ASSIGNMENT_6 DOCUMENTATION:

Simulated Annealing:

The Simulated Annealing algorithm is used to find an optimal solution. It starts with an initial order and iteratively explores new orders. It accepts new orders with a certain probability based on the temperature, allowing it to escape local minima.

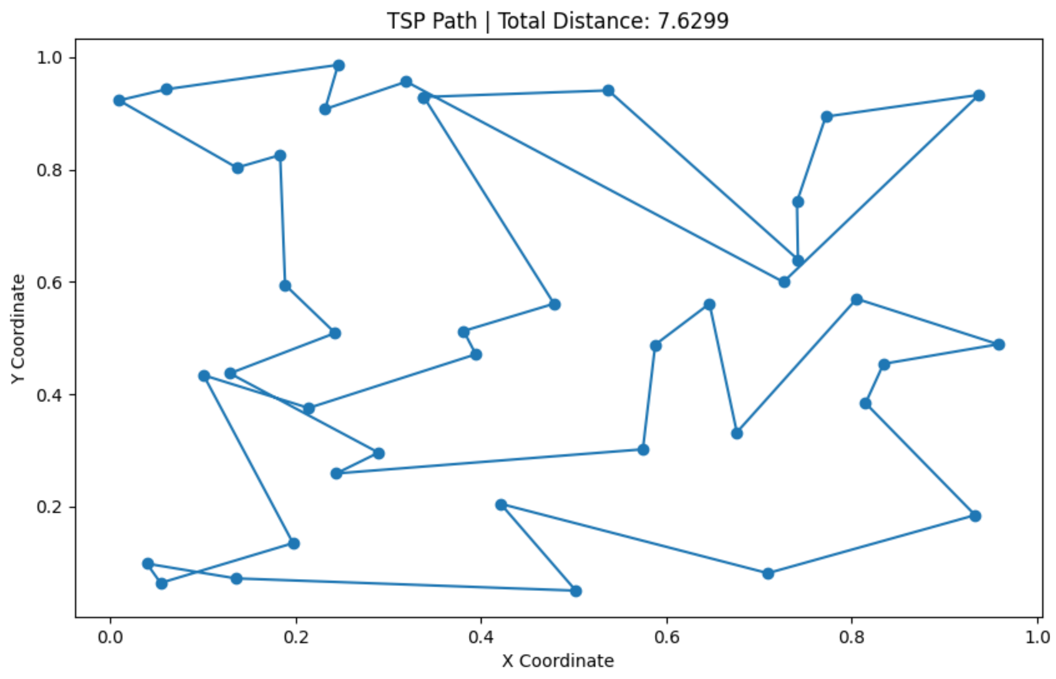
The key parameters for Simulated Annealing are:

temperature: Initial temperature for the algorithm.
cooling_rate: Rate at which the temperature decreases in each iteration.
max_iterations: The maximum number of iterations the algorithm will run.

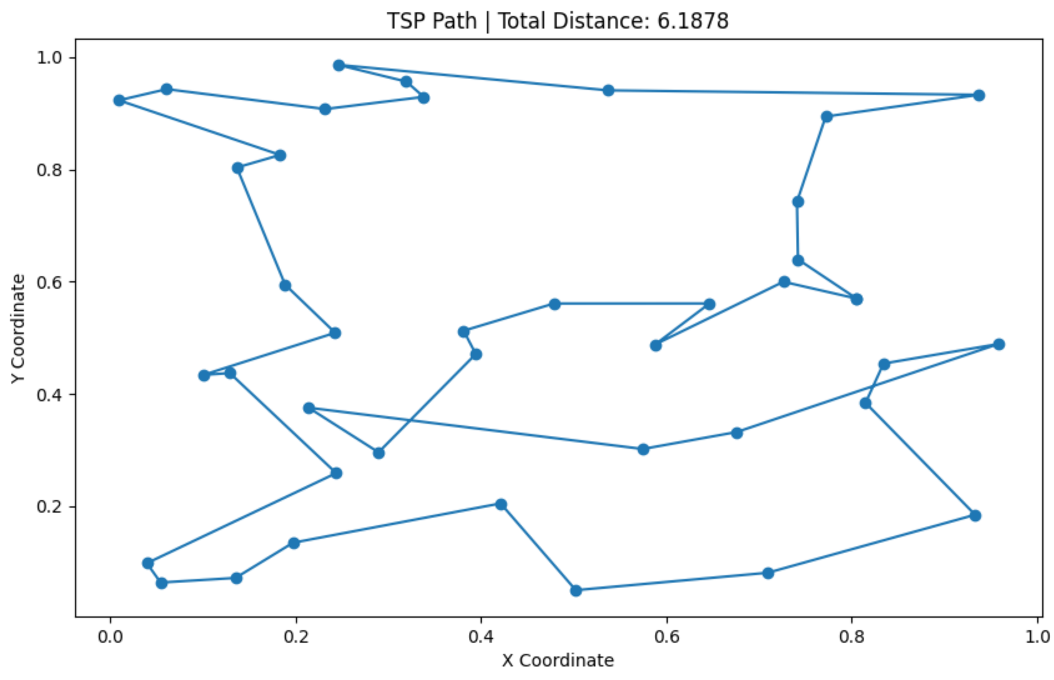
Herewith attached are the plots, results of certain iterations of the code and the code explanation with the function breakdown.

