Name: Hengrui Guo

Student ID: 932504737

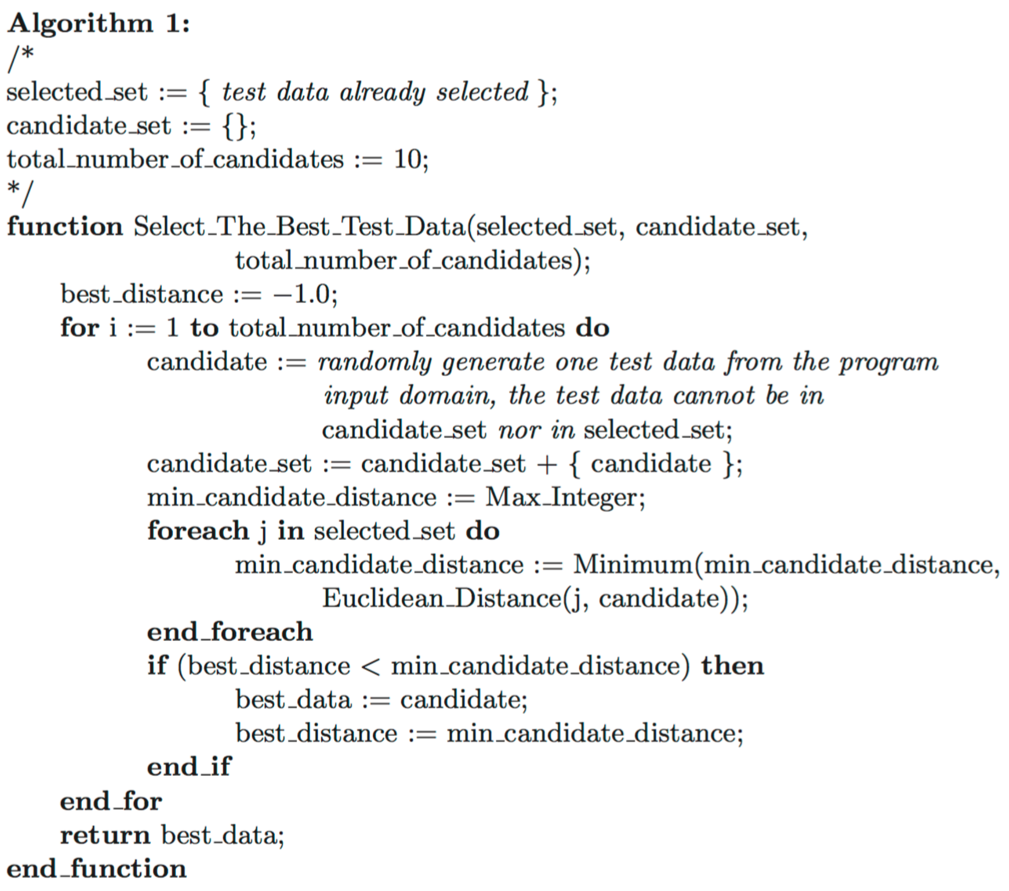
Date: June 6th, 2016

Prof. Alex Groce

Project Milestone

Introduction:

This project is based on Adaptive Random Testing, a black-box testing method from T.Y. Chen. The following is his core code about the algorithm in his paper:



```
Algorithm 1:
/*
selected_set := { test data already selected };
candidate_set := {};
total_number_of_candidates := 10;
*/
function Select_The_Best_Test_Data(selected_set, candidate_set,
                total_number_of_candidates);
    best_distance := −1.0;
    for i := 1 to total_number_of_candidates do
        candidate := randomly generate one test data from the program
                    input domain, the test data cannot be in
                    candidate_set nor in selected_set;
        candidate_set := candidate_set + { candidate };
        min_candidate_distance := Max_Integer;
        foreach j in selected_set do
            min_candidate_distance := Minimum(min_candidate_distance,
                    Euclidean_Distance(j, candidate));
        end_foreach
        if (best_distance < min_candidate_distance) then
            best_data := candidate;
            best_distance := min_candidate_distance;
        end_if
    end_for
    return best_data;
end_function
```

However, the main difference from mine project and his is the method to select the best data from the candidate data. Actually, I set a threshold to get the best data from the candidate data. And then use the best data collected from the candidate data to test by TSTL APIs.

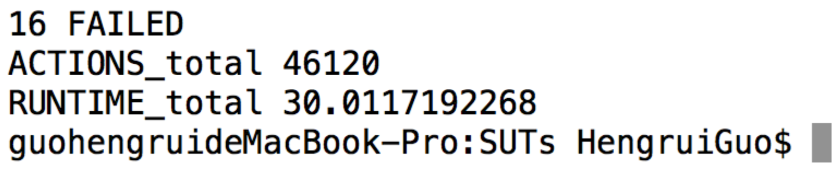
In my project, I would like to implement the Adaptive Random test method in TSTL for this project. As the random testing method is a black-box software testing technique [3], So it will use a large number of test cases to test the programs. So I determine to implement the new generation based on the random test generation with some modifications. The Adaptive Random test generation's main idea it tries to distribute test cases more evenly within the input space. It is based on the intuition that for non-point types of failure patterns, an even spread of test cases is more likely to detect failures using fewer test cases than ordinary random testing[1]. Experiments are performed using published programs. Results show that adaptive random testing does outperform ordinary random testing significantly. In other words, the distance from the selected test case in candidate set to every test cases in executed set is larger than distance from any other test cases in candidate set to every test cases in executed set. In this algorithm, the adaptive technique can reveal the failure quickly than the ordinary random test method. The following is the description and some explanation:

