Travelling Salesperson Problem – Closest Edge Insertion Heuristic

Lindie Chenou
[Computer Science & Engineering]
Speed School of Engineering
University of Louisville, USA
lcchen01@louisville.edu

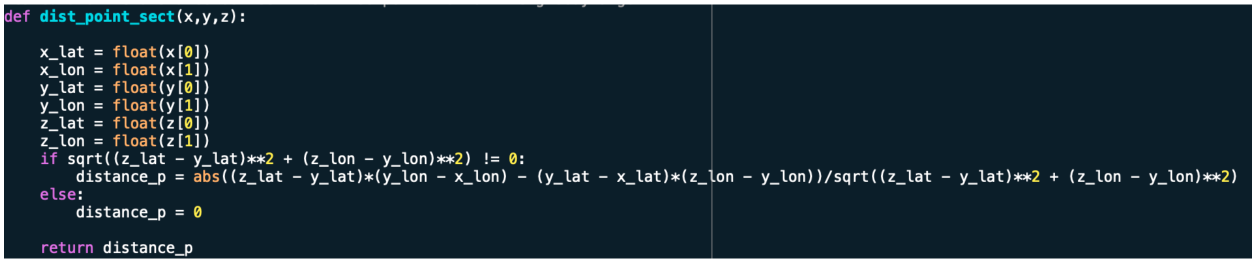
1. **Introduction** (What did you do in this project and why?)

   The goal of this project is to calculate the optical cost of visiting each city in a data file. The algorithm that is supposed to be used to accomplish this problem for this project is the greedy algorithm. The problem is best known as A Traveling Salesperson Problem (TSP). TSP is an NP-complete problem. Depending on how many nodes or cities are given, the salesman must visit each one using the minimum cost it will take to get it done.

2. **Approach** (Describe algorithm you are using for this project)

   A greedy algorithm is an algorithm that follows the problem-solving heuristic of making the locally optimal choice at each stage. For the TSP function the greedy heuristic follows as that at each step of the journey of the nodes, it visits the nearest unvisited node. This doesn't mean it's going to be the best solution or have the minimum cost of the path. What greedy does is eliminate the unnecessary step it will take other algorithms to get through all the nodes and find the minimum cost.

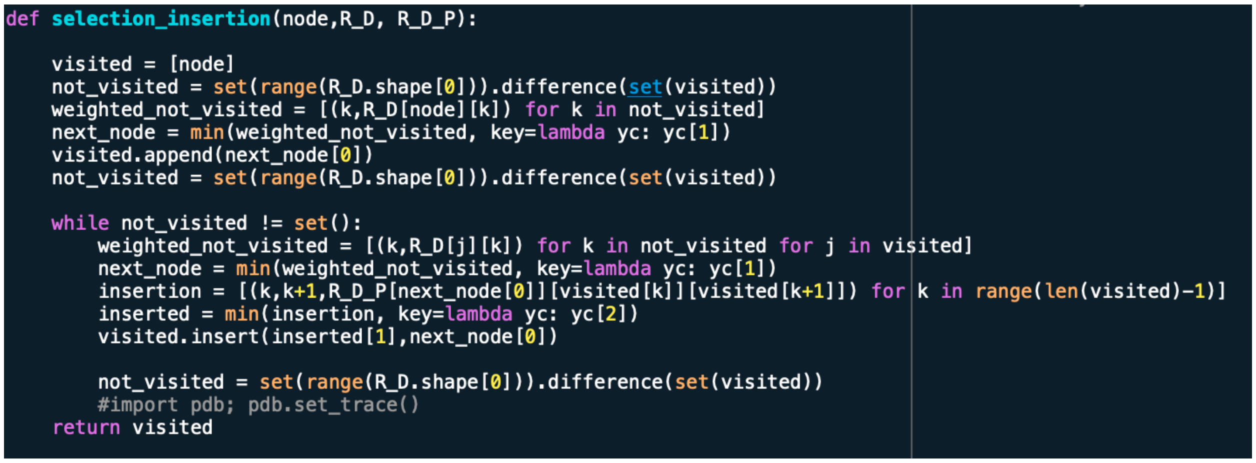
Most of the time greedy algorithm fails to find the optimal solution. The reason being is that it never considers all of the data on file. For the next node it's expected to choices it looks only at the nodes it has so far not looked at future nodes coming.



```python
def dist_point_sect(x,y,z):

    x_lat = float(x[0])
    x_lon = float(x[1])
    y_lat = float(y[0])
    y_lon = float(y[1])
    z_lat = float(z[0])
    z_lon = float(z[1])
    if sqrt((z_lat - y_lat)**2 + (z_lon - y_lon)**2) != 0:
        distance_p = abs((z_lat - y_lat)*(y_lon - x_lon) - (y_lat - x_lat)*(z_lon - y_lon))/sqrt((z_lat - y_lat)**2 + (z_lon - y_lon)**2)
    else:
        distance_p = 0

    return distance_p
```



```python
def selection_insertion(node,R_D, R_D_P):

    visited = [node]
    not_visited = set(range(R_D.shape[0])).difference(set(visited))
    weighted_not_visited = [(k,R_D[node][k]) for k in not_visited]
    next_node = min(weighted_not_visited, key=lambda yc: yc[1])
    visited.append(next_node[0])
    not_visited = set(range(R_D.shape[0])).difference(set(visited))

    while not_visited != set():
        weighted_not_visited = [(k,R_D[j][k]) for k in not_visited for j in visited]
        next_node = min(weighted_not_visited, key=lambda yc: yc[1])
        insertion = [(k,k+1,R_D_P[next_node[0]][visited[k]][visited[k+1]]) for k in range(len(visited)-1)]
        inserted = min(insertion, key=lambda yc: yc[2])
        visited.insert(inserted[1],next_node[0])

        not_visited = set(range(R_D.shape[0])).difference(set(visited))
        #import pdb; pdb.set_trace()
    return visited
```

