AZERBAIJANI-FRENCH UNIVERSITY (UFAZ)

UFAZ UE709 Network and Algorithms

Final Project

Presented by: Afrasiyab KHALILI
Ughur AGHAKISHIYEV
Javid HUSEYNOV

January 3, 2019

Contents

0.1 INTRODUCTION

0.1.1 Travelling Salesman Problem

- The travelling salesman problem (also called the travelling salesperson problem[1] or TSP) asks the following question: "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city?" It is an NP-hard problem in combinatorial optimization, important in operations research and theoretical computer science.

0.1.2 Why NP-hard?

First, let us understand the definition of NP- hard. NP (nondeterministic polynomial time) problem is the one whose solution could be verified in polynomial time but the problem is not guaranteed to be solved in polynomial time. Now, NP - hard problem are those which are atleast as hard as any NP problem. NP complete problem is the class of problems that are both NP and NP-hard.

Let us now check both the conditions.

1. **NP**
   To control that given solution is the right one for a TSP problem. We should verify two things: Firstly, each city must be visited exactly once. (could be done in polynomial time) Secondly, there is no shorter route than the current solution. (This cannot be guaranteed in polynomial time) Hence, TSP is not NP.

2. **NP Hard**
   Surely TSP is a NP hard problem. (For, even it's solution can't be guaranteed in polynomial time)

   Thus, TSP belongs to the class of NP-hard problem and not NP-complete.

   Brute force approach for TSP will need all possible paths to be calculated which is (n-1)! paths(where n is the number of cities). As n increases, it is computationally not feasible to compute that many paths.

   There are certain approximation algorithms for TSP which guarantees to solve the problem in polynomial time at the cost of solution not being exact. Christofides algorithm, is one such heuristics approach which guarantees it's solution to be within a factor of

1.5 of the optimal solution. By far, Christofides algorithm (time complexity : O(n$\hat{3}$)) is known to have the best approximation ratio for a general TSP problem.

0.1.3 Christofides algorithm

The Christofides algorithm is an algorithm for finding approximate solutions to the travelling salesman problem, on instances where the distances form a metric space (they are symmetric and obey the triangle inequality).It is an approximation algorithm that guarantees that its solutions will be within a factor of 3/2 of the optimal solution length, and is named after Nicos Christofides, who published it in 1976. As of 2019, this is the best approximation ratio that has been proven for the traveling salesman problem on general metric spaces, although better approximations are known for some special cases.

1. Create a minimum spanning tree T of G. (**Prim's or Kruskal algorithm**)

    - We have used the Kruskal algorithm and implement it in our program in order to find the MST.

```
def genMinimumSpanningTree(G):
        MST = nx.Graph()
        eLen = len(G.edges())
        vLen = len(G.nodes())
        mst = []
        mstFlag = {}
        for i in [ (u, v, edata['length']) for u, v, edata in G.edges(data
          mstFlag[i] = False

        parent = [None] * vLen
        order = [None] * vLen
        for v in range(vLen):
          parent[v] = v
          order[v] = 0
        while len(mst) < vLen - 1 :
          curr_edge = getMin(G, mstFlag)
          mstFlag[curr_edge] = True
          y = findRoot(parent, curr_edge[1])
      x = findRoot(parent, curr_edge[0])
```

