ASSIGNMENT - 2

Divya Reddy - 300290332
Maheedhar Vundela - 300282606

Question 1:
How does the Alternating Variable Method (AVM) work? How is this algorithm different from the Hill Climbing or Simulated Annealing algorithms we discussed in the class? Please try to describe the main ideas and the working of AVM in less than one page (using a few paragraphs)

AVM:
Alternating Variable Method (AVM)- The Alternating Variable Method (AVM) is a local search method that was originally used to automatically generate numerical test data. It is a quick and effective local search technique. It employs iterative pattern search. Variable search patterns for the local search process. To arrive at an objective value, it performs exploratory moves on the values of variables in each situation by changing each variable in the vector one by one, bringing the objective value closer to the optimal value.

The main part of AVM is the choice of direction, which is accomplished by making pattern moves based on exploratory moves, improving objective values, and deciding whether to go in a negative or positive direction. The direction changes as the objective value of the pattern moves up. This pattern continues until the search for the optimum (pattern moves) and objective values does not improve (over exploratory moves). When there is no improvement in the vector space, the search ends. To better understand the TSP, consider the following example: exploratory moves are performed on the cities while taking into account the pattern moves and changing them by lowering the distance between the cities (objective value).

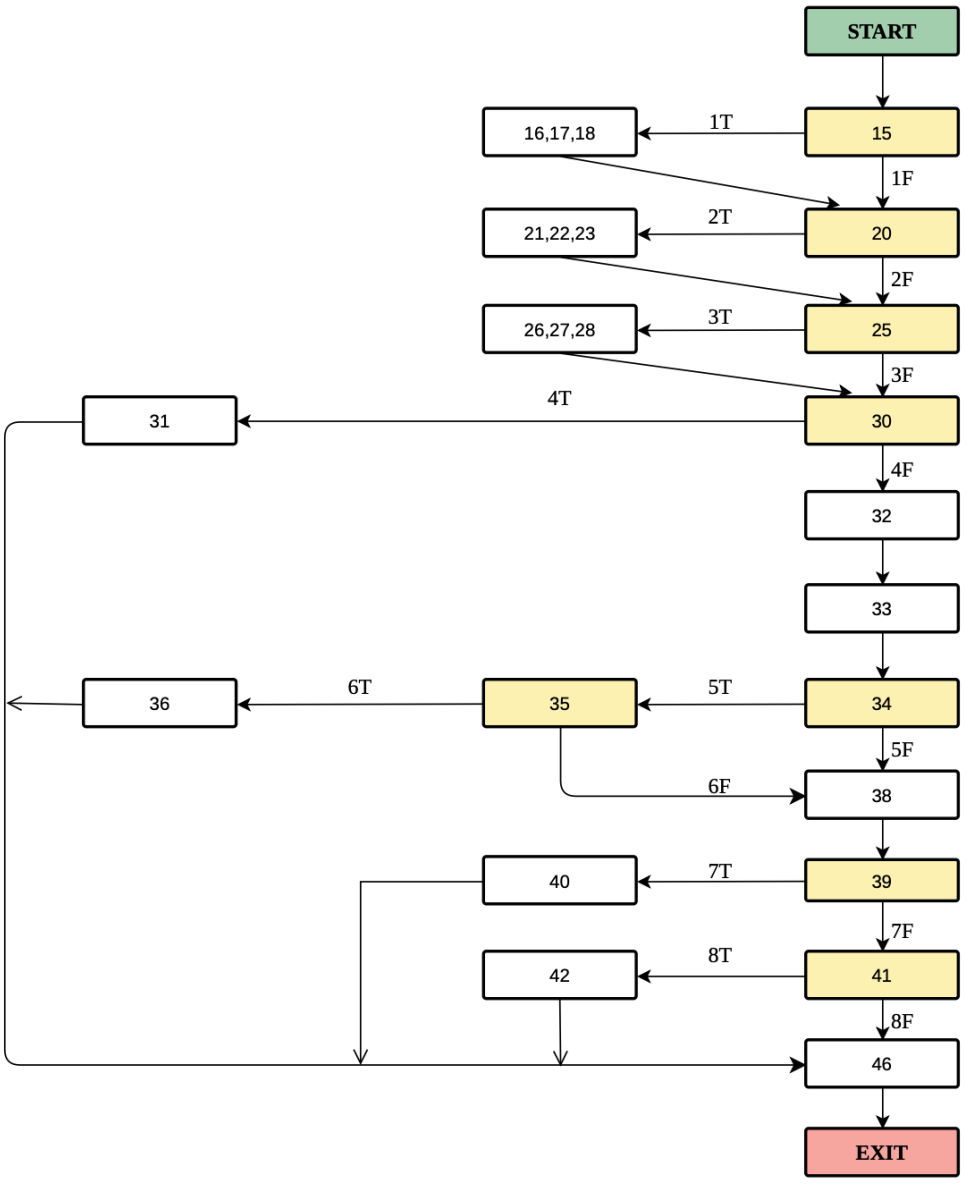
Steps:

1. Exploratory moves - Increases or decreases by the value of 1 of the variables in the vector.
2. Pattern moves - If Exploratory moves make improvement in objective values(+ve or -ve) direction is generated.
3. Pattern Movesof increasing size continue upon increase in objective value. If Pattern Move is not increasing, then search might have overshot the optimal value.
4. If Pattern Move>(current value of xi - optimal value) loop breaks and Exploratory Move starts from a new direction.

