Final report
– CS569 spring 2016

Yi-Chiao, Yang
932526065

Our goal in this project is to design a generator through using the TSTL API. This generator is like a bug finder that can determine potential errors in SUT. In my milestone, I wanted to take the idea of Antirandom Testing that is proposed by Yashwant K. Malaiya in 1995. However, it is hard for me to present and use in my generator. The idea of Antirandom Testing is based on Hamming distance and Cartesian distance. In Antirandom Testing, we assume that the process generators some values after the first testing. In the second testing, we can get the total maximum distance from the previous test or all previous tests, and get the corresponding values of total maximum distance and repeat these procedures until we find a error or the resources of the process is gone. It process in a limited random set to find bugs, but in our project, it is hardly used when we use different random test number in every new statement. Therefore, I decide to change my algorithm to Breadth-Fast search algorithm.

The feature of Breadth-Fast search algorithm is traveling each node in a tree until find the goal that we want. The algorithm starts a node to visit following its adjacent neighbors, and visits these unvisited adjacent neighbors until the node visit all of nodes in the pattern. Basically, the Breadth-Fast search algorithm stops when it finds the goal. However, the goal in our project is to find bugs in my target. Our test file might not contain a bug. In this situation, if we set the factor to stop our generator, it our generators might run be a loop, and never stop. For this problem, we can decide how many times we want to use in search bugs by passing parameters from command line. On the other hand, we have to consider a situation. If users do not give enough time for our generators to search bugs, the efficiency might very lower. For this program, design another section to test low coverage branches, and after collecting these low coverage branches, I might focus on running these parts. This section might increase the overall coverage of SUT. Fist, I use Breadth Fast Search algorithm to travel all statements to find bugs. Second, I have a function to separate all statements into two parts, including lower coverage and high coverage through the median coverage, but in this function, it uses an approximate median coverage. Third,

statements have low coverage that means they do not be visited by system many times. This low coverage might contain software bugs, so I spend more time testing this part.

There are two sections in my main code. The first section is Breath Fast search algorithm that searches bugs and counts statements whose branch statement coverage. The second section is to run low coverage statements in order to find more bugs. It is hard to set how many times we use in first section and second section because our time is limited. If we arrange more time for the first section to search bugs and count statements whose branch statement coverage, it might miss some bugs in some place that our generators seldom visit there. If we arrange more time for the second section to visit these lower coverage statements, we might not have enough statements to develop. Thus, we have to observe how many time is appropriate and enough for both two sections. In milestone2, I had a huge problem in my code when we use the *sqlparse* as our test harness. The result after testing the sqlparse, my generator could not count any statement coverage, and it also affects TSTL branch count and running count. It is a tricky problem because my generators can execute other test harness. After an observation, I found that I use `sut.enable()` in my tester1.py the function of use `sut.enable()` is to return all enable action objects. If users pass the python random seed to my generators, it could make my generators get empty statement coverage. For this problem, I use randomEnable() instead of enable() because we can pass python random seed for randomEnable() to use.

In order to make comparison to show the efficiency between my several ideas in algorithm, I build two generators. These two generators are almost similar, but I add an extra section to run lower branch statements. The finaltester.py is about Breadth-Fast Search algorithm. This is simple one. Another file is named comparsion.py that adds the extra section. The comparsion.py file adds my previous idea about collecting all lower statement coverage and focusing on running this part many times until time is out.

For these two generators, I test then with avlblocks.tstl. This TSTL file is provided in class for testing our generators. I use avlbug1.py as the source file in avlblocks.tstl that contains many bugs.

The following information is the output both these two file:

In this figure, we the finaltester.py has more great statement coverage when source test file contains bugs.

```
10-249-212-240:generators Chiaoysbaby$ python runavl.py 30 30
comparsion comparsion/tester2.py
finaltester finaltester/tester2.py
10-249-212-240:generators Chiaoysbaby$ python tabulateCov.py
comparsion comparsion.avl.40s.30.tout RUNNING: 148 BRANCHES: 206 STATEMENTS: 151
finaltester finaltester.avl.40s.30.tout RUNNING: 210 BRANCHES: 210 STATEMENTS: 153


SORTED BY BEST BRANCH COVERAGE RUNNING TOTAL
finaltester.avl.40s.30.tout 210
comparsion.avl.40s.30.tout 148


SORTED BY BEST TOTAL STATEMENT COVERAGE
finaltester.avl.40s.30.tout 153
comparsion.avl.40s.30.tout 151


SORTED BY EITHER BRANCH COVERAGE METHOD
finaltester.avl.40s.30.tout 210
comparsion.avl.40s.30.tout 206


MEAN OVER ALL RUNS
finaltester 210.0 1 0.0
comparsion 206.0 1 0.0
```

