AISTR Lab 1

Marios Marinos - 5353106
Mihai Macarie - 4668634

February 26, 2021

1 Task 1 : Branch Distance

We implemented a branch distance calculator by making use of the framework provided in FuzzingLab.java file. The program calls a function named **calculateBranchDistance**, which computes the branch distance of a given condition according to the rules, when a branch is triggered in the **encounteredNewBranch** function. This distance is added to the total distance for the current trace and also the branch is added to a map for later processing.

Instead of counting time, we control the program execution by the number of iterations fed into the fuzzer, which iteration is basically one InputTrace of traceLength 10 (which may vary due to the permutations). In our case, we used 2048 * 8 iterations and a random fuzzer. The result are shown in Table 1.

2 Task 2 : Hill Climber

The Hill Climber we implement is a bit different from the one proposed on the assignment. The first step is the same, we calculate the sum of the branch distances for the current trace. After that, we do random permutations (add an input symbol, delete a symbol, or change a current symbol) on the best branch, with some probability that we set in the beginning, (InputTrace) that has 0 BranchdistanceSum or the smallest distance from branches that we have already found with probability of 50%, to trigger this from false to true. Then, we use this sample instead to continue fuzzing. Finally, there is a 20% of generating random samples instead of doing permutations on the best sample we have till now, as it might get stuck in local minimums, and thus, we reset it. For the experiments running for task 2, the iterations were equal to 2048 * 8, the same as it was in task 1.

The unique branches that were found by our random fuzzer and the hill climber are shown in Figure 3. In all cases, a hill climber with high permutation probability works better than random fuzzing. The blue line displays the random fuzzer, whereas the red, yellow, and green represent the 10%, 50% and 80%

Problem number	Number of branches	Trace(s) with highest branch coverage	Trace(s) with lowest branch distance
Problem1	396	[[iF, iC, iA, iH, iJ, iC, iG, iH, iE, iC, R]]	[[iE, iI, iD, iH, iJ, iF, iB, iB, iJ, iA, R]]
Problem2	608	[[iA, iI, iJ, iC, iG, iB, iB, iF, iF, iA, R]]	[[iB, iH, iE, iI, iF, iD, iG, iH, iA, iG, R]]
Problem4	954	[[iB, iJ, iC, iD, iF, iD, iH, iC, iI, iG, R]]	[[iI, iB, iB, iB, iE, iD, iE, iH, iH, iG, R]]
Problem5	1538	[[iG, iI, iG, iD, iA, iF, iH, iH, iG, iF, R]]	[[iD, iE, iJ, iG, iB, iF, iB, iB, iB, iH, R]]
Problem6	2110	[[iD, iI, iB, iJ, iD, iJ, iJ, iE, iI, iF, R]]	[[iE, iG, iC, iI, iA, iI, iA, iI, iC, iH, R]]
Problem7	1560	[[iH, iI, iA, iM, iH, iN, iL, iN, iO, iI, R]]	[[iC, iA, iA, iA, iF, iJ, iC, iK, iI, iB, R]]
Problem8	1622	[[iB, iH, iA, iL, iB, iI, iN, iH, iE, iM, R]]	[[iD, iN, iE, iB, iI, iB, iG, iE, iN, iM, R]]
Problem9	3606	[[iH, iI, iF, iF, iL, iK, iD, iG, iE, iD, R]]	[[iA, iG, iN, iA, iK, iN, iC, iA, iA, iI, R]]

Table 1: Results for Task 1

respectively probability of permutation in a symbol. Finally, in Figure 4. for the same problems(13,14,15,17) we plot the convergence graph of error codes vs time. The same case applies here as in before, we run once the random fuzzer and 3 times with different permutation probability. For two of the four problems, the random fuzzer seems to have good results whereas in Problem 14 and 15 it is way worse than our implementation. Finally, the errors that were encountered by the random fuzzer are included on the hill climber, but the hill climber also finds more errors. For example, as shown in the table the errors that are found from both fuzzers are the same. For more details, in run_output folder are the datalog files that contains each error and what caused it.

	Random fuzzer error	Hill climber error
Problem13 (Error 77)	[I, N, G, A, C, O, I, L, F, J, R]	[C, B, E, N, I, M, C, B, A, B, F, R]
Problem 14 (Error 8)	[M, D, N, N, N, J, D, N, D, E, R]	[D, G, J, N, M, C, N, K, N, I, L, M, A, E, D, A, C, C, C, D, C, N, D, M, A, E, F, M, R]

Table 2: Results for Task 2

3 Task 3 : AFL vs our fuzzer

The traces that were used by AFL to find out error codes are different than the traces that our fuzzer created. For example in problem 15, our fuzzer detected Error 8 caused by [D, N, N, A, M, D, N, G, J, D, N, H, L, B, A, O, I, C, C, I, E, F, R] whereas AFL fuzzer found the same error with [F,D,N,N,D,N,D,E]. Another example in problem 13, our fuzzer detected Error 3 caused by [F, D, J, C, A, E, E, A, F, I, E, B, H, E, G, B, R] whereas with AFL it was detected by trace [C,E,C]. One thing that should be mentioned is that our fuzzer detected the same errors by using way more complex input traces.