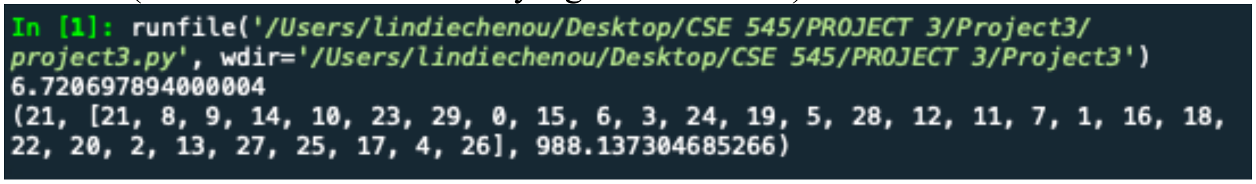
The code above is how the greedy algorithm is defined for this project. Step 1 is to start with a sub-graph consisting of node x only. Step 2 find the next node such that it has a minimum path to. Step 3, since the line was created with the x and y node, next the insertion heuristic node needs to be added in. the dist_point_sect function is to calculate the closest node to the line that is to be inserted. The first part of the selection_insertion function is where the first 2 steps are executed. The while loop of the selection_insertion is where part 3 is performed, it will be repeated until all of the nodes in the files have been visited. By performing these steps, the time complexity for this algorithm will be O(n2).

3. **Results** (How well did the algorithm perform?)

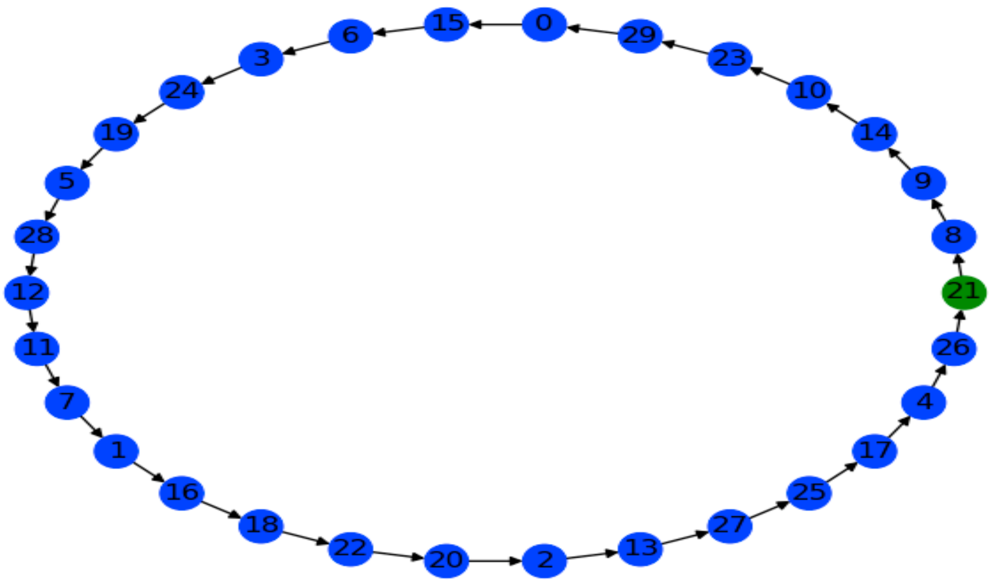
3.1 Data (Describe the data you used.)

