Name: Huipu Xu

Email: xuhui@oregonstate.edu

Report of Part 3 of COMPETITIVE MILESTONE 2 (Revised)

Introduction

Random Testing, also known as monkey testing, is a form of functional black box testing that is performed when there is not enough time to write and execute the tests. Random testing is useful even if it doesn't find as many defects per time interval, since it can be performed without manual intervention. An hour of computer time can be much less expensive than an hour of human time. Even if the random test can help user a lot in detecting errors and failures, but no one test is perfect for debugging, random test cannot cover all statements and branches in SUT. This mean, sometime you cannot detect all fails in all branches in your programs. The problem and drawback of the test generation is not only in algorithm, but also in machine technique issue. Hence, right now, we can write a good test algorithm to improve the efficient of test generation. At present, I only need to fix some algorithm issue and improve its efficient.

Algorithm

In this report, you should have a basic implementation of a novel test generation algorithm using the TSTL API. And our code must include: timeout, seed, depth, width, faults, coverage and running. Here is some code provided by professor below.

See the code in randomtester.py:

```
if sut.newBranches() != set([]):
  print "ACTION:",a[0],tryStutter
  for b in sut.newBranches():
      print elapsed,len(sut.allBranches()),"New branch",b
  sawNew = True
```

My idea is just to improve the test's branch's coverage. Only we the test detect as many branches as possible, the algorithm will be more and more efficient. So I will handle the test method for SUT in black box. First of all, generate random tests with random actions by running pure random tests; after that, I will look up for any failure while the tests are generated and collect all them in a big pool. Then, calculate the mean for

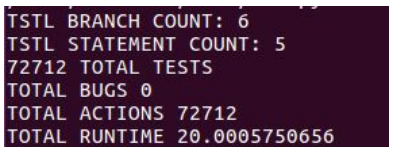
this pool and try to find all the test below the mean, which mean failures. Finally, put the result in another pool and do statistics for them.

In this code, there are 3 functions, the first one is randomAction(), which is for generating random tests and find bugs if faults = 1. And the second one, saveCover(), is for saving the current branches. And belowmean() is for calculating the number of branches belowmean.

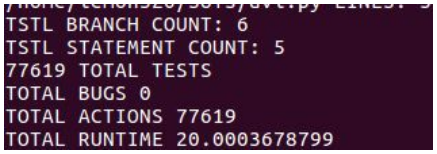
```
def belowmean():
        global belowMean
        sortedCoverage = sorted(coverageCount.keys(), key=lambda x: coverageCount[x])
        coverageSum = sum(coverageCount.values())
        try:
         coverageMean = coverageSum / (1.0*(len(coverageCount)))
        except Zero_Division_Error:
         print ("NO BRANCHES COLLECTED")
        for b1 in sortedCoverage:
         if coverageCount[b1] < coverageMean:
                 belowMean.add(b1)
         else:
              break
         print len(belowMean),"Branches BELOW MEAN COVERAGE OUT OF",len(coverageCount)
```

Here is an example:
Python tester2.py 40 1 100 1 0 1 1
40 is for the timeout: test generation will stop when the runtime reaches that number;
1 is for seeding which generates a random number of seeds;
100 is for the depth;
1 is for width;
0 is for faults: when it is 1 that mean the test generation will start to search for bugs or fails; when it is 0 that means test generation stop to detect bugs or fails.
1 is for 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.
1is 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.

tester1.py



tester2.py



Reference:
[1] Alex Groce coverTester.py. Retrived May 5, 2016, from https://github.com/agroce/cs569sp16/blob/master/SUTs/coverTester.py

[2] Alex Groce NewCover.py. Retrived May 5, 2016, from https://github.com/agroce/cs569sp16/blob/master/SUTs/newCover.py