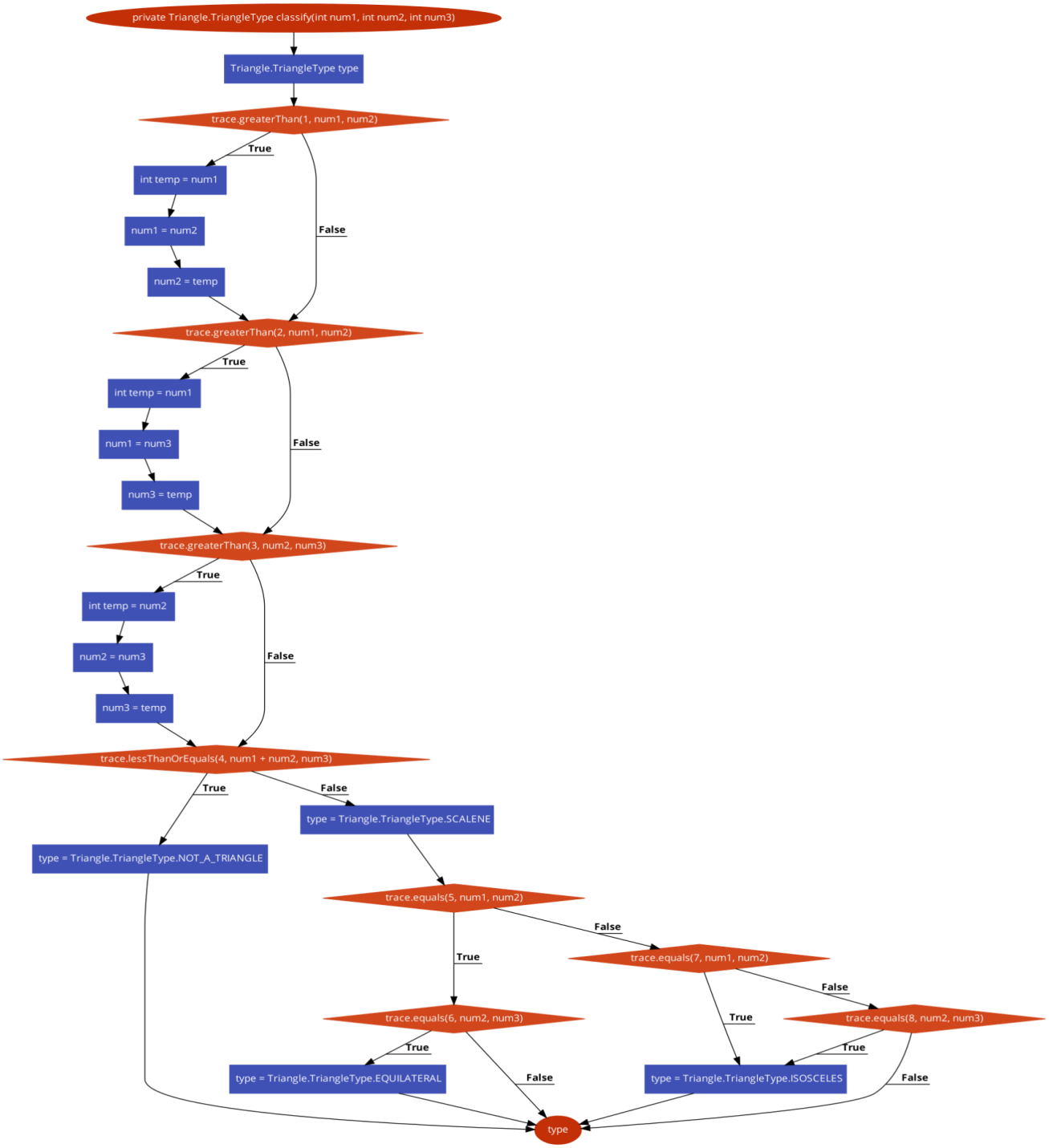
If Exploratory does not improve then the search starts with a new variable on the vector. There are 3 different searches in AVMF.

