Objective / Problem Statement

Given a set of N cities and distance between every pair of cities, find the shortest possible route that starts from a city and visits every city exactly once and returns to the starting city.
Write a program to find the shortest tour of N cities.

Intro to TSP

- This problem known as TSP i.e. Traveling Salesman Problem, involves finding the shortest possible route that a traveling salesman can take through a given set of cities, visiting each city exactly once and then returning to the starting city.
- The TSP is considered an NP-hard problem, which means that there is no known polynomial-time algorithm that can solve it.

Some Algorithms *:*

1. Brute Force Algorithm: This algorithm exhaustively enumerates all possible paths to find the optimal solution. It has a time complexity of O(n!) and only works for small instances of the problem.

2. Branch and Bound Algorithm: This algorithm uses a tree-based approach to eliminate partial solutions that are guaranteed to be suboptimal. It has a worst-case time complexity of O(n^2*2^n) and is more practical than the brute force algorithm.

3. Genetic Algorithm : This algorithm is a type of evolutionary algorithm that uses genetic operators such as mutation, crossover, and selection to generate a population of tours. The algorithm selects the best tours from the population to create the next generation and continues until a stopping criterion is met.

4. Simulated Annealing: This algorithm starts with an initial solution and iteratively moves to a new solution by allowing some "bad" moves initially, but gradually decreasing the likelihood of "bad" moves as the algorithm progresses. The algorithm terminates when the temperature (probability of accepting bad moves) reaches a certain threshold or when a maximum number of iterations is reached.

Initial Phase

When I started writing the code to solve this problem statement, I began with the Branch and Bound algorithm. However, it was taking more time than expected; for example, it took more than 300 seconds (expected time) to solve the problem for only 20 to 25 cities. As this was a very small number of cities, I decided to drop this algorithm.

Instead, I opted for the **optimized Simulated Annealing algorithm.** This is similar to the basic Simulated Annealing algorithm, but with differences in the cooling schedule and the selection of the initial temperature.

Steps

1. Define the initial solution: The initial solution is defined as a random permutation of the cities. This means that each city is visited exactly once, but the order in which they are visited is random.

2. Define the objective function to calculate the total distance traveled: This is done by summing the distances between each pair of consecutive cities in the solution, and adding the distance between the last and first city to complete the cycle.
3. Initialize the temperature: The temperature is initialized based on the average distance between cities. This value can be adjusted based on the problem and desired solution quality.
4. Define the cooling schedule: The cooling schedule determines how the temperature is reduced over time. The cooling schedule needs to balance exploration and exploitation. Initially, the temperature should be high to encourage exploration, and then gradually reduced to encourage exploitation. The cooling schedule can be based on a fixed temperature decrease per iteration or can be adaptive based on the acceptance rate.
5. Define the main loop: The main loop of the algorithm consists of generating new solutions by randomly swapping two cities, and accepting or rejecting them based on the acceptance probability. The acceptance probability is a function of the temperature and the change in the objective function between the current and new solution. If the new solution is better than the current one, it is always accepted. If it is worse, it is accepted with a certain probability that decreases as the temperature decreases.

   The loop continues until a stopping criterion is met, such as a maximum number of iterations or a minimum temperature.
6. Return the best solution: After the loop finishes, the best solution found during the algorithm is returned. This is the solution with the lowest total distance traveled.

Some Improvements:

1. Cooling Schedule: The cooling schedule is adaptive and adjusts the temperature based on the acceptance rate of the new solutions. This helps to ensure that the algorithm quickly converges to a good solution in the beginning stages and then slowly explores the search space to refine the solution.
2. Initial temperature selection: The initial temperature is set to a value that is twice the average distance between cities.

   This is a heuristic, I have seen online, that works well for TSP for a large number of cities.
3. The initial solution is generated using the numpy random permutation function, which is a faster and more concise method than using a for loop.

   I am using vectorized operations wherever possible for efficient implementations.