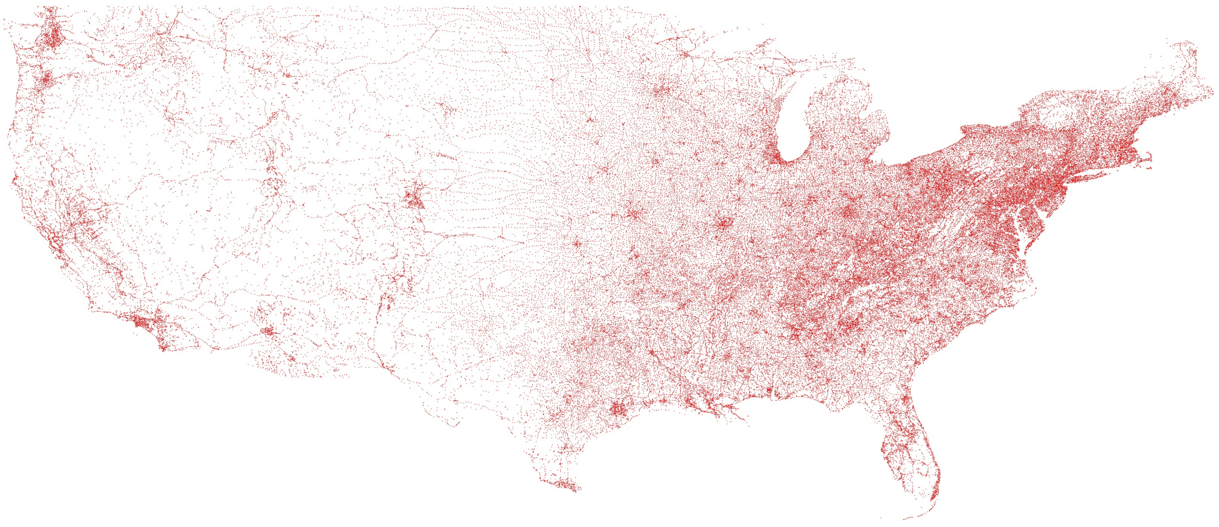
```
   if x != y:
      mst.append(curr_edge)
            union(parent, order, x, y)

for X in mst:
   if (X[0], X[1]) in G.edges():
            MST.add_edge(X[0], X[1], length = G[X[0]][X[1]]['length'])
return MST
```

2. Let O be the set of vertices with odd degree in T. By the **handshaking lemma**, O has an even number of vertices.
3. Find a minimum-weight perfect matching M in the induced subgraph given by the vertices from O.
4. Combine the edges of M and T to form a connected multigraph H in which each vertex has even degree.
5. Form an **Eulerian circuit** in H.
6. Make the circuit found in previous step into a **Hamiltonian circuit** by skipping repeated vertices (shortcutting).

0.2 INPUT

0.2.1 Data visualization



Original data set was taken by data_set_in_usa

As you can see from the original data, it is too huge to work on it since we are utilizing Christofides algorithm. So, we took 20 samples of the cities from the data with three attributes (id number of city, x and y coordinates).

Sample:

1 33613.158800 86118.306100
2 33100.954000 85529.675300
3 31571.835200 85250.489300
4 32070.429900 85687.725700
5 33290.392600 87198.053000
6 33356.221700 86806.933300
7 32105.423800 86276.354700
8 32609.020600 86311.082800
9 33883.988500 86644.713500
10 32121.536600 87906.122700
11 32876.516500 87742.510500
12 34022.319400 86045.528500
13 31126.292500 85072.429400

```
14 31277.947300 86125.779600
15 34267.593700 86208.866900
16 33638.994100 86965.273400
17 33707.052700 87241.946200
18 32944.012000 85953.853200
19 34178.984000 85512.181800
20 33129.568100 88151.416600
```

However, since our algorithm requires a different kind of input format we have implemented a small script to convert this eucl_2d data to our format. The script tsp2format.py converts this data to a N by N matrix by calculating the euclidean distance between each city. Each instance of this matrix at the indices [i][j] represents the distance between ith and jth city. Such expansion in the size of our data is another reason for reduction of the original dataset's size.

At first we tried to convert the whole data-set but after waiting a lot and a bit of calculation we realized that the size of the converted data (115475x115475) will be more than 12.3 GB and requires a lot of time to compute with

