Omkar Thakur

932704163

CS 569 Final Report

Introduction:

Random testing is a type of testing which generates good amount of test cases depending on source code. The things that differentiate random testing from ordinary testing are uniform distribution without replacement. But the random testing suffers from the problem of fault detection. Adaptive Random Testing has 50% better coverage than normal proved to be an effective alternative to random testing. A testing process can be viewed as taking all possible inputs to the software under test, executing the samples one by one, and determining whether the outputs do not match the specification. Therefore, if the outputs do not match any specification, a software fault is revealed. A tester seeks to select test data with a view to maximizing the number of distinct faults detected.

The random testing has a disadvantage in form of fault detection. The Adaptive Random Testing overcomes this. Random Tester can infer statistical and reliability estimates. Since random testing does not make use of any information to generate test cases, it may not be a powerful testing method and its performance is solely dependent on the magnitude of failure rates. ART has been proposed as an enhancement to random testing. This is based on assumptions on how failing test cases are distributed in the input domain. Many authors have considered ART as more effective than only RT. I would also like to investigate how to adapt the coverage of ART. The research paper mentions a specific way to implement adaptive random testing, but I did not make it very specific to the paper. The hybrid approach also takes executed set and candidate set into consideration. It picks a testcase randomly from the input domain without replacement and puts it into a set called as candidate set. If there is no failure and this case goes into another set, is called executed test.

Related Work:

The unknown bugs are not identified by the TSTL (Template Scripting Testing Language). I also read randomtester.py and swarm.py by Dr.Alex Groce. I also used a number of SUTs from TSTL for my finaltester.py and mytester.py. The TSTL documentation provided great help for making of new SUTs.

Working of Algorithm:

1. First of all, I have used a number of parsers in order to get input arguments. The main arguments were as follows:

a.”TIMEOUT” – time for which tester will run

b.”SEED” – this is used for random generation in code

c.”DEPTH” – maximum length of a test generated by the algorithm

d.”WIDTH” - maximum memory that will be used for program e.”FAULT_CHECK” – either 0 or 1 depending on whether you want to check for faults or not

f.”COVERAGE_REPORT” – either 0 or 1 depending on whether report should be generated or not.

g.”DETAIL_OF_RUNNING” – either 0 or one depending on whether running information should be produced.

2. I created some variables for counting the failures and actions. I also used the time library to count the time elapsed in this overall process. Much help was taken from sut.py of Dr Alex Groce.

3. The length of the sut.newStatements will be compared with 0(zero), if it satisfies it then, it will utilize sut.state for it. Then the new statements from sut.newStatements will be used.

4. For each digit in x range from 0 to DEPTH specified by the user, the sut.randomEnabled is performed and the action count happened.

5. For each digit in y range from 0 to WIDTH specified by the user, the sut.randomEnabled is performed and the action count happens.

6. The sut.currStatements is searched and then the test is saved to a set

7. If the RUNNING argument is passed, it searches for new branches and it prints it along with the new branch

8. The variable is not saved in the ok variable, the bugs count will go on rising and the sut.failure will be used along with sut.prettyprinttest.

9. If fault is present, then sut.saveTest is used to print. If the FAULT is not present then the saveTest is not printed.

10. In the end, the results are printed (TSTL statements, Branches with total no of actions and elapsed time)

Running the script:

In the standard script, 7 arguments are being used. The following command runs the script

```
>python2.7 finaltester.py 30 1 100 1 1 1 1
```

Which means 30 = timeout, 1 = seed, 100 = depth, 1= width, 1= fault, 1= coverage, 1= running

```
>python2.7 mytester.py 30 1 100 1 1 1 1
```

