Assignment 6

June 20, 2024

Shah Dhruvanshi Kamlesh

EE22B143

Introduction In this assignment I have implemented Travelling salesman problem using Simulated Annealing method. The code starts with including standard libraries like `math`, `random`, `numpy`, and `matplotlib.pyplot`.

distance(cities, cityorder) function:

- This function calculates the total distance of a tour that visits cities in the order specified by cityorder.
- It computes the Euclidean distance between consecutive cities and sums them up to calculate the total tour distance.

tsp(cities) function:

- This function aims to solve the TSP for a given list of cities.
- It starts with an initial random order of cities(`current_ordeer`) and calculates the total distance(`current_distance`).
- It also sets the Initial Temperatore to 500 and cooling rate(or decay rate) to 0.999.
- It then iteratively attempts to find better solutions using a (modified) simulated annealing approach as follows:
  - It randomly swaps the positions of two cities in the tour and stores the new order in `new_order`.
  - If it's the first iteration, the variable `dist1` is set to the distance of the `new_order`. This variable is used to keep track of the distance at the beginning (initial distance), which will be used to calculate efficiency.
  - If the new tour has a shorter distance, or if a probabilistic condition based on temperature is met(the probablity given by `exp(-delta_distance / initial_temperature)` is less than a random probability), the new tour becomes the current tour.
  - Above idea comes from core of Simulated annealing since at Low temperatures, the probability will be less and we will have less chance to accept a solution and at hight temperatures(at the beginning) the probability will be significant and we can explore the space.
  - If the `current_distance` is less than `best_distance`, the current order is better, the `best_order` is updated to be a copy of the `current_order`, and the `best_distance` is updated to the new, shorter distance.

- The temperature is reduced (cooled) in each iteration to decrease the probability of accepting worse solutions as the algorithm progresses.
- The process is repeated for a specified number of iterations given by cube of number of cities, which makes the time complexity of the algorithm to O(n3).
- The optimal tour and its distance found during the process are returned.

Why to use Modified Annealing: In one iteration, modified annealing swaps only two cities and hence wouldn't affect the solution much. However, simple Annealinng generates a random order in every iteration, which can increase the chances that we don't end up even nearer to the solution, hence it is observed that modified annealing works better when number of cities is less ($<$ 100).

Loading the data from given file: The code reads the city data from a text file, which contains the number of cities and their coordinates.

Calling the tsp(cities) function: The tsp function is called with the loaded city data, and it returns the optimal order, optimal distance, and dist1 (the initial distance before any optimization).

Printing and Plotting the results: The code prints the initial and optimal distances and the optimal order of cities. It also calculates and prints the percentage improvement in distance and prints it.

The x and y coordinates of the cities are extracted from the cities list and ordered according to the optimal order. The code uses `matplotlib` to create a plot of the TSP tour, showing the order in which cities are visited.

The result: The following results are the result of iterations for `cooling_rate` = 0.999 and `initial_temperature` = 500. The percentage improvement is given by:

$$\frac{\text{Initial Distance} - \text{Final Distance}}{\text{Initial Distance}}$$

Iteration (1) Optimal Order: [34, 3, 21, 26, 18, 24, 6, 31, 27, 2, 7, 5, 0, 9, 36, 1, 16, 25, 15, 14, 11, 17, 19, 20, 30, 33, 13, 8, 35, 32, 4, 12, 29, 37, 28, 39, 23, 22, 10, 38] Optimal Distance: 6.990521860622793

The percentage improvement is: 67.09%

Iteration (2) Optimal Order: [22, 10, 38, 34, 19, 20, 30, 26, 18, 24, 33, 13, 31, 36, 1, 16, 25, 15, 39, 37, 29, 12, 5, 0, 9, 28, 4, 32, 7, 2, 35, 8, 27, 6, 21, 17, 11, 3, 14, 23] Optimal Distance: 6.278543861869096

The percentage improvement is: 69.29%

Iteration (3) Optimal Order: [19, 17, 34, 38, 10, 22, 23, 15, 28, 9, 0, 37, 36, 6, 31, 24, 18, 26, 21, 1, 16, 11, 3, 14, 25, 39, 29, 5, 12, 4, 32, 7, 2, 35, 8, 27, 13, 33, 30, 20] Optimal Distance: 6.412932511418584

The percentage improvement is: 70.12%

Iteration (4) Optimal Order: [3, 24, 18, 30, 20, 19, 11, 14, 39, 37, 29, 12, 4, 8, 33, 13, 27, 28, 9, 0, 5, 32, 7, 2, 35, 31, 36, 16, 25, 23, 15, 1, 6, 26, 21, 17, 34, 38, 10, 22] Optimal Distance: 7.975478186901545

The percentage improvement is: 61.33%

Iteration (5) Optimal Order: [29, 0, 9, 28, 37, 39, 23, 15, 14, 3, 11, 1, 16, 25, 34, 38, 10, 22, 36, 31, 6, 26, 21, 17, 19, 20, 30, 18, 24, 33, 13, 27, 8, 35, 2, 7, 32, 4, 12, 5] Optimal Distance: 6.060198362160585

