The Traveling Salesman Problem

Group 7
Nathan Thumen, Larissa Hahn, Nathalie Blume
August 14, 2013

1 Introduction

Given distances between points, the Traveling Salesman Problem (TSP) consists of finding the shortest path through all the points returning to the starting point, without visiting any single point except the starting point more than once.

There exist algorithms that will discover the shortest path. These algorithms, including a brute force algorithm and several bound and branch algorithms (e.g. the Held-Karp dynamic programming algorithm), run in more than polynomial time, making the TSP an NP-hard problem. Alternative algorithms look for an approximation to the shortest path in polynomial time. The Christofides algorithm for instance is guaranteed to find a path that is no more than 1.5 times longer than the shortest path and has a worst-case time complexity of $O(n^3)$. It is one of the most effective TSP heuristic algorithms known today. The Nearest Neighbor algorithm (also called the Greedy algorithm) is quicker than Christofides though less accurate. Several other algorithms exist besides these, including an algorithm inspired by the behavior of ant colonies.

For the current project, we were tasked with developing an algorithm that balanced speed and accuracy. Our group chose to implement the Nearest Neighbor algorithm. We wanted to provide an acceptable solution to input of very large size, given our intention to complete the project before we knew the size of the competition test sets. We understood that our algorithm would not be highly competitive except where speed mattered most.

2 Nearest Neighbor Approximation Algorithm

2.1 Usage and Assumptions

- The input consists of a file listing cities and their x-, y-coordinates in a Euclidian system.
- The input defines a complete graph G = (V,E,w) where the set V is the set of cities and the set E is the complete set of distances between pairs of cities.
- Edge weights satisfy the triangle inequality.
- All edge weights (i.e. distances between cities) are positive.

2.2 Overview of the Algorithm

1. Start on an arbitrary vertex.
2. Discover the closest unvisited adjacent vertex V.
3. Mark V as visited.
4. Test whether V has any unvisited neighbors; if no, terminate.

5. Go to step 2.

2.3 Pseudocode

The input is passed to a matrix M of n cities and 3 data points, the city id, x- and y-coordinate. The distance between two points start and dest is calculated with round(sqrt(pow((startX - destX), 2.0) + pow((startY - destY), 2.0))), where X and Y represent Euclidian coordinates.

The pseudo-code for finding the shortest path is:

```
function TSP(int M[n][3])
    for i in range n do
        insert M[i][1] in citiesNotVisited
    end for
    citiesVisited ← empty set
    startCity = currentCity ← 0
    tourLength ←  0
    i ← n
    while i > 0
    lowestDistance ← +inf
    for j in range n do
        if j is not currentCity and j is not in citiesVisited
        legDistance ← computeDistance(j, currentCity])
        if legDistance < lowestDistance
            lowestDistance ← legDistance
            closestCity ← j
        end if
        end if
    end for
    tourLength ← tourLength + lowestDistance
    insert closestCity into citiesVisited
    currentCity ← closestCity
    decrement i
    end for
    tourLength ← tourLength + computeDistance(currentCity, startCity)
end function
```

The algorithm maintains an array citiesNotVisited for the purpose of iterating through the full set of cities, and maintains an array citiesVisited with the cities that are already in the circuit and that shall not be returned to. It iterates through each city i and computes its distance to every other city j that has not already been visited. It saves the smallest of these distances and adds it to the accumulator tourLength. It

also adds the city at that smallest distance to the citiesVisited set, and it selects this city as its next starting point. Distance is computed in a helper function.

The output of the algorithm is the sequence of visited vertices. This sequence is the shortest circuit discovered by the algorithm.

3 Results for the Three Examples

	Distance	Time
Example 1	150393	0.016 sec
Example 2	3210	0.031 sec
Example 3	1964948	23.086 sec

4 Results for the Competition Instance

	Distance	Time
Instance 1	5926	0.016 sec
Instance 2	9503	0.014 sec
Instance 3	15829	0.020 sec
Instance 4	20215	0.047 sec
Instance 5	28685	0.146 sec
Instance 6	40933	0.473 sec
Instance 7	63780	2.963 sec

References

[1] https://en.wikipedia.org/wiki/Nearest_neighbour_algorithm

[2] https://en.wikipedia.org/wiki/Travelling_salesman_problem

[3] Nilsson Christian, Heuristics for the Traveling Salesman Problem. Linkoping University. https://web.tuke.sk/fei-cit/butka/hop/htsp.pdf