1 Introduction

Starting our research, there were a plethora of information on the various methods to obtain or at least approximate an optimal traveling salesman tour. We found a variety of algorithms and heuristic approaches. Some were straightforward such the nearest neighbor heuristic, while others had a more methodical approach to approximating an optimal tour. After some deliberation we decided upon the Christofides approximation algorithm as there seemed to be ample documentation regarding the algorithm as well a quality that would satisfy the required bounds.

2 Christofide

2.1 Background

The Christofides algorithm improves on the 2-approximation algorithm, providing a worst case ratio of 3/2 at the cost of increasing the running time to $O(n^3)$ [1].

It can be summarized with the following steps:

- Using the entire graph, build a minimum-spanning tree Make a set of vertices
- Make a set of vertices with odd degree
- From the set of vertices with odd degree, make a minimum weight matching
- Add the minimum weight matching to the minimum-spanning tree
- Find a Euler tour from the graph made by combining the minimum spanning tree with minimum weight matching
- Remove repeated vertices, thus turning it into a Hamiltonian circuit [1]

Step one, getting the minimum spanning tree, makes it such that we are working with a new graph that has eliminated some of the higher cost edges without getting rid of any of the vertices we must visit (recall all must be visited). Next, we get the set of vertices with an odd degree which, by the handshaking lemma, must be an even number of vertices [2]. Finding a minimum weight matching on these and adding it to the minimum spanning tree ensures that the shortest of edges will be available when finding the tour. Finding a minimum weight matching also ensures each vertex has enough edges to avoid having to reuse edges in order to get off a vertex. In other words, it ensures a Euler tour can be found on this subgraph that includes all the lowest-weight edges. After finding a Euler tour, we can simply remove repeated vertices because each vertex can only be visited once, and in the actual map all vertices can be reached. Due to the triangle inequality, removing repeated vertices doesnt increase weight. [2] Tests have shown that the algorithm tends to place itself 10% above the Held-Karp lower bound[1].

The key difference between it and the 2-approximation algorithm is that, rather than simply duplicating edges to ensure an Euler cycle, it creates a minimum weight matching on the vertices with odd degree and combines it with the minimum spanning tree [1].

2.2 pseudocode

Below in the pseudocode for the algorithm. Note that it combines the step of finding a minimum weight matching with the one that combines it with the minimum spanning tree.

Algorithm 1 Christofide

```
1: procedure CHRISTOFIDE({$G = (V, E),\ s$})
2:     for $v \in V$ do
3:         $v_{key} \leftarrow \infty$
4:         $v_{parent} \leftarrow \varnothing$
5:     end for
6:     $s_{key=0}$
7:     while $|k|$ do
8:     end while
9: end procedure
```

2.3 Works Cited

3 Nearest Neighbor Heuristic

3.1 Description

The nearest neighbor heuristic was one first method that we encountered. It is one of the most straightforward approaches. The heuristic is not so unlike how we would browse a museum. Similar to how we would start by picking a point of interest, the nearest neighbor heuristic begins at a single node. Then like touring a museum, we pick the next closest exhibit or display, and continue going through the museum avoiding things weve seen. The nearest neighbor heuristic works the same way, it identifies the unvisited nearest neighbor to the current node, and traverses the graph until all unvisited nodes have been seen once.

Algorithm 2 Nearest Neighbor

```
procedure NEAREST NEIGHBOR($\{G = (V, E),\ s\}$)
    $x \leftarrow$ Random Value
    $L \leftarrow \{\}$
    $s_{key=0}$

    while $|L| \neq |V|$ do
        for $\{\ u \in N(v) \mid v \in V\ \}$ do
            $x \leftarrow \min N(v)$
            if vertex unot visited then
            else
                $x \leftarrow$ Next shortest adjacency
            end if
        end for
    end while

    $L \leftarrow L \cup x$
end procedure
```

3.2 Pseudocode

3.3 Detail

In additional to being fairly easily implemented, the nearest neighbor heuristic is also a comparatively quick method. A major attraction of the nearest neighbor is that it runs fairly quickly, it has a time complexity of $O(n^2)$ [1].

However there some rather large drawbacks to the approximation. A major flaw is that in its greed, the tour can miss obvious shorter routes that could be identified with more comprehensive algorithms. Additionally for the same reason, the method can miss nodes till the end and need to include them at high cost. In order to improve the path it would then require additional improvement heuristics and approximation algorithms to decrease the cost of the tour and increase quality. Though it was both simple and clear, for these reasons the approach was not a suitable method for finding an optimal tour.