The percentage improvement is: 72.31%

Iteration (6) Optimal Order: [29, 32, 7, 2, 35, 26, 18, 30, 33, 24, 13, 8, 27, 31, 37, 28, 39, 25, 14, 3, 11, 17, 1, 36, 16, 15, 23, 22, 10, 38, 34, 19, 20, 21, 6, 4, 12, 5, 0, 9] Optimal Distance: 7.339849934622857

The percentage improvement is: 63.44%

Iteration (7) Optimal Order: [36, 6, 31, 27, 8, 35, 7, 2, 13, 33, 24, 18, 30, 20, 26, 21, 1, 16, 39, 25, 17, 19, 11, 14, 15, 23, 3, 34, 38, 10, 22, 37, 29, 4, 32, 12, 5, 0, 9, 28] Optimal Distance: 6.924784865554092

The percentage improvement is: 65.74%

Iteration (8) Optimal Order: [25, 1, 16, 39, 28, 9, 0, 37, 29, 5, 12, 4, 32, 7, 2, 35, 8, 13, 27, 31, 36, 6, 24, 33, 18,

Iteration (8) Optimal Order: [25, 1, 16, 39, 28, 9, 0, 37, 29, 5, 12, 4, 32, 7, 2, 35, 8, 13, 27, 31, 36, 6, 24, 33, 18, 30, 20, 19, 17, 3, 34, 22, 10, 38, 21, 26, 11, 14, 23, 15] Optimal Distance: 6.268651877667061

The percentage improvement is: 71.34%

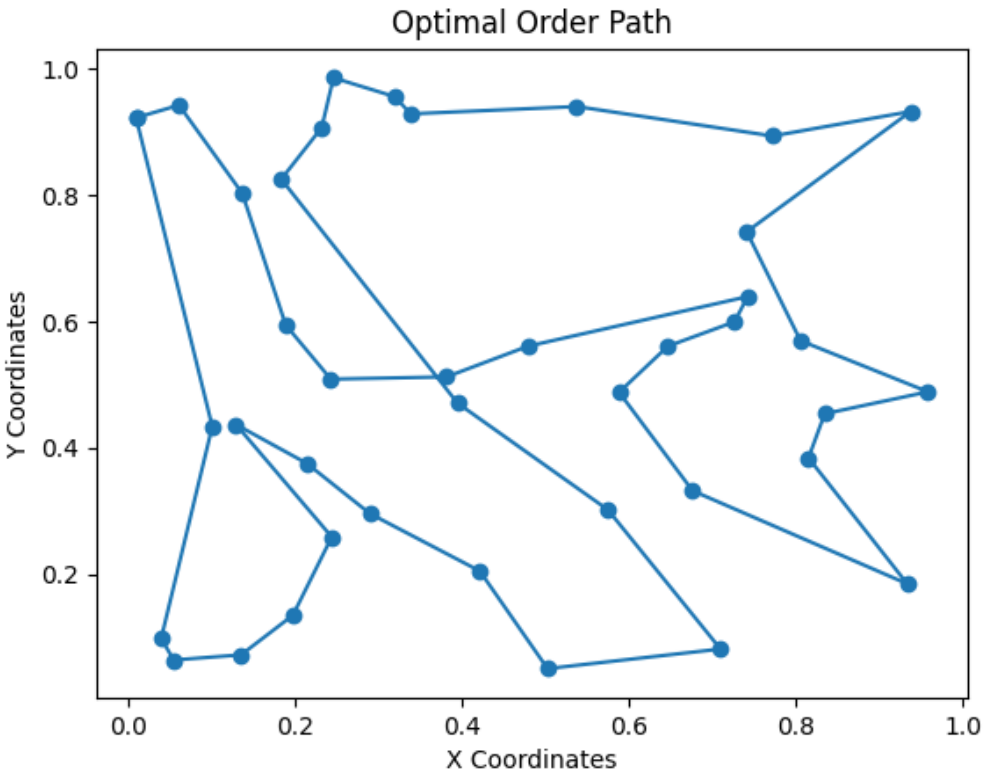
Iteration (9) Optimal Order: [3, 34, 22, 10, 38, 19, 20, 30, 33, 2, 7, 32, 4, 12, 5, 29, 37, 0, 9, 28, 6, 26, 18, 24, 13, 35, 8, 27, 31, 25, 39, 23, 15, 14, 17, 21, 36, 16, 1, 11] Optimal Distance: 6.737338638469761

The percentage improvement is: 68.27%

Iteration (10) Optimal Order: [9, 23, 22, 10, 38, 34, 3, 15, 14, 11, 17, 19, 20, 21, 1, 37, 29, 5, 12, 4, 32, 7, 2, 35, 13, 33, 24, 18, 30, 26, 6, 31, 27, 8, 36, 16, 25, 39, 28, 0] Optimal Distance: 6.7799925724078625

The percentage improvement is : 68.32%

The plot for the most optimised solution is given below:


Optimal Order Path
Y Coordinates
X Coordinates
0.0
0.2
0.4
0.6
0.8
1.0

[ ]: