**Optimization**

**Simulated Annealing**

**Lab**

Scenario:

You are managing a delivery company that needs to optimize the routes of its delivery trucks to minimize travel time and fuel costs. The delivery trucks need to visit a set of cities to deliver packages, and your goal is to find the most efficient route that visits each city exactly once and returns to the starting point.

Cities:

1. City A

2. City B

3. City C

4. City D

5. City E

Distances between Cities (in kilometers):

- A to B: 10

- A to C: 15

- A to D: 20

- A to E: 25

- B to C: 12

- B to D: 18

- B to E: 22

- C to D: 14

- C to E: 16

- D to E: 10

Follow the instructions below to implement simulated annealing on problem.

Step 1: Create a Python Class

Create a Python class named `DeliveryCompany` to encapsulate the problem and algorithm logic. This class will contain methods for initializing the problem, calculating distances, generating random solutions, and implementing simulated annealing.

Step 2: Initialize the Problem

Define the cities and the distances between them. Use this information to initialize the `DeliveryCompany` class, setting up the cities and their corresponding distances.

Step 3: Implement Solution Representation

Decide how to represent a solution (route) in your code. A route should represent the order in which they are visited.

Step 4: Calculate Total Distance

Write a method to calculate the total distance of a given route. Use the distances between cities to compute the total distance traveled.

Step 5: Generate Random Solution

Implement a method to generate a random solution (initial route). Shuffle the order of cities to create a random route.

Make a copy of cities, then shuffle them and return the new generated route.

Step 6: Acceptance Probability

Write a method to calculate the acceptance probability based on the current and new distances, as well as the temperature. This probability determines whether a new solution should be accepted. Use “math.exp ()” to calculate the probability.

Step 7: Simulated Annealing Algorithm

Implement the main simulated annealing algorithm. Start with a random solution and iteratively explore new solutions while gradually decreasing the temperature. Accept or reject new solutions based on the acceptance probability.

You must do this check before updating the current solution and current cost with the new one:

If acceptance\_probability(current\_distance, new\_distance, self.temperature) > random.random():

We do this check in order to make sure that the acceptance probability is greater than a random number. The idea is that even if the acceptance probability is relatively low, there's still a chance that the algorithm will accept a worse solution to explore the solution space more broadly. This allows the algorithm to escape local minima and potentially discover a better solution.

Step 8: Test the Implementation

Instantiate the `DeliveryCompany` class, run the simulated annealing algorithm, and print the optimal route and total distance.