Name: Krishna Bhatu

UID: 116304764

CMSC818B Homework-1

DESCRIPTION:

The problem given to us is called Traveling Salesman Problem(TSP). Where we are given with the graph of fully connected vertices with given edge costs and the robot has to find a tour visiting every vertex in the graph where the total length of tour (sum of all edge cost visited by robot) should be minimum.

This problem can be easily solved by Brute Force method (checking all the possible tours). With this method we will get optimal tour with lowest length. **But,** this algorithm is non-polynomial time , thus, it becomes computationally very expensive to implement.

Hence, we implement an algorithm which gives us near optimal solution and runs in polynomial time.

We use 2-approximation algorithm to solve this TSP.

This guarantees a solution to be not more then twice of the optimal solution.

Therefore, l(ALGO) <= 2 * l(OPT)

Following is the explanation of the implementation.

Note: We assume that for the given graph the triangle inequality holds for every vertex.

- Step 1

  Find the MST (Minimum Spanning Tree) of the graph.

  1. Create an empty list with the start vertex in it <explored list>(This will be out list of explored vertices). And create an empty vector<parent_list> (This will contain the MST [where every vertex will have a parent vertex of MST])
  2. Create a priority queue which sorts the keys (parent, child) tuple on the basis of the edge weights. Note: parent and child are start and end vertices of an edge
  3. Populate the priority queue with vertices connected to the vertex under exploration (It will be start vertex during first iteration)
  4. Select and remove the top element of the priority queue (vertex with the shortest path length).
  5. Check if the new vertex is not creating cycle (Check if the new vertex is not in the explored_list). If cycle is created then repeat 4.

6. If cycle is not created then Add this vertex to explored_list and add this edge in the MST (parent_list).
7. Assign this new vertex as the vertex under exploration.
8. Loop to 3. End loop if the all the vertex in the graph are explored.

- Step 2

  Depth First Search to find the tour

  1. Now we have the tree (parent-child list) of every vetrex in the for of minimum spanning tree.
  2. We, start with the root vertex (starting vertex) and keep on adding the child vertices in the tour until we find a leaf vertex(vertex without any child).
  3. Now, we go up in the direction of the root to find a vertex which has more child vertices other than the vertex includes in the tour.
  4. When we find one, again we keep adding the child vertex until we reach leaf.
  5. We repeat this until all the vertices in the graph are added to the tour. Thus we find the tour which has length no more that 2 times the optimal length.

With the following implementation, we get a tour with good enough tour length. But, in order to further improve the tour length, we implement a heuristic to bring the tour length close to optimal tour.

The heuristic used by our case is removing the path which has intersections.

- Step 3

  Removing the paths with intersection.

  1. Compare every edge in the tour with every other edge to find the intersection.
  2. If we find the intersection between two edges, we remove in the following manner

     Consider our tour to be;

     1-3-4-6-7-5-2-0-8-9

     And we find that the edge 3-4 intersects with edge 2-0.

     So, in order to remove the intersection, we swap the in-between sub tour.

     And, we get

     1-3-2-5-7-6-4-0-8-9
  3. Thus, we keep on removing the edges until all the edges are removed.

NOTE: In problems with large number of vertices (> 150), comparing all edges to check for intersection becomes expensive, so we set a limit on the number of intersections to solve and exit the loop when that limit is exceeded.

Link to Video showing implementation:

https://drive.google.com/open?id=1nZTX_3d5NkLbQCtl0ytnAL556KJ1B1N-

OBSERVATIONS:

1) Output for the given data set of vertices

Sr. No.	TSP Data Set	Length of Final Tour	Optimal Tour Length	Length of Tour Without Heuristics	Length of MST	Time Taken (in sec)
1.	eil51	516.99	426	611.90	376.49	0.01
2.	eil76	605.64	538	688.76	472.33	0.04
3.	eil101	750.72	629	814.194	562.25	0.09

2) Output for randomly generated set of vertices

Sr. No.	Number of randomly generated vertices	Length of Final Tour	Length of Tour Without Heuristics	Length of MST	Time Taken (in sec)
1	100	887.78	960.63	661.739	0.006
2	100	961.23	994.53	660.29	0.006
3	100	880.37	993.86	662.313	0.007
4	100	881.36	974.73	672.35	0.007
5	100	952.52	1020.98	649.62	0.009
6	100	950.16	1079.58	710.5	0.01
7	100	888.51	989.46	674.72	0.008
8	100	898.91	1002	676.33	0.008
9	100	903.18	959.013	694.722	0.005
10	100	860.061	979.664	686.045	0.007
1	200	1271.75	1365.26	961.828	0.04
2	200	1346.38	1504.42	953.57	0.07
3	200	1243.03	1380.2	939.323	0.06

