CS569: Competitive Milestone #2

Introduction:

This is an update to the previous algorithm that I submitted in milestone #1. The previous algorithm was a very simple random tester. The current implementation is an extended version of the previous algorithm. This one implements a coverage based approach. This approach has been borrowed from Dr. Alex Groce's in class lectures and the examples we have done in class. As of now, it is not my final version. I am still implementing an Adaptive Random Tester on the side which will eventually become my final tester.

Changes:

As of milestone 2, I would say that it is not exactly what I want it to be so I will be further improving The algorithm consists of the following stages: a random action is generated based on a random seed. The action is performed. If the performed action results in a failure, the failure is printed and is saved to a file. It then checks if the test found new branches. If so, it saves that test so we can back track to it later on. The algorithm is finished when the terminating condition is reached which can be if the timeout is reached. As of now, the algorithm is a simple random tester which relies on branch coverage information to perform some good tests. The few changes I have made to this algorithm are related to branch coverage information which allows the tester to perform better tests. Initially, my goal was to implement Adaptive Random Testing which I am working on the side. Unfortunately, I could not get the Adaptive Random Testing algorithm to work correctly so I decided to implement a random tester such as this which will form the base for Adaptive Random Testing.

How to run:

The tester requires a few different parameters as inputs from command line. These parameters include timeout, seed, depth, width, faults, coverage and running. Timeout is the time for how long the test needs to be run. Seed is used to determine the random generator seed. If the fault option is enabled, the failed test is saved to a file. Coverage enables the final results to be displayed such as branch count and statement count. Running info allows branch information to be displayed as the tests are performed.

Future Work:

It has been mentioned before that this is not my final version. I am still improving it and eventually will transform this into an Adaptive Random Tester. Currently, it relies on branch coverage information e.g. if there is a new branch found, we will save the test and later backtrack to it. This is the same concept as was discussed in class. Some of the information is borrowed from Dr. Alex Groce's examples. For the Adaptive Random Testing algorithm, I have been stuck on how to correctly take test cases and compare the distance between them. I have two options use Jaccard Distance or use Coverage Manhattan Distance to measure the distance between

[1] T. Y. Chen, F. C. Kuo, R. G. Merkel and S. P. Ng, "Mirror adaptive random testing," Quality Software, 2003. Proceedings. Third International Conference on, 2003, pp. 4-11.

[2] F.-C. Kuo ,An Indepth Study of Mirror Adaptive Random Testing,&rdquo, Proc. Ninth Int',l Conf. Quality Software, pp. 51-58, 2009.

[3] T. Y. Chen, "Adaptive Random Testing," 2008 The Eighth International Conference on Quality Software, Oxford, 2008, pp. 443-443.

[4] Z. Q. Zhou, "Using Coverage Information to Guide Test Case Selection in Adaptive Random Testing," Computer Software and Applications Conference Workshops (COMPSACW), 2010 IEEE 34th Annual, Seoul, 2010, pp. 208-213.