**CECS 545 Project 3 report**

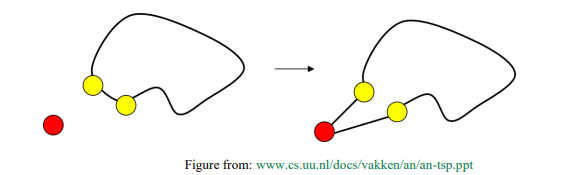
**Ryan Schron**

**September 18, 2019**

**Introduction**

At this point we’ve explored finding the least cost Hamiltonian Cycle using a brute force approach as well as via search. We saw the exponential explosion of complexity in trying a brute force approach, and were unable to solve for graphs larger than 12 nodes. Using search, we could find the cycle in a reasonable amount of time, but the problem was also constrained. In this project, we will use a greedy approach to find the least cost Hamiltonian Cycle in graphs up to 40 nodes in size.

This project will implement a greedy solution that is based on a best candidate insertion heuristic. Nodes will be added to the cycle based on the lowest cost, determined by distance to a node that already exist in the tour. Furthermore, the exact location will be determined based on the neighbors of the closest matching node, so that the correct order will be maintained. This is roughly illustrated in the following graphic:



We expect this approach to provide a reasonable solution for much larger instances of the Traveling Salesman Problem than our previously explored solutions.

**Running the Program**

The program can be run by navigating to the appropriate directory and using the following command in a terminal:

python greddytsp.py [filename]

The program requires that a single argument be given, which specifies the .tsp file holding the graph. This file will be read into the program, and it return the best-found tour, the tours cost, program execution time, and the order in which the nodes were inserted into the solution. An error message will print to the console if the program was launched incorrectly. Finally, numpy must be installed for the program to properly function.

**Code Description**

The program starts by reading in the provided .tsp file and converting the coordinates within into an adjacency matrix representing the graph. Following this, we initialize the state of the system by creating an empty list to hold the best-found tour, and set the best-found distance to infinity for further comparisons as the tour is determined.

Once the state has been set, the program picks a starting node and begins to solve the graph. We create a list of unvisited nodes and initialize the current path to begin and end with the starting node. The unvisited node list is passed to the helper function find\_next() where it compares each unvisited node to every current node in the tour. Once the closest node is located, the function returns the closet node and which node in the current path is closest to it.

After identifying the node to add, the system compares the distance of that node to the nodes before and after its determined closest match, so that it may be inserted into the correct place in the tour. That is, if node n is closest to node m in the current tour, n is compared to tour[index(m+1)] and tour[index(m-1)]. The node is inserted into the tour between node m and which of those two candidates is closer to maintain the correct path. In addition to determining the correct insertion location, a list of identify nodes is constructed in the order they are found.

The program repeats this process for every starting node to exhaust all possible permutations of the best tour. As a tour is constructed, the program calculates the path length, and stores the tour if that value is better than the current best-found tour. Finally, the program outputs to the console the best-found tour, the path length of that tour, and the order in which the nodes in the tour were discovered.

**Code Performance**

The greedy implementation tested in this project yielded the following results:

Random30.tsp

Best Tour: [8, 10, 30, 15, 1, 24, 7, 23, 28, 19, 3, 21, 17, 16, 12, 14, 9, 22, 27, 5, 26, 2, 4, 13, 6, 20, 25, 29, 18, 11, 8]

Best Tour Distance: 572.63299

Run Time: 0.0410056 seconds

Insertion Order: [8, 10, 16, 11, 12, 18, 4, 29, 13, 25, 6, 20, 2, 14, 26, 5, 9, 27, 22, 17,

7, 30, 24, 15, 1, 23, 19, 28, 3, 21]

Random40.tsp

Best Tour: [2, 14, 9, 22, 27, 5, 26, 4, 29, 25, 20, 6, 13, 38, 33, 31, 35, 12, 37, 36, 34, 28, 32, 19, 3, 21, 23, 16, 17, 7, 24, 1, 15, 30, 40, 10, 39, 8, 11, 18, 2]

Best Tour Distance: 701.37091

Run Time: 0.109547 seconds

Insertion Order: [2, 14, 26, 35, 4, 33, 29, 38, 31, 18, 12, 11, 16, 10, 40, 8, 13, 25, 6, 20, 37, 23, 34, 19, 17, 7, 30, 24, 15, 1, 36, 39, 5, 28, 32, 9, 27, 22, 3, 21]

**Conclusions**

This TSP solution could solve a much larger graph in a significantly faster time than previous solution implemented in Project 1. In that project, we were not able to reasonably attain a solution for graphs above 12 nodes, whereas in this experiment we were able to solve the 40-node graph in around a second. Additionally, the implementation scaled incredibly well with regards to the graph size. Moving from the 30-node graph to the 40-node graph resulted in only doubling the run time, which remains a strong result for a project with exponential complexity. In short, this greedy approach worked well to solve problems with larger, fully connected undirected graphs.