4	200	1313.95	1446.18	946.25	0.05
5	200	1265.22	1399.72	969.85	0.05
6	200	1260.43	1385.95	952.143	0.05
7	200	1283.56	1440.43	934.55	0.06
8	200	1283.23	1402.95	961.83	0.04
9	200	1232	1311.07	909.53	0.03
10	200	1247.9	1359.56	925.405	0.05
1	300	1674.39	1760.06	1140.78	0.9
2	300	1700.26	1747.32	1154.59	1.32
3	300	1801.05	1832.01	1187.49	1.05
4	300	1721.85	1751.69	1173.22	0.44
5	300	1729.83	1782.89	1172.9	1.04
6	300	1734.42	1756.46	1181.66	0.25
7	300	1690.69	1718.99	1140.17	0.86
8	300	1777.09	1791.46	1154.17	0.7
9	300	1750.2	1799.71	1186.78	0.8
10	300	1673.25	1703.04	1158.06	0.6

Answer 2:

Now, the question is for k-robots, performing the task of tour selection where all the robots have to start from the same point and will end at the same point.
The condition is that all the vertex of the graph must be included in the tour at least once.
This means that the optimal tour length fro k-robots will be same as that of optimal tour length for one robot.

Following is the algorithm that I propose for this problem:

1. Remove the start vertex from the vertex list and select k random vertex where k is the number of robots.
2. Assign these randomly selected vertex to the robot ids. Thus each random vertex selected indicates a cluster center for their respective robot.
3. Now, iterate through all the vertices and on the basis of nearest neighbor (shortest euclidean distance) assign these vertices to their respective robot cluster.
4. Now with all the vertices in the cluster, calculate the mean and assign the nearest vertex to that mean as our new cluster center. Do this step for all(k) the cluster centers.
5. Terminate this process when the there is no change in cluster centers. (Thus, we have successful y classified the vertices into clusters for each robot to find tour)
6. Now, for selecting the root vertex for tour, we select the vertex in the cluster closest to the start vertex. Do this for every cluster(robot).

7. Now, whit this root vertex, find the MST for every cluster and then using DFS, find the near optimal tour for all the robots.
8. Link the tour with the start vertex, which will ensure that the tours are starting and ending at the same vertex for all robots.

Pseudocode for the algorithm will be:

```
vertices → Data structure holding the list of vertices in the graph
start_vertex = (x,y)
vertices.remove(start_vertex)
vertex_class → A dictionary to hold all the vertex as key and the cluster number as value
cluster_center → List of cluster centers where the index number is robot id
for (k robots){
        temp = random(vertices) → random function gets random vertex from list
        cluster_center[k] =  temp
        vertex_class[temp] = k
}

change = false
while(!change){
        for(i vertex){
                minimum = Inf
                temp_class = -1
                for(k robots){
                        if(euclidean_dist(vertices[i], cluster_center[k]) < minimum ){
                                minimum = euclidean_dist(vertices[i], cluster_center[k])
                                temp_class = k
                        }
                }
                vertex_class[vertices[i]] = k
        }
        sum_of_vertex_in_class → A list which holds the sum where index is robot id
        number_of_vertex_in_class → A list which holds the number where index is robot id
        for(j vertex){
                current_class = vertex_class[vertices[j]]
                sum_of_vertex_in_class[current_class] += vertices[j]
                number_of_vertex_in_class[current_class] += 1
        }
        mean_of_class  → A list which holds the mean where index is robot id
        for(k robots){
                 mean_of_class[k] =  sum_of_vertex_in_class[k]/ number_of_vertex_in_class[k]
        }
        previous_centers = cluster_center
        cluster_center = mean_of_class
        if(cluster_center  == previous_centers){
                change = true
        }
}
//Now, we have k cluster of vertices, hence k graphs
```

graphs → Method to add the vertices to a list of graph where index is robot id
//Apply MST + DFS to individual graphs as solving individual k- problems.
//The implementation of MST and DFS will be same as the code submission.

Reasoning:

- This algorithm was selected because it ensures us that the length of tour will always be less than c * l(Optimal) where c is an integer.
- Also, this method will give us the freedom to implement algorithms to improve the tour length for one robot which in turn will improve the tour length of all robots thus improving the total tour length.
- The complexity of this algorithm is polynomial so it is **not** computationally very expensive.

As, we are dividing the graph in cluster, thus we are essentially breaking the connections which are in-between two clusters.
So, $l(MST1) + l(MST2) \ldots + l(MSTk) <= l(MST_{entire})$ ….k is the number of robots

By the current 2-approximation algorithm we know that;
$l(ALGO) <= 2*l(MST) <= 2*l(OPT)$

Hence;
$l(ALGO1) + l(ALGO2) \ldots. + l(ALGOk) <= 2*k*l(OPT) = c*l(OPT)$

Thus, the worst case scenario will be 2*k*l(OPT). In which, every cluster will have MST such that there are is only one parent vertex and rest all are leaf vertex.

Following method is a little expensive then 1 robot, but it will help cover the region in 1/k th of the time then 1 robot.

Other advantage of this method is that this process can be parallelized very easily.