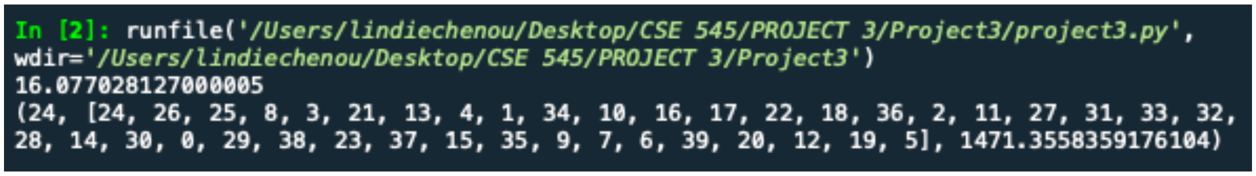
Two sets of .tsp files were given to execute this project. Random30.tsp and Random40.tsp were given to them having 30 and 40 cities.

3.2 Results (Numerical results and any figures or tables.)

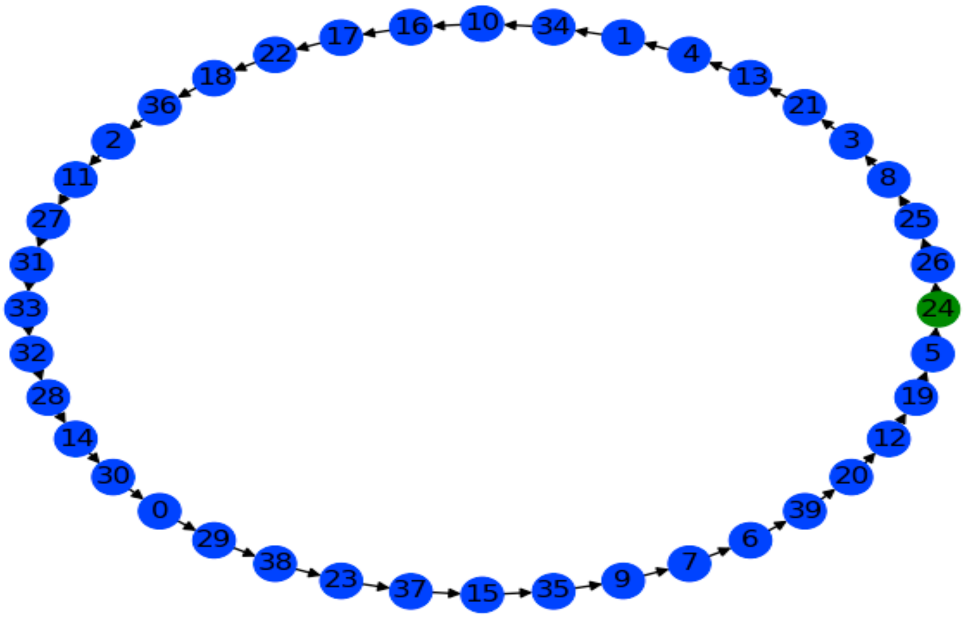


```
In [1]: runfile('/Users/lindiechenou/Desktop/CSE 545/PROJECT 3/Project3/
project3.py', wdir='/Users/lindiechenou/Desktop/CSE 545/PROJECT 3/Project3')
6.720697894000004
(21, [21, 8, 9, 14, 10, 23, 29, 0, 15, 6, 3, 24, 19, 5, 28, 12, 11, 7, 1, 16, 18,
22, 20, 2, 13, 27, 25, 17, 4, 26], 988.137304685266)
```





```
In [2]: runfile('/Users/lindiechenou/Desktop/CSE 545/PROJECT 3/Project3/project3.py',
wdir='/Users/lindiechenou/Desktop/CSE 545/PROJECT 3/Project3')
16.077028127000005
(24, [24, 26, 25, 8, 3, 21, 13, 4, 1, 34, 10, 16, 17, 22, 18, 36, 2, 11, 27, 31, 33, 32,
28, 14, 30, 0, 29, 38, 23, 37, 15, 35, 9, 7, 6, 39, 20, 12, 19, 5], 1471.3558359176104)
```



4. **Discussion** (Talk about the results you got and answer any specific questions mentioned in the assignment.)
   Since with the greedy algorithm, there is no specific node to start with for the algorithm to function. So, for the Random30.tsp and Random40.tsp the starting node respectively where 21 and 24 because they are the most central node from all other in their files. With the closest nodes being 8 for the 30 cities file and 26 for the 40 cities file. This step will be repeated until all the nodes have been repeated. So, while doing this it will take a modest amount of time with the algorithm program to complete. For the 30 cities file on average, it took about 6 seconds to complete the task, with the 40 cities file having an average of 13 seconds to complete. There are possible faster ways and more effective ways to provide a solution for this project. With the way, the past project has been coded and the timing between them, the greedy algorithm is by far the faster

especially with the number of nodes that need to be computed. For the first project using the brute force on a 30 nodes file will take forever to complete. For the second project, the timing will be better but will still not compare to the greedy algorithm.

5. **References** (If you used any sources in addition to lectures please include them here.)

https://www2.isye.gatech.edu/~mgoetsch/cali/VEHICLE/TSP/TSP009__.HTM