$$O(n^2)$$

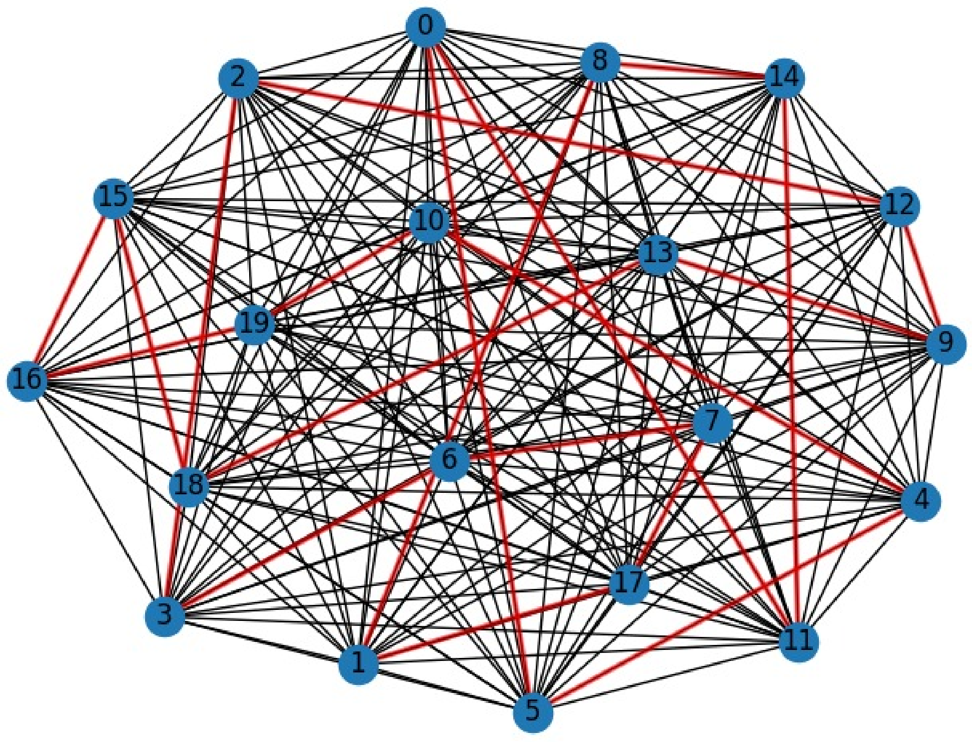
complexity. Hence, we settled for a smaller size in order to enable us to display graphs with ease.

0.3 Output

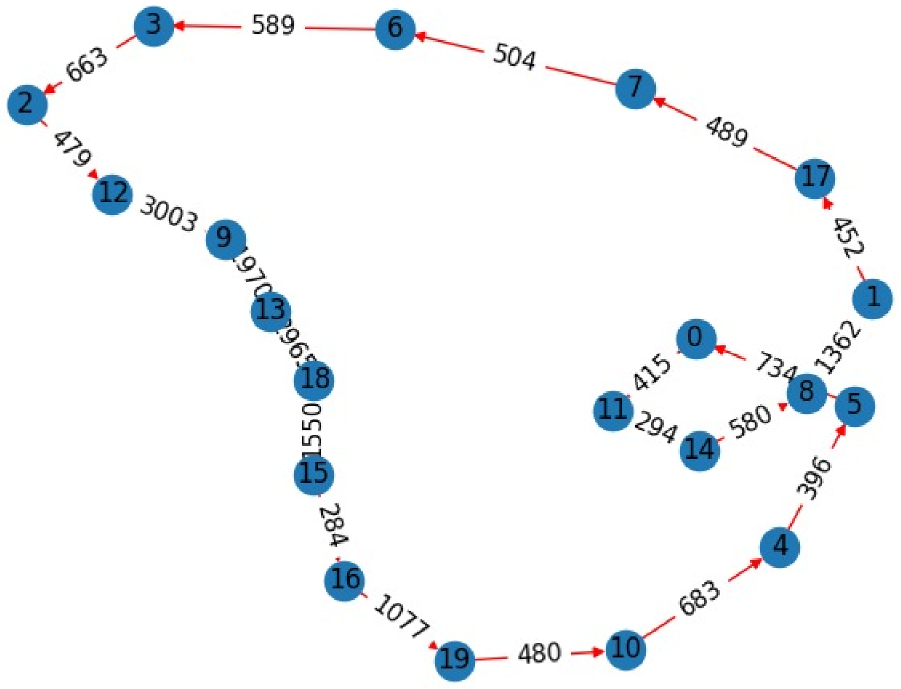
0.3.1 Data visualization

When you run the program, you will face with 2 plots. The first one will demonstrate all the connections between cities and second one will display the shortest path. Let's look at the sample outputs from the program:

1. The graph which represents connection between cities:



2. The graph which represents the shortest path:


3
589
6
504
7
663
2
489
17
479
12
3003
9
452
970
1
13
965
0
18
415
734
1362
8
5
11
294
580
1550
14
15
396
284
4
16
1077
683
19
480
10