Below, in Figure 1 (Problem 11 from top left moving to the right to Problem 15 to last one) and 2, it is shown how many the trace plots for the errors that can our fuzzer detect vs AFL fuzzer tool. Our fuzzer works better for small periods but then it converges and cannot find any more errors whereas AFL struggles in the start to find the errors but if we let it run for more time it will get better results than our fuzzer. The reason behind that might be that our fuzzer is specifically implemented for RERS problems whereas AFL can work in any program/problem. In addition to that, AFL heuristics might need more time to find the errors that occur whereas our fuzzer after some time it converges and cannot find any more errors.

Figure 1: Number of found unique errors(crashes) on the vertical axis, against the number of fed traces into our fuzzer on the horizontal axis, for problems 11 to 15.

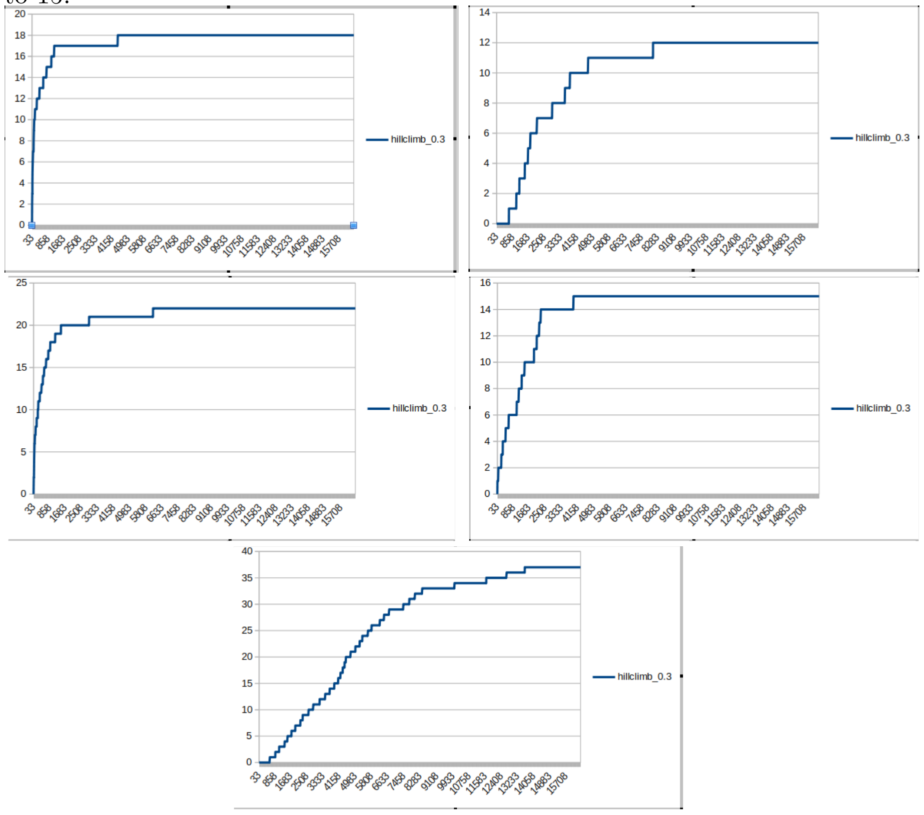


Figure 2: Number of found unique errors(crashes) on the vertical axis, against the number of fed traces into the AFL fuzzer on the horizontal axis, for problems 11 to 15.

