Feedback-directed Random Test Generation

CS569 Course project report
Jingyuan Xu

Feedback-directed Random Test Generation

1. Introduction

In software testing method, random testing can be used to supplement functional testing. It mainly according to the tester's experience in checking functionality and performance for a software. Random testing can retest models not been covered by test cases in a software. It also can be used in testing new features in a software. The key points are checking special circumstances points, the special usage of the environment , and concurrency. Usually each of the test versions of software need to perform random testing, especially for the final version, it should pay more attention.

2. Algorithm

The algorithm I use is from paper "Feedback-Directed Random Test Generation"[1]. The algorithm in there requires sequences that can be executed and get the flag value equals to redundant or illegal, and then creating a new sequence. Also, the algorithm checks two sequences are equivalent if they translate to the same code, modulo variable names, if it is, then the algorithm will generate a new sequence again, and repeat these steps. If it returns satisfied or violated, then we can check the test result. In the pseudocode, we append failures into error sequence, then finally we can check all errors in a file.

3. Experiment

3.1 finaltester.py

Feedback-directed test method is a technique that improves random test generation. It can outperform systematic and undirected random test generation, in terms of coverage and error detection [1]. The code, which follow the method in Professor Groce's newCover.py file [3], implements the algorithm "Feedback-directed Random Test Generation". It contains three lists: newseq, error, noerror, they all use sut.currStatements() to store values. newseq stores all new values, error stores bugs, and no error stores non bug values.

For the first option, the code uses "while(time.time() < start + timeout):" to check if the time is out or not. This step should begin at the top of the program, in order to avoid timeout problem happens. If the condition successful, then use the method from "newCover.py", call randomAction(), to check the sut.safty in order to get bugs. A new algorithm function is defined like:

```
def checkAlg():
  global error,
noerror,newseq,currseq,states
  newseq=sut.newStatements()
  c = sut.check()
  if c:
     if(not ((newseq in error) or
(newseq in noerror))):
        states.insert(ntest-1,sut.state())

noerror.append(sut.currStatements())
  else:

error.append(sut.currStatements())
```

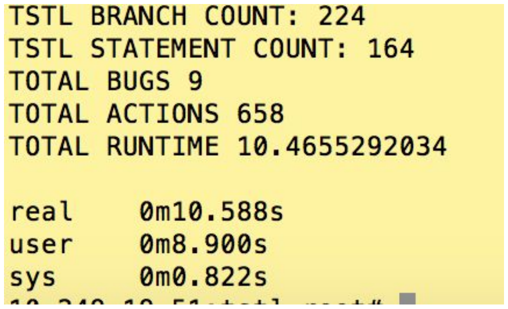
In this function, I follow the method from paper, and try to rewrite the java version pseudo code into python code. The basic idea from the algorithm is :If previous steps failure, the bugs will record into error list. If no errors, then the code can store the non bug value with sut.currStatements(). If the coverage number be entered, then the code can call "sut.internalReport()" to show the coverage message.

The changes for the algorithm implementation with papers are:

1. give newSeq all statements value, then justify error or no error sequence
2. output a .out file when error sequence is not empty
3. According to the paper, there are two conditions for noerror sequence append sequences, and change "and" keyword to "or" keyword
4. minor errors changed in running option.

3.2 Test result evaluation

In this standard test script, we have 7 options can use to adjust the test. For example, we pick avlbug1.py as the source for avl.tstl, then we can get sut.py for finaltester.py. Later we can input the command for finaltester.py :
>python2.7 finaltester.py 10 10 100 10 1 1 1
The result shows like this:



