CSE 2046

Analysis of Algorithms

Homework #3

Student Name: Osman Mantıcı

Description

TSP(Travelling Salesman Problem) is a np-hardness(non-deterministic polynomial time hardness) problem and this homework is about this problem that try to find an optimal algorithm as can do as. Let’s say there is a couple of cities and one salesman. Desired thing is the salesman would travel each city just once and come back to starting point without any repetition.

There are several approaches to find a solution such as genetic algorithm, brute force approach, greedy algorithm.

Genetic Algorithm: this algorithm using usually solving very complex problems or situations that optimal solution is difficult to find. This algorithm computes the fitness function for each member of the population and further it creates new individuals and it uses mutation to add randomization to the process. At the end algorithm selects a solution with the higher fitness function. However in order to use this algorithm it requires limitations like city properties or repetition issue.

Greedy Approach: this algorithm is a member of heuristic. It looks for the local optima and finds the global optimal solution that optimizing one of the best local solutions. This algorithm starts with edges and sorts all the edges. After that it finds the edge which has minimum cost. Algorithm continues the best of next choices. It has $O(n^2 \log_2 n)$ and algorithm not gives a guarantee for the optimal solution.

Nearest Neighbor Algorithm: this algorithm is finds a sub-optimal solution. Also I use this approach to my implementation. This algorithm starts with a city as beginning of a path and it finds the nearest city of that starting point and then adds the weight of that distance(which is minimum between other neighbors). Furthermore, now it finds minimum distance between its neighbors and goes on this manner until there is no city unvisited. This algorithm may the most straightforward TSP solution but maybe there is more optimal solutions waits for discovery.

Pseudo code of genetic algorithm:

```
1  Select a city as current city.
2  Find out the shortest edge connecting the current city and an unvisited city.
3  Set the new city as current city.
4  Mark the previous current city as visited.
5  If all the cities are visited, then terminate.
6  Go to step 2.
```

So the greedy algorithm maybe finds more optimal solutions than genetic algorithm and nearest neighbor algorithm when input size is small. It also maybe finds optimal what input is huge but the required time also groves hugely. So the nearest neighbor may not give the most optimal choice but it gives it fast and solution is close to the optimal. On the way of that thought I try to nearest neighbor approach.

In implementation of code I use the java language and create two class named “City” and “TSP”.

TSP is the main class of my program. It includes many methods and main method.

City class defines the instances of cities. This definition includes id, x and y coordinates, and Boolean “isVisited” information which specifies of visited or not. Also there is a method “distanceToCity” which is finds distance between two cities. Method has one parameter and it is also a city. A city calls this method in order to calculate parameter city how far than itself. This calculation done by a mathematical expression named distance between two points. It shown on this project document like:

$$d(c_1, c_2) = round\left(\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}\right).$$

TSP class includes a global variable named “totalDistance” and includes methods named “main()”, “readFileAndCreateCities()”, “createDistanceMatrix()”, and “tsp” and starts with main().

In main I create an array list structured “cities” list and this array list filled with readFileAndCreateCities(). In readFileAndCreateCities() method, input file is reading line by line, readed line splits and assigns the splitted[0] as city ID, splitted[1] as x coordinates of related city, splitted[2] as y coordinates of related city. After this information taken, a new city instance created and added to cities array list with line “cities.add(city)”. This method returns the cities array list.

So I have cities and their informations. Now I can construct adjacency matrix with the weights of ways. In addition information this point, I only calculates upper triangular of matrix because list is thought as an undirected graph. I create a 2D matrix named “distanceMatrix”. This matrix assigned to return value of createDistanceMatrix(cities, numOfCities). As it seen, function takes two inputs, cities array list and number of cities. This method calculates distances using City typed distanceToCity() in a nested for loop. First loop starts first read city from input file (because its ID=0), and second loop starts with city that in first loop. Because there is no necessary calculation, before i^{th} city, all of them already calculated. After i reaches to end of list, matrix construction also end. Therefore I have adjacency matrix of this graph.

Final part of my implementation is applying nearest neighbor algorithm on this adjacency matrix. There is an array named “path”and it assigned to return value of tsp(). tsp(numberOfCities, distanceMatrix, cities) takes three parameter and these are total count of cities, distanceMatrix that assigned right above paragraph, and list of cities. Function starts with a city and before beginning, there are “distance” and “minDistance” variables created. "distance” is for total distance travelled a tour, “minDistance” is for minimum distance of each city and its neighbors. I search each city one by one and look its neighbours in inner for loop and at the end of this loop I obtain the minimum distance between a city and its neighbours. I added this founded distance to total distance and add the city to the path array.

Finally I obtain a path and I printed it to an output file using “FileOutputStream” library. I assign default system output a “PrintStream” for this output file.

I check my results with “tsp-verifier.py”, verified solutions matched with each other.