Algorithm description：
Step 1. Initialize, initialize the variable I may use and get the parameter from the inputs.
Step 2. Init sut.sut, sut.sut is the a function in TSTL API
Step 3. Init random.Random()
Step 4. Init time.time()
Step 5. Set variable depth, explore and savecoverage_test, depth, savecoverage and explore value are from the inputs. They will set after input the parameter.
Step 6. Set parsing parameter: BUDGET, seed, width, faults, coverage ,running. These values are same as the value like depth and they are assigned when the program get the inputs.
Step 7. Generate random STATEMENT, use the method in TSTL API to get the random statement

Step 8. Collect random statemtnt, Collect the statement generated by TSTL API for future use.
Step 9. Calculate data coverage, get the coverage for each data and stored them in a list for future use.
Step 10. Set coverage tolerance, the coverage threshold is based on the coverage for each data.
Step 11. Collect the statement lower than the tolerance.

Parsing Parameters:
There are seven parameters in my project, they are as following:
BUDGET: BUDGET tells the program how long the program is permitted to run.
SEED: SEED is used to generate random number in my program
DEPTH: DEPTH defines how much length the program can test in the algorithms
WIDTH: WIDTH is like DEPTH. It defines how deep the program can search in my project.
FAULT: FAULT is a bool value to define whether the program will check the bugs or not. If FAULT is 0, the project will not find any bug after running.
COVERAGE: Coverage of testing of final coverage are reported by this parameter for using internalReport() function.
RUNNING: RUNNING is a bool value to check the branch coverage after running randomtester.py.

Generation Testing：
Before test our generation, we should collect three files in one directory, which are tester.py, sut.py and AVL.py. The input the comment that shows below is used to compile the python file.
Command : python tester1.py 30 1 100 1 0 1 1
In this comment, there are 7 parameters and each of them has different meaning. The first one represents the running time. The second represents the seed, python random. Random objects used

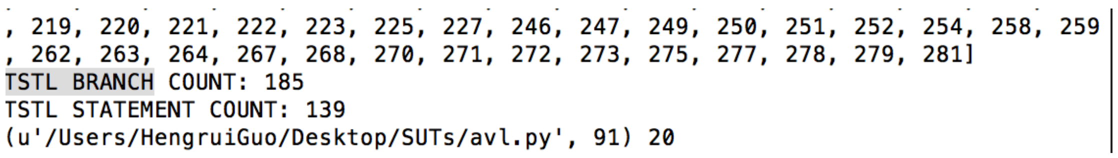
for random number generation code. The third parameter, which is 100 indicates the maximum length of the test generation. The next, which is 1, represents the maximum memory is basically a search width. The next three parameters can be set to 0 or 1. The following two parameters means that the final report will generate coverage and branch coverage will be generated.
The following result is the output of my test program:


```
16 FAILED
ACTIONS_total 46120
RUNTIME_total 30.0117192268
guohengruideMacBook-Pro:SUTs HengruiGuo$
```

The running time is 30 but the RUNTIME_total is 30.01. That is because the program is running some loop or statement in the code and doesn't reach the condition check statement. So it might have some delay. As shown on the screen, this bug tester is strong and it can find 16 failure or bug in 30 seconds by using the adaptive random test method to collect the best data. The most important part of my code is how to get the best data and store them into a list for use. Actually, the program will calculate the data occurrence rate and the more occurrence rate the data get the less weight it will be assigned by the program. And the weight of the data larger than the threshold will be moved to the in another list which is the best data and the data less than the threshold will stay in the candidate list. This method improves the efficiency to find bugs. The program will focus more on the data with larger weight and less occurrence rate.

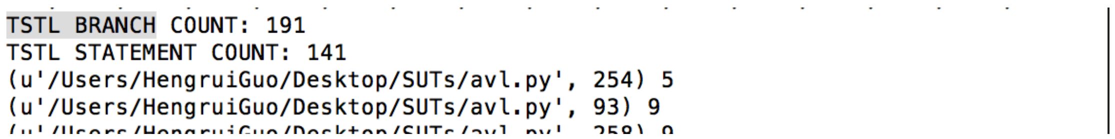
Improvement:

I have modified some code in the tester1.py and improved the algorithm. As a result, the new tester2.py have more branches and statements. Meanwhile, it can find more bugs.
The parameter I used: 40 1 100 1 1 1 1
Result:
(1) tester1.py:



```
, 219, 220, 221, 222, 223, 225, 227, 246, 247, 249, 250, 251, 252, 254, 258, 259
, 262, 263, 264, 267, 268, 270, 271, 272, 273, 275, 277, 278, 279, 281]
TSTL BRANCH COUNT: 185
TSTL STATEMENT COUNT: 139
(u'/Users/HengruiGuo/Desktop/SUTs/avl.py', 91) 20
```

(2) tester2.py:



```
TSTL BRANCH COUNT: 191
TSTL STATEMENT COUNT: 141
(u'/Users/HengruiGuo/Desktop/SUTs/avl.py', 254) 5
(u'/Users/HengruiGuo/Desktop/SUTs/avl.py', 93) 9
(u'/Users/HengruiGuo/Desktop/SUTs/avl.py', 258) 9
```

Reference

[1] Chan, F.T., Chen, T.Y., Mak, I.K., Yu, Y.T.: Proportional sampling strategy: guidelines for software testing practitioners. Information and Software Technology 38 (1996) 775–782

[2] T.Y. Chen, H. Leung, and I.K Mak. Adaptive Random Testing, 2004

[3] A. Groce, G. Holzmann, and R. Joshi. Randomized differential testing as a prelude to formal verification. *Software Engineering,* 2007. ICSE 2007. 29th International Conference on, pages 621-631. IEEE, 2007.