Iterative Search: When you overshoot, go in the opposite direction to discover the best solution. keep on until all resources have been used up or the desired result has been achieved.
Geometric Search: After overshoot, establish a bracket with an upper and lower limit, and then apply linear search to locate the vector that provides the best value.
Lattice Search (LS) is identical to geometric search, with the exception that it converges on the best solution by adding the Fibonacci number to steps that raise the value from the lower side of the bracket.

Question 2. Show the control-flow graph for the classify method of the Triangle class and label each branch with an id such that your branch ids are consistent with those specified in TriangleBranchTargetObjectiveFunction. Note that each branch id should be a number followed by T (for the true branch) or F (for the false branch).



**** LINE 40 won't be executed as it is dead code because the condition that is mentioned is already executed at line 35****



Question 3. Provide a test suite that achieves branch coverage for the classify method of the Triangle class.

Test Suite - A test suite is a container that contains a set of tests that testers can use to execute and submit test results.

Test Case 1: (230,230,230)
Branches Covered: 1F,2F,3F,4F,5T,6T

Test Case 2: (80,120,150)
Branches Covered: 1F, 2F, 3F, 4F, 5F, 7F, 8F

Test Case 3: (80, 100, 150)
Branches Covered: 1F, 2F, 3F, 4T

Test Case 4: (87, 87, 123)
Branches Covered: 1F, 2F, 3F, 4F, 5T, 6F, 7F
(Although it satisfies num1 == num2, since line 39 is a dead code it is not reachable)

Test Case 5: (67, 123, 123)
Branches Covered: 1F, 2F, 3F, 4F, 5F, 7F, 8T

Test Case 6: (152, 131, 147)
Branches Covered: 1T, 2F, 3T, 4F, 5F, 7F, 8F

Test Case 7: (347, 353, 137)
Branches Covered: 1F, 2T, 3T, 4T

Question 4.

Provide a smallest test suite that can achieve statement coverage for the code below. Does this test suite achieve branch coverage as well? If yes, for each test case in your test suite, specify the branches covered by the test case. Otherwise, provide a smallest test suite that can achieve branch coverage for this code.

```
Add (int a, int b) {
        if (b > a) {
           b = b - a
           Print b
        }
        if (a > b) {
           b = a - b
           Print b
        }
        if (a = b) {
           if (a = 0) {
              Print '0'
           }
        }
}
```

There are 4 conditions as we can observe from the above code: (a<b),(a>b),(a==b) and (a==0)

Let's consider a few test cases and infer from the observations/results.

Test Suite: (a,b)

Test Case 1: (5,10)

Branch(conditions) covered: (b>a)

Statements Covered: if(b>a), b = b - a,Print b

Test Case 2: (10,5)

Branch(conditions) covered: (a>b)

Statements Covered: if(a>b),b = a - b,Print b

Test Case 3: (0,0)

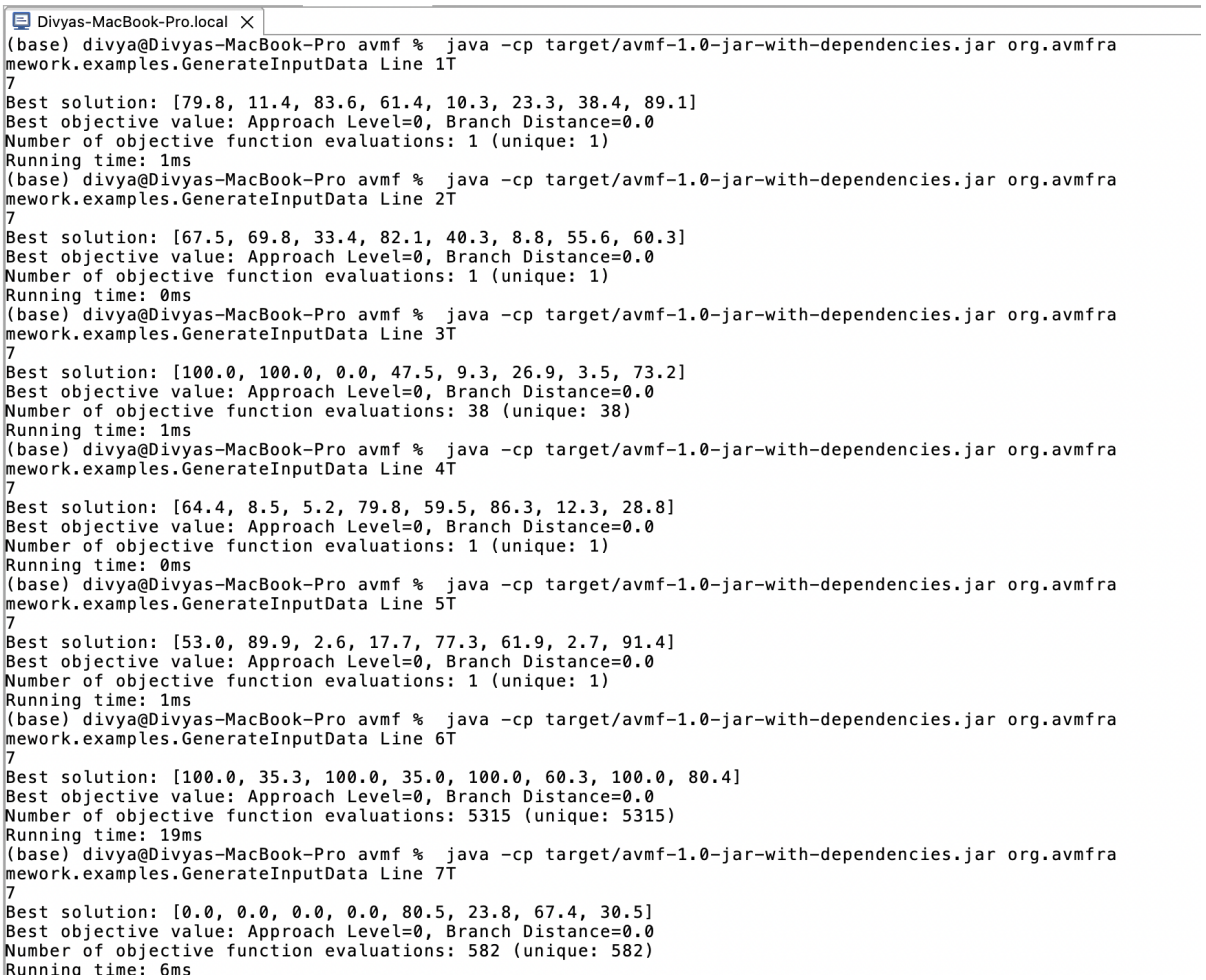
Branch(conditions) covered: (a==0)

