Travelling Salesman

M.Saketh Sai Ram - EE22B022

The Python Script `shortest_path.py` computes the Shortest possible path between the given cities such that the salesman visits all cities exactly once and returns to the starting point. It uses a simple heuristic approach called the **Nearest Neighbour Algorithm** and compares the result with a randomly generated city order.

- Import necessary libraries such as `numpy` for array handling, `sys` for accessing the input file through command line arguments, `matplotlib` for data visualization.

1 How to run the Code

- Ensure that your Input file with an extension `.txt` is placed in the "`ee22b022`" directory.
- Navigate the terminal to the "`ee22b022`" directory and run the command line given below to produce the desired output.

$\Longrightarrow$ `python3 shortest_path.py <Input File>`

2 Description of the Code

2.1 Reading City Coordinates

- The code reads the data from the given input file and stores the x and y coordinates of the cities in two lists x[ ] and y[ ].

2.2 Distance Computation Functions

distancebwpts(): It takes two cities as its arguments and calculates the distance between these two cities.
distance(): It calculates the total distance of a given order of cities. It iterates through the city order and sums up the distances between consecutive cities, including the distance between the last city and the first city.

2.3 Nearest Neighbour Algorithm

- The **tsp()** function solves the Travelling Salesman Problem using this algorithm.
- It initializes the **remaining_cities** list to keep track of unvisited cities.
- It randomly selects an initial city and appends it to the **tour** list while removing it from the **remaining_cities**.
- It continues to select the nearest unvisited city to the last added city and adds it to the **tour**, updating the **remaining_cities** until all cities are visited. This process constructs a tour order. At last, this function returns the shortest possible path found so far.

2.4 Creating a List of City Coordinates

The code creates a list named **cities**, where each element is a tuple containing the x and y coordinates of a city.

2.5 Random Path Generation

- First, it generates a random city order and computes the total distance required to visit all cities for this random order.

2.6 Running Nearest Neighbor Algorithm

- It runs the Nearest Neighbor Algorithm 50 times to try and find a better solution.
- This algorithm can be sensitive to the choice of the initial city. Depending on the initial city, the algorithm may lead to different solutions, which may not be the optimal solution. Let's consider two cities **A** and **B**, assuming the nearest city for **A** to be **B** among all the cities. But it is not necessary that **A** will be the nearest city to **B** and hence the path may not be the same for both cases. Therefore, it leads to different solutions.

2.7 Plotting

It plots the random path and the shortest possible path and outputs the total distance.

2.8 Percentage Improvement

This code calculates the percentage improvement achieved by the Nearest Neighbour solution compared to the random order by comparing the distances.

$$\text{Improvement} = \left(1 - \frac{\textit{Shortest Total Distance}}{\textit{Random Total Distance}}\right) \times 100\%$$

It has a time complexity of $\mathbf{O}(n^2)$ and becomes computationally intensive as the number of cities increases.

3 Observations and Conclusions

The Total Distance covered in the Random Path and the Shortest Possible Path found so far are tabulated as shown. These observations were made from the given input file `tsp40.txt`.

Observation No.	Total Distance in Random Path	Total Distance in Shortest Path	Percentage Improvement
1	23.639	5.704	75.8709%
2	20.284	5.704	71.8794%
3	20.711	5.762	72.1782%
4	19.728	5.944	69.8675%
5	20.827	5.703	72.6128%
6	23.848	5.762	75.8386%
7	21.249	5.704	73.1572%

City order for the path with a length of 5.703 can be [21, 26, 18, 24, 13, 27, 8, 35, 7, 2, 32, 4, 12, 5, 29, 37, 28, 0, 9, 39, 25, 15, 23, 14, 11, 3, 34, 38, 10, 22, 17, 19, 20, 30, 33, 31, 6, 36, 16, 1].

From the above observations, we can conclude that the total distance for the shortest possible path could be between 5.703 and 5.944.

4 Shortest Possible Paths