Project fibal report
Zehuan Chen

Final Project Report

Introduction

This report is for reporting about my final version of my tester and random test algorithm. In my proposal, I still follow the main stream and use random testing, because there are several advantages about random tester. The random testing method is cheap to use, does not have any bias, and quick to find bugs. the way I choose is feedback-directed random testing.
However, there are some weaknesses of common random testing. The test cases are generated randomly by the generator. There could have some meaningless test cases in unguided random testing. In this way, some think that random testing is not effective as systematic testing. There are some enhanced random testing algorithm. Adaptive Random Testing is one way to decrease the meaningless test cases by trying to generate test cases more evenly.
First of all, with all my understanding about unit test and random tester, this algorithm is not very easy to applied on the TSTL, so after several times trying, a simple way comes out first, I decided to try on some basic method that is not that complicated.
In my first project, I try on with bread first search algorithm that mentioned on a paper that professor Alex Groce wrote, in that case the method is shown quite good performance result. After that I also applied some new improvement such as set time limit on each level of unit test, but I quickly found out that this method may not give enough time for each layer to go through lot branches and coverage.
The idea behind my new test generation is to apply mutation operator and crossover operator on the initial pool of tests that have been created. When the testing starts, tester uses one third of total time budget to perform pure random testing for collecting population. After getting the populations, tester will perform mutation and crossover on the populations, until they are fully “evolved” or timeout. Expectedly, This algorithm could get quite enough coverages and detect the bugs.

Implementations

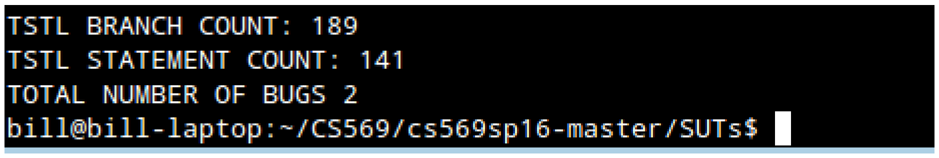
In general thinking, the random tester could keep working till it find the bug, the processing could also be infinity because the bug could be multiple.
The random source is derived from random.Random(), when every time the iteration is done, we shuffle the sequence of sut.state(), the stop condition could be vary, in my program, in order to meet the course requirement, maximun time restrict, depth restrict and width restrict can be applied. When the time you set up is not run out, the testing goes down to the button of the tree using backtrack(), then when the current depth of the traversal bigger than the max depth, the testing iteration is over this round. Using the sut.safely(actions) and test if there are any bug against the restriction write in TSTL file.
In paper “Feedback-directed random test generation”, the authors give the algorithm of Feedback-directed Random Test. First of all, I want introduce some of the innovative part of this algorithm. There are three sequences are introduced in: error sequence, non-error sequence, and new sequence. Error sequence and non-error sequence are used to store the cases that have been tested, the new sequence is used to store the new case that generator creates.

the generator will create a test case and put it into new sequence. Second, checking whether the new test case, which is stored in the new sequence, has been tested before. If the new test case has beentested then create a new case. Third, executing the new sequence and check the feedback.
If the feedback has no error then store the new sequence into non-error sequence. If the feedback has

error then store the new sequence into error sequence. This algorithm will avoid lots of non-sense test cases. The authors implement the algorithm in RANDOOP.
RANDOOP is a unit test generator for Java. It will automatically create unit tests for your classes in Junit format [2]. I want to implement Feedback-directed Random Test in TSTL. In the same time, let my random tester could deal with more SUTs and have a higher efficacy than unguided random test algorithm.

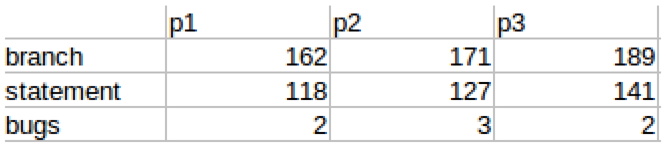
Results

I use (30 1 100 1 0 1 1) as my input parameter, and I use this set of input data to test my newest finaltester.py and my project 1 and 2. In the following content, I will show you my TSTL branch count and TSTL statement countresult of all my previous work and my final project.



```
TSTL BRANCH COUNT: 189
TSTL STATEMENT COUNT: 141
TOTAL NUMBER OF BUGS 2
bill@bill-laptop:~/CS569/cs569sp16-master/SUTs$
```