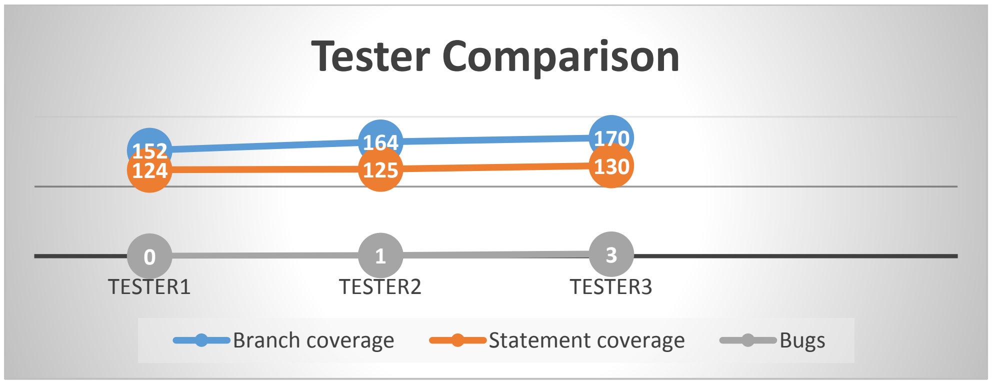
Actual Working:

```
val2 = 16                                # STEP 0

val1 = 12                                # STEP 1

val0 = 16                                # STEP 2
```

```
val3 = 18                                    # STEP 3
avl1 = avl.AVLTree()                             # STEP 4
avl1.delete(val0)                              # STEP 5
val0 = 18                                    # STEP 6
```

Improvement:

My previous implementation did not detect the faults and as a result, the failure file was not created. The current implementation detects keyboard interrupt as a bug and then lists it in the failure file. The current implementation was tested on various standard python suts given by the professor. No error was detected. The mutgen implemented by Prof.Groce was implemented did not work well for the previous implementation. But there was improvement in the number of branches. The improvement was also found in the number of branches and statements covered from the first turnover to the final tester. A number of suts were used. But the avlbug1.py was the most beneficial one. I have also implemented mutgen in mytester.py. There is a new interface in the mutgen which states the number of values inserted and deleted. Therefore, this interface is different.



