Title: Comprehensive Report - Solving the Half TSP Problem

Introduction:

The Half Traveling Salesman Problem (TSP) is a variant of the classic TSP where the goal is to find the minimum tour length that visits a subset of cities exactly once and returns to the starting city, while only considering half of the total cities. This comprehensive report describes an algorithm that solves the Half TSP problem using dynamic programming and Euclidean distance calculation. The objective is to minimize the total distance travelled. This report describes a Python implementation of a Half TSP solver using the Nearest Neighbour algorithm.

Code Overview:

The provided code consists of several functions and procedures that work together to solve the Half TSP problem and generate the output. Let's go through each component:

1) euclidean_distance(city1, city2): This helper function calculates the Euclidean distance between two cities represented by their coordinates. It uses the mathematical formula for Euclidean distance to calculate the straight-line distance between the cities.

2) nearest_neighbor(city, unvisited_cities): Another helper function, it determines the nearest unvisited neighbor of a given city from a list of unvisited cities. It iterates through each unvisited city, calculates the distance to the current city using euclidean_distance, and keeps track of the nearest neighbor found so far.

3) solve_half_tsp(cities): This is the main function that solves the Half TSP problem using the Nearest Neighbor algorithm. It takes a list of cities as input, where each city is represented by its coordinates. Here are the steps involved in this function:

a. It initializes some variables and data structures, including the number of cities and a 2D distance matrix to store the distances between each pair of cities.

b. It selects a random starting city from the list of cities and removes it from the list of unvisited cities.

c. It constructs the tour by iteratively finding the nearest neighbor of the current city using the nearest_neighbor function.

d. It adds the nearest neighbor to the tour, removes it from the list of unvisited cities, and updates the current city.

e. The process continues until half of the cities have been visited, resulting in a partial tour.

f. Finally, it adds the starting city at the end of the tour to complete the cycle.

g. It calculates the total tour length by summing the distances between consecutive cities in the tour.

4) read_input_file(filename): This function reads the input from a file specified by the filename parameter. It assumes that the file contains city data in the following format: each line contains the city ID, x-coordinate, and y-coordinate separated by spaces. The function reads the file line by line, extracts the city information, converts the coordinates to floating-point numbers, and appends the city to the list of cities.

5) write_output_file(filename, output): This function writes the output to a file specified by the filename parameter. It takes the output, which is a list of strings representing the tour length and the sequence of cities visited, and writes them to the file, separated by newline characters.

Execution:

The main part of the code performs the following steps:

a. It defines the input and output file names.

b. It reads the input from the file using the read_input_file function.

c. It solves the Half TSP problem by calling the solve_half_tsp function with the list of cities.

d. It writes the output to the file using the write_output_file function.

Conclusion:

The provided code offers a solution for the Half TSP problem using the Nearest Neighbor algorithm. It effectively calculates the tour by finding the nearest unvisited neighbor for each city, resulting in a partial tour that covers half of the cities. The code demonstrates a modular approach by separating the functionality into different functions, making it easy to understand, modify, and reuse.

Input Description:

The input provided to the code consists of city data, where each city is represented by its ID and coordinates. The input file contains multiple lines, with each line representing the data for a single city. The format of each line is as follows:

"city_id x_coordinate y_coordinate"

City ID:

The city ID is represented using the format "1", "2", "3…., etc.

It serves as a unique identifier for each city in the input data.

Coordinates:

Each city has associated x and y coordinates, which represent its position on a two-dimensional plane.

The coordinates are real-number values indicating the location of the city in the plane.

Output Description:

Here I am mentioning one of the results (Please see file output-1)

The output value of 1400 indicates the total tour length or the distance traveled in the Half TSP problem. This value represents the sum of distances between consecutive cities in the tour.

The tour is described as a sequence of cities, each represented by their coordinates (x, y). The tour starts and ends at the first city (280.0, 133.0) and visits all other cities in the specified order. The last city in the tour is the same as the starting city, forming a closed tour.

Based on the provided information, code successfully constructed a tour using the Nearest Neighbor algorithm and achieved a tour length of 1400.

Nearest Neighbor algorithm for the Half TSP:

The Nearest Neighbor algorithm is a simple and intuitive heuristic approach commonly used in solving the Traveling Salesperson Problem (TSP) and its variations, including the Half TSP. It starts with an arbitrary city and repeatedly selects the nearest unvisited city as the next destination. This process continues until all cities have been visited, resulting in a tour that visits each city exactly once.

Here's a step-by-step description of the Nearest Neighbor algorithm for the Half TSP:

1) Initialize the starting city: Select an arbitrary city to start the tour. This city can be chosen randomly or based on a predefined criterion. In the Half TSP, the starting city is typically the first city in the input list.

2) Mark the starting city as visited: Add the starting city to the tour and mark it as visited.

3) Select the nearest unvisited city: From the remaining unvisited cities, find the city that is closest to the current city in terms of Euclidean distance. This can be calculated using the Euclidean distance formula.

4) Update the tour and mark the selected city as visited: Add the selected city to the tour, remove it from the list of unvisited cities, and mark it as visited.

5) Repeat steps 3 and 4: Continue selecting the nearest unvisited city from the current city until all cities have been visited. This process forms a loop that iterates for half of the total number of cities.

6) Complete the tour: Once all the unvisited cities have been visited, add the starting city again at the end of the tour to complete the cycle.

7) Calculate the tour length: Compute the total length of the tour by summing the distances between consecutive cities in the tour. In the Half TSP, this length represents the distance traveled while visiting half of the cities.

The Nearest Neighbor algorithm is easy to implement and provides a reasonably good solution for the Half TSP. However, it does not guarantee an optimal solution and may sometimes produce suboptimal tours. Nonetheless, it is widely used due to its simplicity and efficiency, particularly for large input instances where exact algorithms may be computationally expensive.