CS 569 Project Report

Part #2

He Zhang

zhangh7@oregonstate.edu

Description of Algorithm

My algorithm is based on a practical software engineering rule: normally 80% defects are in 20% part of the software. So when find a bug somehow at the first time, we'd better test its "neighbors". These neighbors normally refer to neighbors in source code. But since it is not easy to track these neighbors, I use neighboring states and their actions instead, which is easy to get when using TSTL. States and actions also reflect the structure and logic of source code, so I think this idea is reasonable.
Random test randomly choose the input sequence (or called action sequence) through all the reasonable input space, which is a very efficient test algorithm. When we know nothing about the software under test, it's better to use random test. So I choose random test to find the first bug. When it is found, I use breadth first search (BFS) algorithm to find its neighbors and check if they contain any bugs. After reaching a setup BFS layers, which represents most "dangerous zone" has been checked, the algorithm restarts to use random test to find another "dangerous zone".
The algorithm can be described briefly below:

```
While time <= timeout:
{
    Do random test
    If find a bug when state is s:
        While BFS layer <= Target:
            Do BFS test start from s (or its neighbor)
}
```

Test Result Analysis

I using this algorithm to test avlbug1.py from class repo and linkedlist.py. The results is acceptable. For avlbug1.py, this algorithm finds out its only bug, and for linked list.py this algorithm finds out its 2 bugs out of 3. The setup timeout is 30 seconds and depth is 100. The bug information can be found in faults1.test after running test code.
I think this algorithm is good to test software which contains a lot of bugs, which concentrating in several zones. But for high quality software, this algorithm is not so good, because it always finds the same bugs. At this situation, it is better to use pure random test.

Possible improvement of the Algorithm

Compared to FSCS Adapted Random Testing (FSCS ART), this algorithm is also an improvement of pure random testing. But I does not use the concept of "distance",

which is hard to measure in software. But when one test case does not find a bug, it may be better to use a test case that is “far from” previous one. The interesting thing is that, calculating distance needs time. So the efficiency of ART may be not better than pure random test. But it is a possible way for improvement.

Also, there are a lot of algorithm which can replace BFS. BFS is not a very fast algorithm, although it searches completely. So maybe another search algorithm can be a better choice.

I do not consider branch and statement coverage in this algorithm, which is a possible improvement. If it can find some new coverage, it is more likely to find a new bug.

For avlbug1.py, this algorithm continues to find the same bug. When analyzing bug report and locate bugs, this is really annoying. I have not came up with the idea how to reduce similar bugs.