Output:

```
    **********  CS569 Final Report  **********


Number of bugs found: 2

Number of actions: 62700

Number of width action count: 627
```

```
Total elapsed time: 30.00893116

174 BRANCHES COVERED

130 STATEMENTS COVERED


TSTL INTERNAL COVERAGE REPORT:

/home/omkar29/Desktop/cs569s16/tstl/examples/AVL/avl.py ARCS: 174 [(-1, 6), (-1, 11), (-1, 16), (-1,
31), (-1, 35), (-1, 44), (-1, 55), (-1, 70), (-1, 85), (-1, 111), (-1, 136), (-1, 148), (-1, 159), (-1, 171), (-1, 184),
(-1, 233), (-1, 245), (-1, 254), (6, -5), (11, 12), (12, 13), (13, -10), (16, 17), (17, -15), (31, -30), (35, 36), (36,
37), (37, 39), (39, -34), (44, 45), (45, -43), (55, -54), (70, 72), (72, 73), (72, 75), (73, -69), (75, 76), (75, 78),
(76, -69), (78, 79), (78, 80), (79, -69), (80, 81), (81, -69), (85, 87), (87, 89), (89, 90), (89, 95), (90, 91), (91,
92), (92, 93), (93, 104), (95, 96), (95, 98), (96, 104), (98, 99), (98, 102), (99, 104), (102, 104), (104, -84),
(111, 112), (112, 113), (113, -106), (113, 114), (114, 115), (114, 123), (115, 116), (115, 119), (116, 117),
(117, 118), (118, 119), (119, 120), (120, 121), (121, 123), (123, 113), (123, 124), (124, 125), (124, 128),
(125, 126), (126, 127), (127, 128), (128, 129), (129, 130), (130, 113), (136, 137), (137, 138), (138, 139),
(139, 141), (141, 142), (142, 143), (143, -134), (148, 149), (149, 150), (150, 151), (151, 153), (153, 154),
(154, 155), (155, -146), (159, 160), (159, 168), (160, 161), (160, 166), (161, 162), (162, 163), (163, 164),
(164, 166), (166, -158), (168, -158), (171, 172), (171, 180), (172, 173), (172, 178), (173, 174), (174, 175),
(175, 176), (176, 178), (178, -170), (180, -170), (184, 185), (184, 214), (185, 186), (185, 207), (186, 187),
(187, 188), (187, 190), (188, 205), (190, 191), (190, 192), (191, 205), (192, 193), (192, 197), (193, 205),
(197, 198), (198, 199), (199, 200), (200, 203), (203, 205), (205, 206), (206, -182), (207, 208), (207, 209),
(208, 212), (209, 210), (210, 212), (212, -182), (214, -182), (233, 234), (234, 236), (236, 237), (237, 238),
(238, 239), (238, 241), (239, -229), (241, 236), (245, 246), (245, 249), (246, -244), (249, 250), (250, 251),
(251, -244), (254, 255), (254, 257), (255, -253), (257, 258), (258, 259), (259, 260), (259, 262), (260, 259),
(262, 264), (264, 265), (265, 266), (265, 268), (266, 265), (268, -253)]

/home/omkar29/Desktop/cs569s16/tstl/examples/AVL/avl.py LINES: 130 [6, 11, 12, 13, 16, 17, 31, 35,
36, 37, 39, 44, 45, 55, 70, 72, 73, 75, 76, 78, 79, 80, 81, 85, 87, 89, 90, 91, 92, 93, 95, 96, 98, 99, 102, 104,
111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 123, 124, 125, 126, 127, 128, 129, 130, 136, 137,
138, 139, 141, 142, 143, 148, 149, 150, 151, 153, 154, 155, 159, 160, 161, 162, 163, 164, 166, 168, 171,
172, 173, 174, 175, 176, 178, 180, 184, 185, 186, 187, 188, 190, 191, 192, 193, 197, 198, 199, 200, 203,
205, 206, 207, 208, 209, 210, 212, 214, 233, 234, 236, 237, 238, 239, 241, 245, 246, 249, 250, 251, 254,
255, 257, 258, 259, 260, 262, 264, 265, 266, 268]

TSTL BRANCH COUNT: 174

TSTL STATEMENT COUNT: 130
```

Evaluation:

The latest version of the tester is successful in finding the bug. But this is possible from avlbug1.py. The tester fails to detect bugs from other suts. TSTL fails to detect unknown bugs. Therefore, my finaltester fails to detect unknown bugs. The final version does detects failure and a failure file is made. This was not possible in the previous implementation. The finaltester does detect the keyboard interrupts appropriately. The running time of the tester sometimes goes over the allotted time when more than 2 bugs are found.

References:

[1] 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.
[2]. A. J. Offutt, A. Lee, G. Rothermel, R. H. Untch, and C. Zapf, “An Experimental Determination of Sufficient Mutant Operators,” *ACM Transactions on Software Engineering and Methodology*, vol. 5, no. 2, pp. 99–118, April 1996.
[3] T. Y. Chen, "Adaptive Random Testing," *2008 The Eighth International Conference on Quality Software*, Oxford, 2008, pp. 443-443.
[4]. P. S. Loo and W. K. Tsai. “Random testing revisited”. Information and Software Technology, Vol.30. No 7, pp. 402-417, 1988.
[5]. T.Y. Chen, F.-C. Kuo, R.G. Merkel, and T.H. Tse, “Adaptive random testing: the ART of test case diversity,” *Journal of Systems and Software*, vol. 83, no. 1, pp. 60–66, 2010.
Louis, Missouri, 15-21 May 2005, pp. 402 – 411.
[6] K. P. Chan, T. Y. Chen, and D. Towey. Restricted random testing. In *Proceedings of the 7th European Conference on Software Quality*, Lecture Notes in Computer Science, 2002.
[7] “Mirror Adaptive Random Testing”, T. Y. Chen ; Sch. of Inf. Technol., Swinburne Univ. of Technol., Hawthorn, Vic., Australia ; F. C. Kuo ; R. G. Merkel ; S. P. Ng