will check on the actions for the SUT and check on properties. If there is any bug, the tester will save it in a file named failure”i”.test in the current directory. Then, I am collecting all the tests in a list. I will calculate the average for that list and try to find all the tests below the average which might have uncaught failures. Then, I put the result in another list and try to do some statistics on them for exploit. I make some measurement after that which is take 1 and I minus it from the below mean list divide it by the length of coverage list and the result we will use it for exploit [2]. {`exp = 1 – len(belowMean) \ len(covCount)}`. After that, I generate random float number for exploit and check if it is greater than the result of the expression and then backtrack for states which will take the tests which generated by state and restore the system [3]. Finally, the tester will check for any failure again in case there still any after the exploit.

Run Test Generation:

The test generation is able to be run in the command line which allows us to specify different parameters (timeout, seed, depth, width, faults, coverage and running) with it. To run the test generation on the command line we need to type for example: python tester1.py 30 1 100 1 0 1 1. The first parameter is the timeout which the test generation will stop when the runtime reaches that number. The second parameter is the seed which generates a random number of seeds. The third parameter is the depth and The fourth parameter is width. The fifth parameter is to check for fault when it is 1. 1 means the test generation will search for bugs and 0 means the test generation do not look to find bugs. The sixth parameter is printing TSTL internal coverage report to know the statement and branches were covered during the test to guide it. This will work when we specify 1 and 0 for do not print it. The last parameter is for running which will print the elapsed time, total branch count and new branch. It works when we put 1, 0 will not do anything. At this stage, the test generator cannot find the bugs all the time. It needs some improvement to order to find the failures.

References:

1. Z. Zhou, "Using Coverage Information to Guide Test Case Selection in Adaptive Random Testing," in 34th Annual IEEE Computer Software and Applications Conference Workshops, 2010.
2. B. Jiang, Z. Zhang, W. K. Chan, and T. H. Tse, "Adaptive random test case prioritization," in *Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering*. IEEE Computer Society Press, November 2009, pp. 233–244.
3. A. Groce; J. Holmes; J. Pinto; J. O'Brien; K. Kellar; P. Mottal, "TSTL: The Template Scripting Language," in International Journal on Software Tools for Technology Transfer.