Travelling Salesperson Problem

William Davies, Trent Louie, Ross Shoger

March 17, 2017

1 Introduction

There are many ways to approach the Travelling Salesman Problem (TSP), otherwise referred to as the Travelling Salesperson Problem in this project.

In this report are summaries for three approaches to solving the TSP: The Christofides approach, the nearest neighbor solution and an approach by greedy heuristics. These are all approaches that do not produce optimal solutions, but solutions that are some multiple of the optimal solution.

2 Christofides Heuristic

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].

The following steps summarize the Christofides algorithm:

- 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 [1]. 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 [2].

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 [2].

2.1 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.

procedure CHRISTOFIDES($\{G = (V, E),\ s\}$)
    for $v \in V$ **do**
        $v_{key} \leftarrow \infty$
        $v_{parent} \leftarrow \varnothing$
    end for
    $s_{key=0}$

    $K \leftarrow$ *min-priority queue based on* v_{key}
    $K \leftarrow v \in V$
    while $|K|$ **do**
        $x \leftarrow dequeue(K)$
        for $\{u \mid u \in N(x)$ **do**
            if $u \in K \wedge distance(x, u) < u_{key}$ **then**
                $u_{parent} = x$
                $u_{key} = distance(u, x)$
            end if
        end for
    end while

    $G_{MST} \leftarrow MST(G)$
    $V_{odd} \leftarrow$ *set of odd vertices from* G_{MST}
    for $\{v \mid v \in G_{MST}\}$ **do**
        $d = 0$

for $\{u \mid v \in G_{MST}\}$ **do**
$d \leftarrow d + 1$
if $d \bmod 2 \neq 0$ **then**
$V_{odd} \leftarrow V_{odd} \cup u$
end if
end for
end for

while $|V_{odd}|$ **do**
$x \leftarrow$ *a vertex removed from* V_{odd}
$d \leftarrow \infty$
for $\{v \mid v \in V_{odd}\}$ **do**
if $distance(v, x) < d$ **then**
$d \leftarrow distance(v, x)$
$match \leftarrow v$
end if
$A(G_{MST}) \leftarrow A(G_{MST}) \cup match$
$G_{odd} \leftarrow G_{odd} - match$
end for
end while
Initialize stack for vertices
Initialize an array or queue called tour
Current = s
Put Current in tour
while $|N(current)|$ **do**
if $|N(current)$ **then**
temp = current
current = first vertex in $A(V_{MST})$
push temp onto stack
remove temp from $A(V_{MST})$
remove current from $N(temp)$
else
add current to tour
current = pop stack
end if
end while
for $\{v|v \in \text{tour}\}$ **do**
vertexInTour = 0
for $\{u|u \in \text{tour}\}$ **do**
if $v \implies u$ **then**
vertexInTour++
end if
if vertexInTour > 1 **then**
remove u from tour
end if
end for

```
    end for
end procedure
```

3 Nearest Neighbor Heuristic

The nearest neighbor heuristic 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.

3.1 Pseudocode

Algorithm 1 Nearest Neighbor

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

    while |L| ≠ |V| do
        for { u ∈ N(v) | v ∈ V } do
            x ← min N(v)
            if vertex unot visited then
            else
                x ← Next shortest adjacency
            end if
        end for
    end while

    L ← L ∪ x
end procedure
```

3.2 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 $O(n^2)$ [2].

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.

4 Greedy Heuristic

Another straightforward approach is the Greedy approach heuristic. The greedy heuristic works by evaluating all the edges to slowly build the tour. First the edges are all sorted and the shortest edges is identified. The approach begins by adding this smallest edge to a possible tour. It then finds the next shortest edges and evaluates it. Does adding this new edge to the tour create a cycle with less than V edges? Does the newly added edge increase the degree of the node to more than 2? Has this edge been used before? The approach checks each edge with these criteria and as long as it does not violate them will add it the tour. It will repeat this process until all of the nodes have a degree of two and there are $|V|$ edges. This means the tour has created a Hamiltonian cycle.

4.1 Pseudocode

Algorithm 2 Greedy

procedure GREEDY($\{G = (V, E), S\}$)
    $E_{sorted} \leftarrow Sort(E)$
    $x \leftarrow \min E$
    $T \leftarrow T \cup x$

    while $|V| \neq |T|$ **do**
        $x \leftarrow \min E_{sorted}$
        if $x \cup T$ *has no cycle with edges* $> |V|$ **then**
            if $deg(x) \leq 2$ **then**
                if $\nexists x \in T$ **then**
                    $T \leftarrow T \cup x$
                end if
            end if
        end if
    end while

end procedure

4.2 Detail

A major drawback to this approach is that it does no forecasting and picks the best selection at the moment. This greedy shortsightedness is exactly like nearest neighbor, as they both do not look ahead. The difference between the two greedy approaches is where the focus lies. The nearest neighbor evaluates adjacent nodes, while the greedy approaches focuses on all edges to build a tour piece by piece. As one can see it has a runtime of $O(n^2 log2(n))$ [2] where a large portion of the running time is spent ordering the edges. Like any greedy algorithm, it often produces sub-optimal tours and requires additional improvement heuristics and algorithms to increase quality. For all its simplicity, it also was not a good choice due to its inability to approximate an optimal tou

References

[1] Michael T Goodrich and Roberto Tamassia. The christofides approximation algorithm. *Algorithm Design and Applications, Wiley*, pages 513–514, 2015.

[2] Christian Nilsson. Heuristics for the traveling salesman problem. Technical report, Tech. Report, Linköping University, Sweden, 2003.