Statements Covered: if(a==b),if(a==0) Print '0'

This is the smallest test suite that covers **all the statements** of the Add function. Yes it does branch coverage as well, since when either of test cases in our test suite execute the value of b becomes equal to a. The condition **a == b** is satisfied as **b and a both are 5** after either of the branches are executed but **a is not equal to 0**, so that's the False condition of the branch. Hence for the above considered test suite both the statements and branch are covered.

Question 5:
Provide a test suite for the intersect method of the Line class that achieves statement coverage, but not branch coverage. Explain which branches of the intersect method are not covered by your test suite.

Statement Coverage	Branches
26-31	1T,2T,3T,4F,5F
26-39	1T,2T,3T,4T,5T
43	6T
47	1F, 6F, 7T
50	1F, 6F, 7F



Question 6. The fitness function to compare different candidate test inputs is defined based on approach level and branch distance metrics. Explain how these two metrics are combined to compare different test input vector candidates and specify in which method of which Java class in the AVMf framework, this comparison is implemented.

- The approach-level and branch-distance metrics are mostly used to calculate the fitness function value.
- These are the number of branches leading to the test target that the test candidate did not execute, as well as the minimum distance required to meet the branch condition where the test candidate's execution has left the target's path.
- The approach level increases branch distance fitness, which links each branch to a level. Level is determined by the number of branches that separate a branch from the test target.
- F(i) = level(b) + normalized branch distance I b), where b is the branch of i.
- The two test parameters play an important role in determining whether a test data suit is optimal. Since many test cases may tie at this level, we use branch distance to break it.
- Normally, this value is adjusted between 0 and 1 to avoid the absolute value from being overpowered. It is usually calculated between two inputs.

  Code snippets : The “objective” folder in the “avmframework “/avmf/src/main/java/org/avmframework/objective” computes the metrics.

- DistanceFunction.java: To compute the distance between two numbers(nodes).
- NormalizeFunction.java – Uses alpha and beta to normalize branch distance.
- NumericObjectiveValue.java – checks the optimal and value using the “higherIsBetter” variable.
- Numbericobjectvalue.java ; Objectivefunction.java – monitor and evaluate the vector, objective value comparison.
- An AVM object keeps track of the candidate solution with the best objective value, the number of objective function evaluations completed, and other data using a

Monitor instance.In generating input data the implementation is being used: (GenerateInputData.java).

Question 7. As discussed in the class, one way to combine approach level and branch distance metrics is to first normalize branch distance and then add it to approach level. Why do we need to normalize the branch distance metric but not the approach level?

To avoid dominating the approach level, the branch distance is normalized and scaled in the range [0,1], whereas the approach level is an integer number.That is to say if the branch distance is not scaled and normalized, it can become arbitrarily large, outweighing changes in approach level. By normalizing the branch distance, minimizing the approach level has a greater impact on the fitness value and is automatically prioritized.