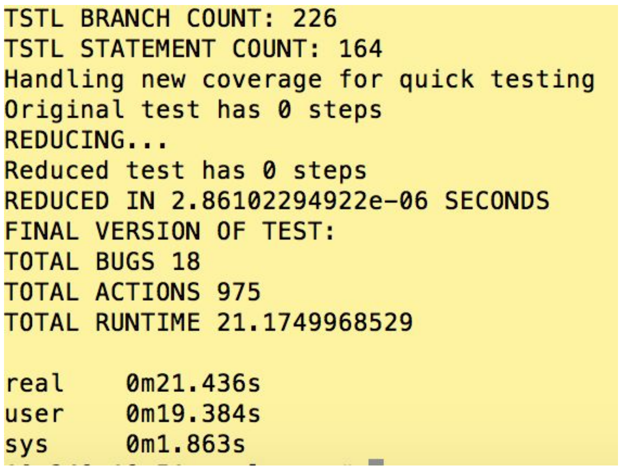
```
TSTL BRANCH COUNT: 224
TSTL STATEMENT COUNT: 164
TOTAL BUGS 9
TOTAL ACTIONS 658
TOTAL RUNTIME 10.4655292034

real    0m10.588s
user    0m8.900s
sys     0m0.822s
```

If we open coverage option, we can get internal coverage results.
we found out if we use system command time to test running time for our script, the system times shows 10.588, then the total time shows 10.465, just minor difference.
Back to our algorithm, we use error and noerror sequence to store our actions, when the failure happens, it will record the current statement into error sequence. This is a new way to record failures.

3.3 mytester.py

In this test script, I add a new option with quicktest method we learn from lecture. With the method, the code will generate new test cases for future test. If we want to use this option, just set 1 at end of the command. Here is the example:
> python2.7 mytester.py 30 1750 100 10 1 1 1 1
The command means, timeout = 30, seed = 1750, deepth = 100, width = 10, open show faults, show coverage, running, and quicktest generate option.
When we start quicktest, we will get the whole final version of test at end of the test result.
After we run quicktest with

>python2.7 mytester.py 30 1750 100 10 1 1 1 1
We will get a file called quicktest.0, in there we can found test cases.
If we don't need to generate the quicktest cases, then we can just set 0 instead of 1 in the command:
>python2.7 mytester.py 30 1750 100 10 1 1 1 0
The run result show like this:



```
TSTL BRANCH COUNT: 226
TSTL STATEMENT COUNT: 164
Handling new coverage for quick testing
Original test has 0 steps
REDUCING...
Reduced test has 0 steps
REDUCED IN 2.86102294922e-06 SECONDS
FINAL VERSION OF TEST:
TOTAL BUGS 18
TOTAL ACTIONS 975
TOTAL RUNTIME 21.1749968529

real    0m21.436s
user    0m19.384s
sys     0m1.863s
```

Then we can use the script regress.py which professor shows in the lecture to do future test.

4. Conclusion

Last term I learned how to write tstl script, and this term I learned how to use sut.py which comes form tstl, and use test cases in there to do random test. Tstl is a good testing tool,
It can help students understand testing method, and provide hands on experiences in testing. Also this term we learned use different algorithm method to design our test scripts, this is a good experience for us.

5. Reference

[1] C. Pacheco, S. K. Lahiri, M. D. Ernst, and T. Ball, "Feedback-Directed Random Test Generation," *29th International Conference on* Software *Engineering (ICSE'07)*, 2007.
[2] A. Groce and J. Pinto, "A Little Language for Testing," *Lecture Notes in Computer Science NASA Formal Methods*, pp. 204–218, 2015.
[3] "agroce/cs569sp16," GitHub. [Online]. Available at: https://github.com/agroce/cs569sp16/blob/master/suts/newcover.py. [Accessed: 06-May-2016].

6. Appendix: regress.py

```
import sut
import glob
import random

def runTests(ts):
  global sut
  for t in ts:
     assert not (sut.failsAny(t))
sut = sut.sut()
rgen=random.Random()

def randomTest(n):
  global sut,rgen
  sut.restart()
  for i in xrange(0,n):
     sut.safely(sut.randomEnabled(rgen))
  return sut.test()

qts = []

for qt in glob.glob("quicktst.*"):
  t=sut.loadTest(qt)
  t2=randomTest(len(qt))
  qts.append(t2)

runTests(qts)

m = []

for t in qts:
  m.extend(t)
runTests([m])

mr = sut.reduce (m,sut.failAny, keepLast=False,verbose=False)

runTests([mr])
```