The Final Part

CS569

Chao Zhang

Prof Groce Alex

1. Introduction

   The software testing is become more and more important right now. It is one of the most important technology to test the software and assure the quality of the software. Because of this, we already have many different ways to finish the test. Those testing approaches can generate the test case, by this way, they can detect the system failures. With those methods, random testing is a basic and wide used one because it is just pick the objects from the set of almost all possible inputs randomly, then the generate test cases and executing the software by using those test cases. But there are still people call it least effective method. So I think there are two problems with it, the first one is how to make it more accurate and efficient. The other one is how to generate a good random test. As we know that the random in the computer is not real random. If we give a huge amount of inputs which may include many meaningless test, this way can cover more possibilities but will lower the efficiency for sure. The most important thing for the testers is to find out some algorithms to generate the test case which can cover more branches and gives more effective set for the random test to minimize the test cases.

2. Implementation

   For this project, I implement it as three part. In the first part, I changed my idea about the algorithm for the ART because it is too hard. So for that part, I changed it to the BFS, the breath first search algorithm. During this part of this project, I focus on the basic of this project, as the request, I need to implement a novel test generation algorithm using the TSTL API. It should include the timeout, seed, depth, width, faults, coverage and running. So this part will finish the basic function to meet the requirement and I will improve them in the future. Like the article Adaptive Random Testing said, the adaptive random testing is based on the intuition. But the biggest problem is how to implement this algorithm with the TSTL API. The most important part of this in the F-measure. In this part, I didn't fully implement the Adaptive random testing because it is too hard to be finished. The first step is to implement the basic function as mentioned in requirement which are those seven arguments. I use parse_args to implement those seven arguments from the command lines. So the user can use the command line to make the specific limit to each of them. I also have random function to generate the random number in the project.   print nbugs,"FAILED" print "TOTAL ACTIONS", actCount print "TOTAL RUNTIME", time.time()-start It also prints the number of bugs founded and the number of action had been take.

   For the second part, I first fixed the problem which will cause crash in my tester1.py, I forgot the line: elapsed = time.time() – start, so it made my tester1.py crashed. After that, I add the coverage_count which can count the coverage, for this, I also need the leastcovered. I also added the mutate test. This test will get the data from the statebuff and do the mutate test. The data in the statebuff is collected from the state loop. With this test, my branches and statements goes up to 157 and 119 from 87 and 63. This made my test better than before.

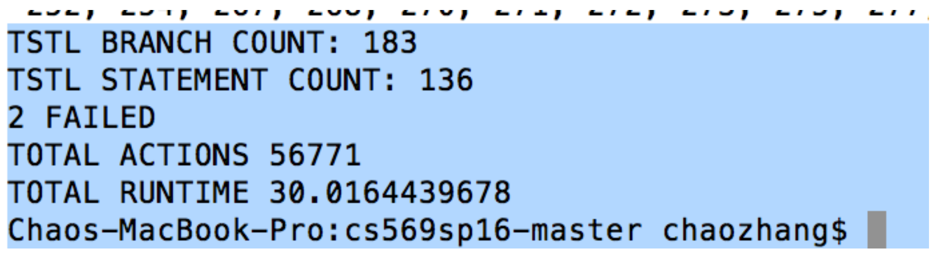
   Then, at the final part, I faced the biggest problem which is mytester.py cannot found any failures and it can't save the test results. I try to fix it but I failed. So in the last part, I changed the algorithm totally. The new idea is based on the coverTest.py from the SUTs. I still use the same way to get the seven arguments, the parse_args function. I set all the types and default values in this function. The make_config function will set the configs. So the main can get all of those needed information with the line:

parsed_args, parser = parse_args()

Config = make_config(parsed_args, parser)

Also, in this part, I added the saveFaults function and fix it to work. When the config.faults equal to 1, then this function will call the sut.saveTest and save the results in the file called failure#.test. the algorithm I used is basically the same as we mentioned in class. It will find and test the new branch and new statement within the time limit, for every new branch or statement found, it will plus one to the actCount. If found anything is not ok, it will add one to the number of bugs and call the saveFaults to save this failure, then, call the sut.restart to restart. For this part, I use the new algorithm to do the random test and this time the tester can save the failure. So this is the best version I did.

3. Results



```
TSTL BRANCH COUNT: 183
TSTL STATEMENT COUNT: 136
2 FAILED
TOTAL ACTIONS 56771
TOTAL RUNTIME 30.0164439678
Chaos-MacBook-Pro:cs569sp16-master chaozhang$
```

This is the results for the final part of my tester. From this picture we can see that I have a branch count of 183 and a statement count of 136. It is better than the second part. In the second part I got the 157 branch and 119 statement. I also found two failures which I never found in the previous parts. From the results, it shown that from the start to now, my algorithm improved each time. The first part, I just finish the basic functions. The second part I improved the algorithm I used in the first time, the coverage is increased and the same to the branch and statement but have no failures founded. The final part, I use the new algorithm and got a better result. This time I can found the failures. I think I can get an even better result if the time is longer.

4. Conclusion

For this project, I started with a complicated algorithm and I changed it to an easier one. During this process, I got deeper understand on the random tester and improved my algorithm to make it work better. At last, my tester can cover more branch and statement, can find failure which it can't in the beginning. It is a process to be better.

5. Reference

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

Discussed with Zhou Zheng.