Figure 1, the final tester's result



	p1	p2	p3
branch	162	171	189
statement	118	127	141
bugs	2	3	2

Figure 2, data comparing

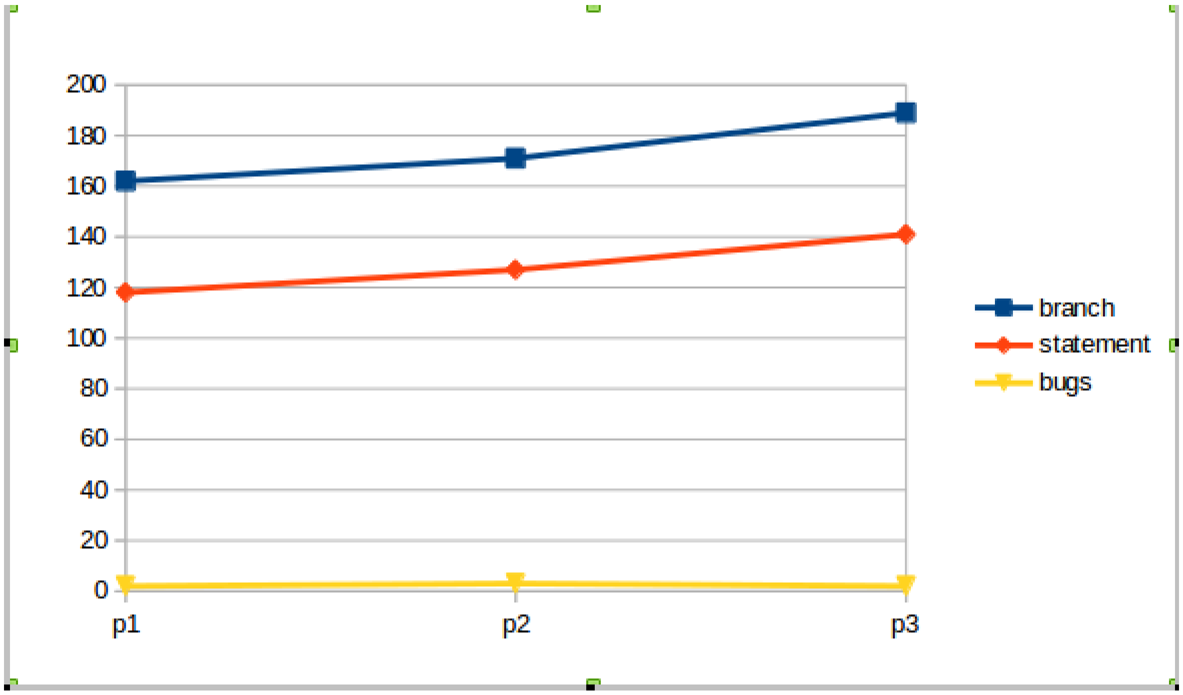


Figure 2, comparing among project 1,2 and 3

In this chart, we can see the improvement over the previous work, the branched coverage increase a quite bit when executing time for each level for 30 seconds. Once the program run out of the time, it would be terminated. So this result does not look so fancy. In my final version I add mutation function and crosscover functions, we eventually found that we find bugs and cover most of branches in the first phrase of BFS random testing and record all test case and branches coverage, we can find more branches and statements by using covered branches to mutate and crossover in the second phase.

Future works

This project indicates the finest understnding of myself,
Before I write my random tester, I read TSTL randomtester.py which are written by Professor Alex Groce. In my random tester file, I use same structure and lots of same function from TSTL randomtester.py. In the same time, I use sut functions in my tester.py.
All the functions from sut are produced by TSTL.
So, In the future, there will be more parameters could be accepted by my tester. For example, my tester could compare allfailing tests.

References

[1] Pacheco C, Lahiri SK, Ernst MD, Ball T. Feedback-directed random test generation. InSoftware Engineering, 2007. ICSE 2007. 29th International Conference on 2007 May 20 (pp. 75-84). IEEE.
[2] Pretschner, Alexander, Wolfgang Prenninger, Stefan Wagner, Christian Kühnel, Martin Baumgartner,
Bernd Sostawa, Rüdiger Zölch, and Thomas Stauner. "One evaluation of model-based testing and its automation." InProceedings of the 27th international conference on Software engineering, pp. 392-401. ACM, 2005.
[3] Chan FT, Chen TY, Mak IK, Yu YT. Proportional sampling strategy: guidelines for software testing practitioners. Information and Software Technology. 1996 Dec 31;38(12):775-82.