Final Project Report of CS569

Name: Wei Liu

Student ID: 932305582

April 19 2016

Section 1: Introduction

Based on many research papers and reports about different test generation methods, automatic test generation is discussed in these recent papers. In my opinion, to implement the principle of test generation complex algorithm is the best way. So, I prefer to use static test and dynamic test driven generation code automatic test generation. In this project, I will used it to realize the algorithm. Random tester has been proved to be a simple and effective way to perform software testing tasks. However, the performance of random tester is really bed based on repeat and useless test cases. To compare with it, SUT has its better performance to reduce overhead, save time and so on. Based on these information, I prefer to find a way or implement a test harness on random tester. Specificly, Adaptive Random Testing is good way to implement adaptive test. For more information, that is to say, the adaptive random testing is an intelligent algorithm, can choose a relatively good test case, in order to reduce unnecessary testing input. Commonly used random testing study, the fault caused by input rate used in the determination of the validity. This paper also shows that ART provide higher efficiency and more confidence in the reliability of the SUT, even without a failure is detected.. As a large number of flexible API, TSTL is absolutely feasible to implement ART algorithm. There are many different standard to describe the uniform. In the article, the author mainly consider using in the execution group and

the Euclidean distance between the elements in the candidate group. Generally, the more evenly distributed with less test cases have a better chance of test cases to a failure mode. Also, ART, which is considered the distribution of the input space of test cases, and attempts to generate test cases more evenly. In addition, based on article Code Coverage of Adaptive Random Testing, ART can have the same number of test cases, and this is an interesting fact random testing to a higher code coverage. We picked up the next test case, making the minimum distance should be the largest minimum distance. According to this standard, we actually choose more appropriate test cases the next iteration. In my opinion, test generation for testing software is not a exactly perfect technique, because it is hard to find all bugs or mistakes. We don't have any tester or test generation can cover all SUT files to find bugs or mistakes. Even though all effective process may have error or revealing test cases, and sometimes most of test cases cannot be reachable. When you are trying to find any bugs by narrow paths, its disadvantage will be obvious. By this fact, we need a cheap and easy algorithm or machine learning for test SUTs. In my project, I will try to find a machine learning way or an easy algorithm, which is ART algorithm to collect data and try to figure out all bugs and in this way, I prefer to use this algorithm to cover software and find bugs and try my best to cover more than now. I think I can make it better. After taking the information of test cases into consideration, the tester will be more reasonable and experiment significantly better.

Section 2: ALgorithm

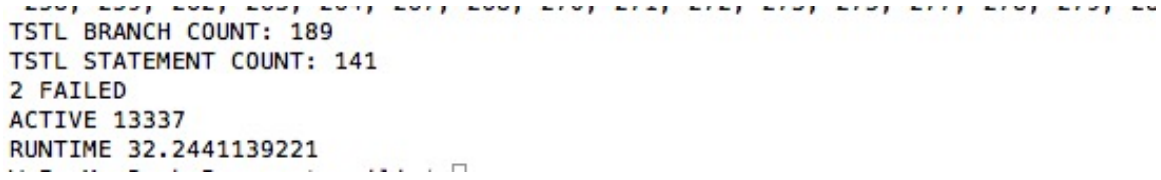
This algorithm comes from Alex, which is our professor. I used it to test software and use it as black box algorithm. I think this algorithm can not operate the analysis part effectively. So I

found more improvement to make it perfect.
For my algorithm, firstly, I initialized some parameters, which are random.Ramdom(), sut.sut and time.time. I used global variables in my code for structure. More than these, I also set some parsing parameters in my code, I will talk about them later. Secondly, I collected my data statements, which are random statement and data coverage.

Section 3: Testing process of generation

When running the testing generation, users can just run in the command line with parameters. First of all, put the sut.py file and tester1.py file in the same directory. To run the test generation on the command line, use the following code:
python tester1.py 30 1 100 1 1 1 1
The first number, which is 30 indicates the test generation timeout will stop running. That means the tester will run 30 seconds. The second represents the seed, python random. Random objects used for random number generation code. The third parameter, which is 100 indicates the maximum length of the test generation. The next represents the maximum memory is basically a search width. The next three parameters can be set to 0 or 1. Finally, the third parameter is the test generation SUT does not check for errors. The following two parameters means that the final report will generate coverage and branch coverage will be generated.

Here are some images when I processing my code.



```
Processing
First: AVL tree
Failed
(<type 'exceptions.ZeroDivisionError'>, ZeroDivisionError('integer division or m
odulo by zero',), <traceback object at 0x109026fc8>)
REDUCE
avl0 = avl.AVLTree()                                                    # STEP
0
val3 = 1                                                                # STEP
1
avl0.insert(val3)                                                       # STEP
2
val3 = 2                                                                # STEP
3
avl0.insert(val3)                                                       # STEP
4
val3 = 3                                                                # STEP
5
val0 = 4                                                                # STEP
6
avl0.insert(val3)                                                       # STEP
7
```

In the part 3, I can't get the generated files of faults and I only get 11/20. In this final test python code, I fixed this problem and now it can get everything that Prof Alex required.

Section 4: Parsing Parameters

I used BUDGET, SEED, COVERAGE, RUNNING, WIDTH, DEPTH and FAULT in this project.

I choose some of them and explain them.

Coverage: By internalReport function, the coverage of final coverage will be reported after I used this parameter.

Width/ Depth: These parameters are used to calculate the length and deep of the testing program.

Seed: This parameter is used to calculate the number of generated randomly.

Section 5:

References

[1] Chen, T. Y., Kuo, F., Liu, H., & Wong, W. E. (2013). Code Coverage of Adaptive Random Testing. IEEE Transactions on Reliability IEEE Trans. Rel., 62(1), 226-237. doi:10.1109/tr.2013.2240898

[2] I. M. T.Y. Chen, H. Leung. Adaptive random testing, 2004.

[3] A. Groce, G. Holzmann, and R. Joshi. Randomized differential testing as a prelude to formal verification. Software Engineering, 2007. ICSE 2007. 29th International Conference on, pages 621-631. IEEE, 2007.