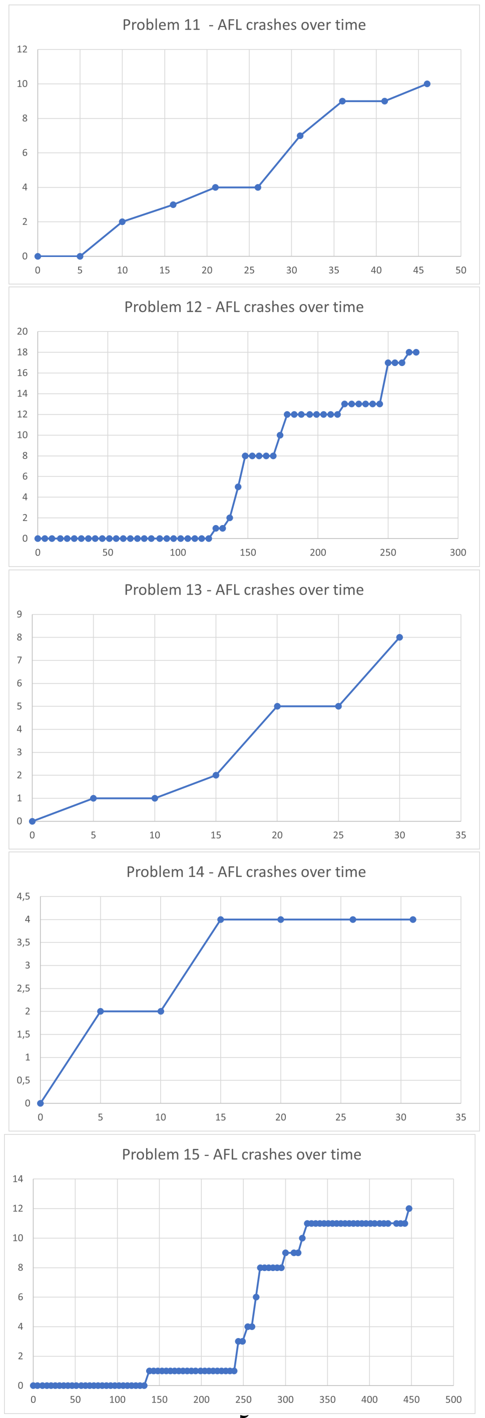


Figure 3: Number of found unique branches on the vertical axis, against the number of fed traces into our fuzzer on the horizontal axis, for problems 13,14,15,17.

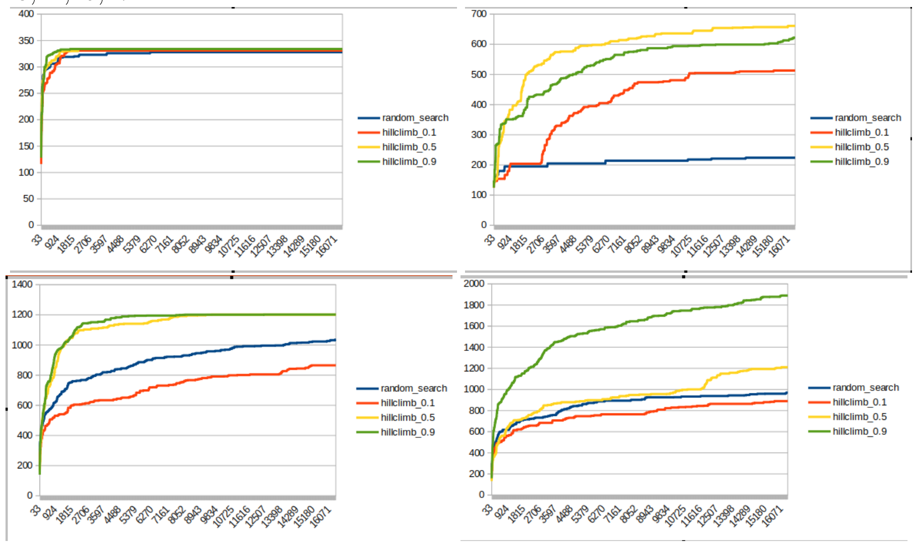
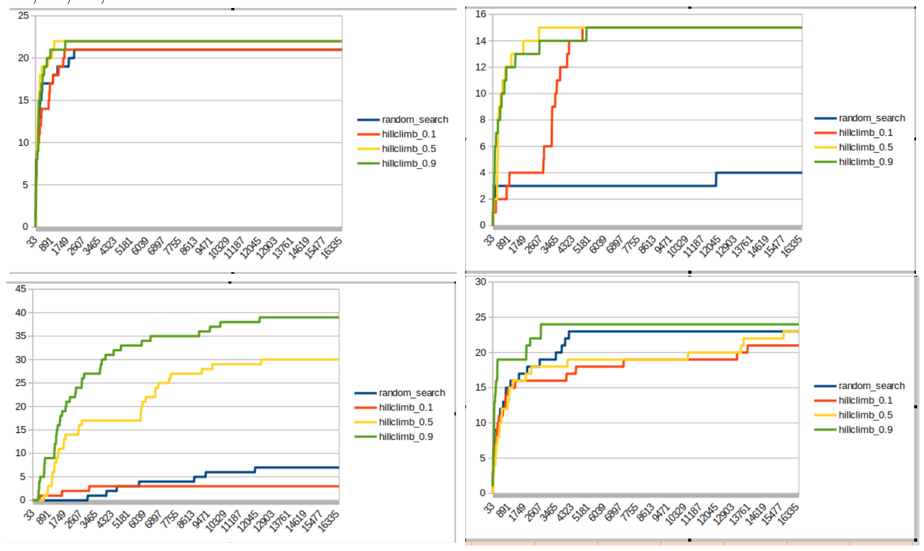


Figure 4: Number of found unique error codes on the vertical axis, against the number of fed traces into our fuzzer on the horizontal axis, for problems 13,14,15,17.


random_search
hillclimb_0.1
hillclimb_0.5
hillclimb_0.9