Plots and Results obtained:

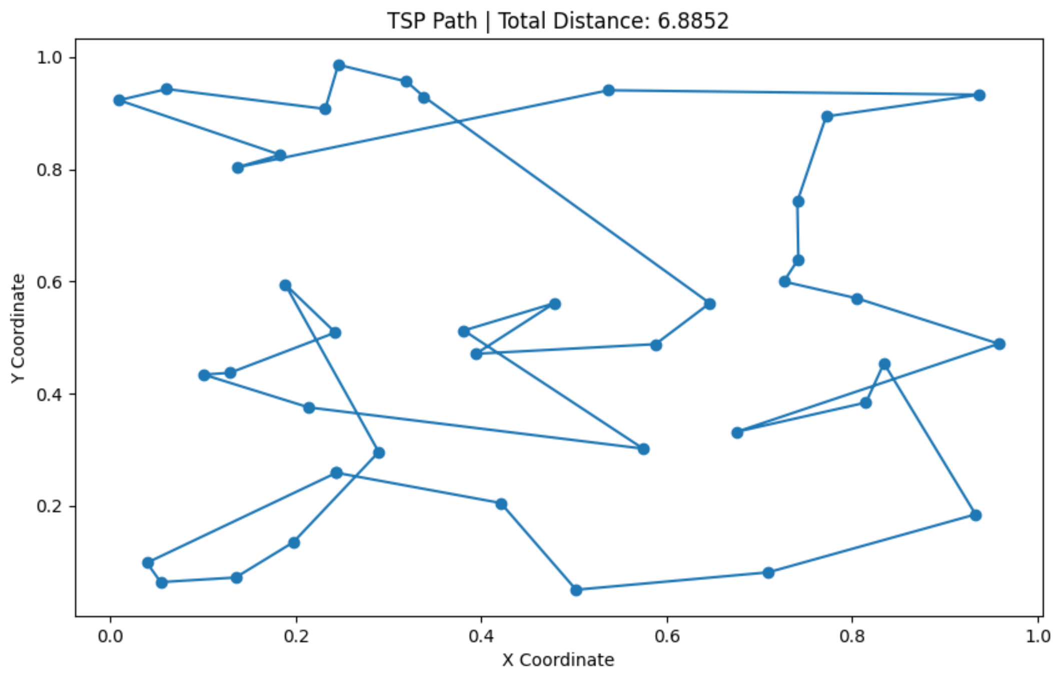
Order of visiting cities: [4, 32, 8, 35, 7, 2, 27, 12, 29, 5, 0, 9, 28, 37, 39, 25, 15, 11, 3, 21, 6, 31, 26, 13, 33, 24, 18, 30, 20, 17, 19, 38, 22, 10, 34, 23, 14, 1, 16, 36]
Total distance (Simulated Annealing): 7.629864702135186
Total distance (Random Initial Order): 20.72076463415638
Percentage Improvement: 63.18%