This is finaltester.py output:

```
TSTL INTERNAL COVERAGE REPORT:
/Users/Chiaoysbaby/tstl/generators/avlbug1.py ARCS: 210 [(-1, 6), (-1, 11), (-1, 16), (-1, 31), (-1, 35), (-1, 44), (-1, 55),
 (-1, 58), (-1, 70), (-1, 85), (-1, 111), (-1, 136), (-1, 148), (-1, 159), (-1, 171), (-1, 184), (-1, 233), (-1, 245), (-1, 2
54), (6, -5), (11, 12), (12, 13), (13, -10), (16, 17), (16, 18), (17, -15), (18, 19), (18, 20), (19, -15), (20, 21), (21, 23)
, (23, 24), (24, 25), (25, 26), (25, 27), (26, -15), (27, 29), (29, -15), (31, -30), (35, 36), (36, 37), (37, 39), (39, -34),
 (44, 45), (44, 46), (45, -43), (46, 47), (46, 48), (47, -43), (48, 49), (48, 50), (49, -43), (50, 51), (50, 52), (51, -43),
(52, -43), (55, -54), (58, 59), (58, 64), (59, 60), (60, 61), (61, 62), (62, -57), (64, -57), (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), (8
9, 95), (90, 91), (91, 92), (92, 93), (93, 104), (95, 96), (95, 98), (96, -84), (96, 104), (98, 99), (98, 102), (99, -84), (9
9, 104), (102, 104), (104, -84), (111, 112), (112, 113), (113, -106), (113, 114), (114, 115), (114, 123), (115, 116), (115, 1
19), (116, -106), (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
, -134), (139, 141), (141, 142), (142, 143), (143, -134), (148, 149), (149, 150), (150, 151), (151, 153), (153, 154), (154, 1
55), (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), (25
0, 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)]
/Users/Chiaoysbaby/tstl/generators/avlbug1.py LINES: 153 [6, 11, 12, 13, 16, 17, 18, 19, 20, 21, 23, 24, 25, 26, 27, 29, 31,
35, 36, 37, 39, 44, 45, 46, 47, 48, 49, 50, 51, 52, 55, 58, 59, 60, 61, 62, 64, 70, 72, 73, 75, 76, 78, 79, 80, 81, 85, 87, 8
9, 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: 210
TSTL STATEMENT COUNT: 153
depth: 100
*** coverage_counter is : *** 153
*** failed *** 22
***total running time *** 34.8021190166
```

This is comparsion.py output:

```
TSTL INTERNAL COVERAGE REPORT:
/Users/Chiaoysbaby/tstl/generators/avlbug1.py ARCS: 206 [(-1, 6), (-1, 11), (-1, 16), (-1, 31), (-1, 35), (-1, 44), (-1, 55),
 (-1, 58), (-1, 70), (-1, 85), (-1, 111), (-1, 136), (-1, 148), (-1, 159), (-1, 171), (-1, 184), (-1, 233), (-1, 245), (-1, 2
54), (6, -5), (11, 12), (12, 13), (13, -10), (16, 17), (16, 18), (17, -15), (18, 19), (18, 20), (19, -15), (20, 21), (21, 23)
, (23, 24), (24, 25), (25, 27), (27, 28), (27, 29), (28, -15), (29, -15), (31, -30), (35, 36), (36, 37), (37, 39), (39, -34),
 (44, 45), (44, 46), (45, -43), (46, 47), (46, 48), (47, -43), (48, 49), (48, 50), (49, -43), (50, 51), (50, 52), (51, -43),
(52, -43), (55, -54), (58, 59), (58, 64), (59, 60), (60, 61), (61, 62), (62, -57), (64, -57), (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), (8
9, 95), (90, 91), (91, 92), (92, 93), (93, 104), (95, 96), (95, 98), (96, 104), (98, 99), (98, 102), (99, -84), (99, 104), (1
02, 104), (104, -84), (111, 112), (112, 113), (113, -106), (113, 114), (114, 115), (114, 123), (115, 116), (115, 119), (116,
-106), (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, -134), (139, 141), (141, 142), (142, 143), (1
43, -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, 17
3), (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, 21
2), (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)]
/Users/Chiaoysbaby/tstl/generators/avlbug1.py LINES: 151 [6, 11, 12, 13, 16, 17, 18, 19, 20, 21, 23, 24, 25, 27, 28, 29, 31,
35, 36, 37, 39, 44, 45, 46, 47, 48, 49, 50, 51, 52, 55, 58, 59, 60, 61, 62, 64, 70, 72, 73, 75, 76, 78, 79, 80, 81, 85, 87, 8
9, 90, 91, 92, 93, 95, 96, 98, 99, 102, 104, 111, 112, 113, 114, 115, 116, 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: 206
TSTL STATEMENT COUNT: 151
ACTION: self.p_avl[0].delete(self.p_int[1])
depth: 100
*** coverage_counter is : *** 126
*** failed *** 5
***total running time *** 36.0086090565_
```

In these several figures, finaltester.py has more efficiency in finding bugs, but it has less branch count. However, my comparison generator crashed when I test it with mysqlparse.tstl. The problem is about sut.reply() function. Because of time is limitation to this project. Thus, I do not fix it.

In my observation, compare these two different generators; if we want to focus on lower coverage statements, we have to consider how to arrange the time for these two sections. In my idea, the perfect test generator has two sections, including searching algorithm, and focus on weakness branches. The algorithm is like in the comparsion.py, but I have something wrong in my code that it crashed. However, I think that my idea of the test generator might be wrong, and it has less efficiency because I do not overall factors. If the second section famous on lower coverage statements, it cannot go back to search whole test file. In my algorithm in test generators, I do not build a function to go back to search whole test file. I think that it is also one of my drawbacks in comparsion.py file. The finaltester.py uses traditional method to look for bugs that has great efficiency than my idea of test generator. Thus, it is my final version test generators in this project.