Implementation:

1. compareCoverage Function:

I've converted the 'curr_metric' and 'total_metric' lists to sets and checked if any elements in the 'curr_metric' set are not present in the 'total_metric' set. If there are, it means that the coverage has improved (because new IR statements have been added) and returns 'True'. If all the IR statements in 'curr_matric' are already in 'total_matric', it returns 'False', indicating that the coverage has not improved.

2. updateTotalCoverage Function:

In this function, I've updated the 'total_matric' list by adding any IR statements from the 'curr_matric' list that are not already present in the 'total_matric' list. It ensures that the 'total_matric' list reflects the cumulative IR statements seen so far, and it returns the updated list.

3. mutate Function:

I've created 7 different mutation functions and calling them randomly.

The working of the mutation function is as follows:

Mutation 1:
Multiplying my input with a random number, ranging from -1000 to 1000.

Mutation 2:
Taking zor of input with a random number, ranging from -1000 to 1000.

Mutation 3:
Changing all the input values to 0.

Mutation 4:
Reversing the input value.

Mutation 5:
Changing input value to any random number ranging from -1000 to 1000.

Mutation 6:
Changing the sign of input values.

Mutation 7:
Taking “Bitwise AND” or “Bitwise OR” with any random number ranging from -1000 to 1000 consecutively.

Assumptions:

The initial assumption is that the input values for programs fall within a range of -1000 to 1000. Consequently, when mutations are applied, random values are selected within this range to ensure that the mutated inputs remain close to this boundary. This assumption is made to prevent extreme input values, as they could potentially lead to prolonged program execution times. For instance, if a program contains a statement like "forward :x," and the value of :x is exceptionally high, it might significantly prolong the program's runtime.

Limitations:

One limitation arises from the fact that the initial range of input values is constrained to -1000 to 1000. Consequently, mutations tend to stay within this range. When program conditions involve comparisons with significantly larger numbers, it becomes challenging to trigger those branches with these relatively small mutations.