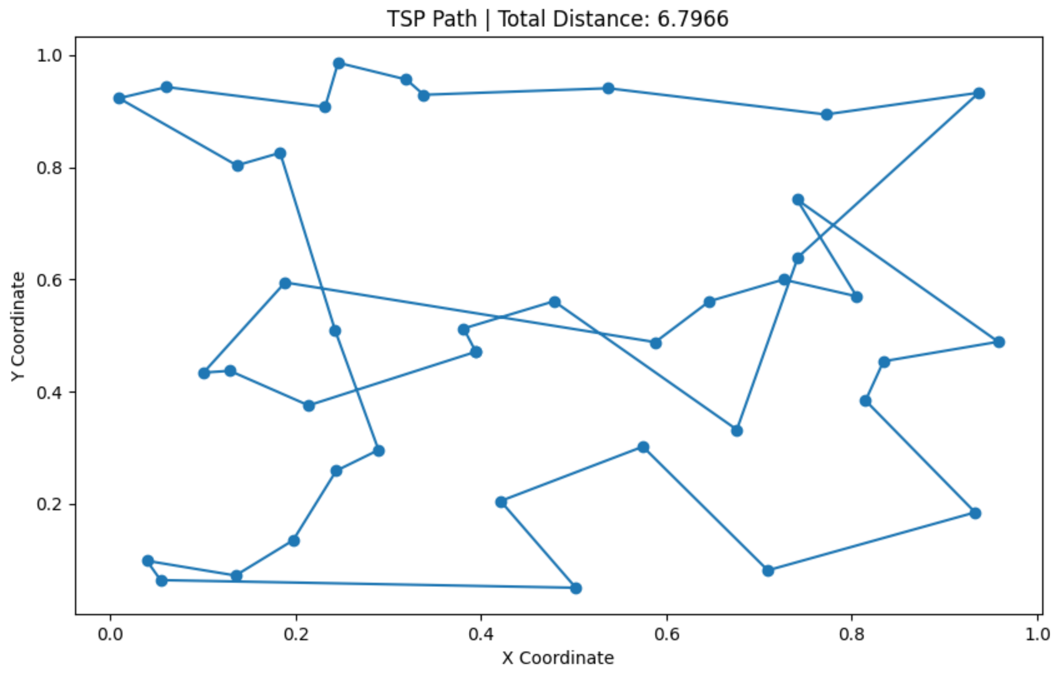
Order of visiting cities: [13, 8, 35, 7, 2, 32, 5, 12, 4, 29, 0, 9, 37, 28, 39, 25, 23, 15, 3, 22, 10, 38, 34, 17, 19, 20, 30, 18, 24, 33, 26, 21, 14, 11, 1, 16, 36, 31, 6, 27]
Total distance (Simulated Annealing): 6.187765043719344
Total distance (Random Initial Order): 21.608605701832314



Order of visiting cities: [3, 17, 19, 20, 30, 24, 18, 26, 33, 13, 27, 8, 35, 7, 2, 32, 28, 37, 9, 0, 29, 5, 12, 4, 31, 6, 1, 36, 16, 21, 14, 23, 15, 25, 39, 11, 34, 38, 10, 22]
Total distance (Simulated Annealing): 6.8851623741904655
Total distance (Random Initial Order): 20.236688124410843
Percentage Improvement: 65.98%



Order of visiting cities: [1, 16, 36, 26, 8, 2, 7, 32, 4, 12, 5, 29, 0, 9, 28, 37, 25, 11, 3, 34, 38, 22, 10, 19, 17, 21, 20, 30, 18, 24, 33, 35, 13, 27, 31, 6, 39, 23, 15, 14]
Total distance (Simulated Annealing): 6.796591511368154
Total distance (Random Initial Order): 20.36559995045867



Code explanation:

This code is a Python implementation of a solver for the Traveling Salesman Problem (TSP) using the Simulated Annealing algorithm which is a combinatorial optimization problem where the goal is to find the shortest possible route that visits a given set of cities exactly once and returns to the starting city. In this code, we calculate the order in which cities should be visited to minimize the total distance traveled.

Functions:

1. distance(city1, city2):

This function calculates the distance between two cities represented as (x, y) coordinates.

Parameters:
city1 (tuple): Coordinates of the first city.
city2 (tuple): Coordinates of the second city.

Returns:
The distance between city1 and city2.

2. total_distance(cities, cityorder):

This function calculates the total distance of a path that visits a list of cities in a given order.

Parameters:
cities (list of tuples): List of city coordinates.
cityorder (list): The order in which cities are visited.

Returns:
The total distance of the path that visits the cities in the specified order.

3. tsp(cities):

This function uses the Simulated Annealing algorithm to find an optimal order in which to visit the cities.

Parameters:
cities (list of tuples): List of city coordinates.

Returns:
best_order (list): The optimal order of visiting cities.
best_total_dist (float): The total distance of the optimal path.
random_total_dist (float): The total distance of a random initial order.
improvement_percentage (float): The percentage improvement achieved by Simulated Annealing.

The code reads city data from a file named 'tsp40.txt', where the first line specifies the number of cities, and subsequent lines provide the (x, y) coordinates of each city.

Results:

After running the Simulated Annealing algorithm, the code provides the following results:

The optimal order of visiting cities.

The total distance of the optimal path (Simulated Annealing).
The total distance of a random initial order.
The percentage improvement achieved by Simulated Annealing compared to the random initial order.

Collaborations:

Used multiple online resources to understand the algorithm of the code and fixing issues with implementations and sections of the code and took certain parts that were not working from them.