The Long Road: Comparison of Approximation Heuristics for the Traveling Salesperson Problem

Daniel Chen, Artem Yushko, Arthur Viegas Eguia, Miles Hoene-Langdon

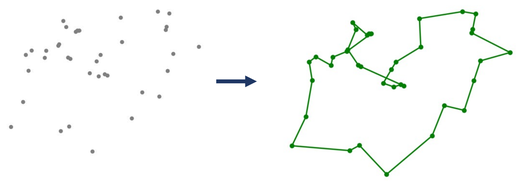
Advisor: Layla Oesper

Problem Formulation

The traveling salesperson problem (TSP) is an important problem in optimization. Simply put, say a salesperson wants to visit n cities and then return to the one they started in. A solution to TSP would be a tour through these cities of the shortest possible distance. More generally:

Input: A complete, undirected, weighted graph

Output: An ordered list of nodes in the graph, the edges between which are of minimum possible weight



TSP has many real-world applications ranging from simple ones like the problem's namesake to more complex ones. We looked at a few applications of the problem: The basic case with cities from various countries[1], a more specific problem using electrical grids, and a non-Euclidean variation involving protein-protein interaction[2].

Objectives

TSP is NP-Hard meaning a polynomial time algorithm that solves it exactly is not currently known. Exact algorithms are exponential at best, which will run very slowly when the input is large. More practical are heuristics: algorithms that yield approximate solutions to TSP. Our goal is to implement heuristics of varying degrees of complexity and compare their accuracy and runtime to see which is best.

Heuristics:

- Nearest Neighbor: Starting at a random node, visit the closest node to it and add it to the tour. repeat until all nodes have been visited and then return to the first node and add it to the tour. This algorithm's worst-case runtime is $O(n^2)$.
- Smallest Insertion: Starting with two random nodes in the tour, add nodes to the point in the tour that will increase the total distance by the lowest value. This algorithm's worst-case runtime is $O(n^2)$.
- The Christofides Algorithm[3]: Our most complex algorithm, whose worst-case runtime is $O(n^3)$. A detailed explanation will follow.

All heuristics were implemented in python, mostly from scratch with some use of NetworkX helper functions. Testing was done on a Carleton lab machine for consistency of results.

References

1. *National traveling salesman problems.* UWaterloo.

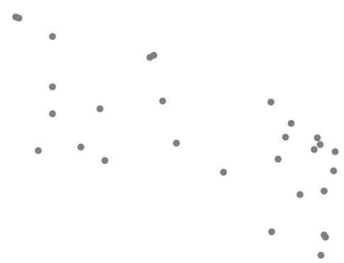
2. *A traveling salesman approach for predicting protein functions.* Johnson, O., Liu, J.

3. *Worst-Case Analysis of a New Heuristic for the Travelling Salesman Problem.* Christofides, N.

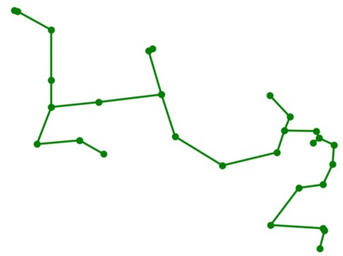
4. Paths, Trees, and Flowers, Edmonds J.

The Cristofides Algorithm

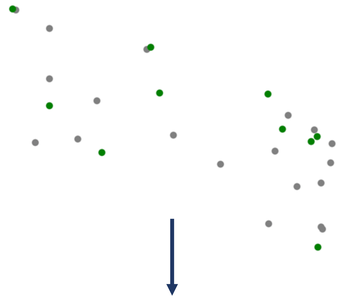
The Christofides algorithm goes through multiple computational steps to reach its solution. We'll walk through the process step by step on our Western Sahara dataset that includes 29 cities:



Create a minimum spanning tree T of our input graph:

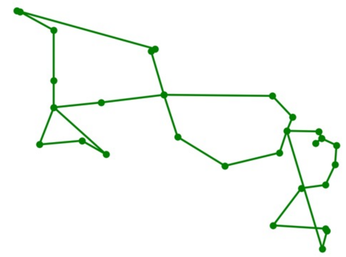


Create a new graph O with all the odd-degree nodes in T.

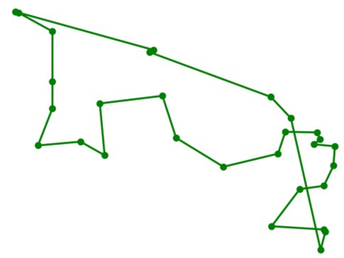


Find a minimum weight perfect matching M of O using the Blossom algorithm with optimization for weighted edges.

Combine the edges of M and T to form a multigraph H, and then form an Eulerian circuit in H:



Finally, skip repeated vertices in our Eulerian circuit to form a Hamiltonian circuit. The result is our approximate solution to TSP:

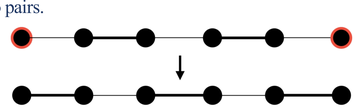


The Blossom Algorithm

The most complicated step in the Christofides algorithm is finding a minimum weight perfect matching of the odd degree nodes in the MST generated in the first step. The Blossom Algorithm solves this problem by finding augmenting paths and blossoms.[4]

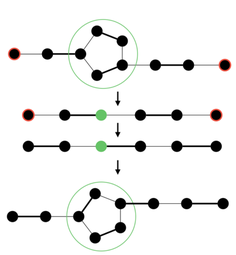
Augmenting Paths:

The algorithm searches for paths that begin and end with unpaired nodes. This allows it to switch the matchings to ensure that these terminal nodes are added to pairs.



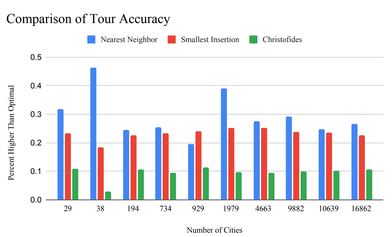
Blossom Contraction:

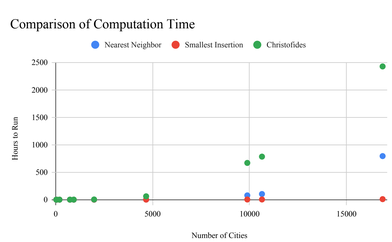
Odd-length cycles can hamper the algorithm's ability to find augmenting paths, so we look for them and contract them when found. After we finish finding the augmenting path, we can expand the blossom to its original size.



Results

We ran each of our three heuristics on country data sets ranging from just 29 cities up to over 16,000. Each were analyzed based on total time for which the algorithm ran and how close the tour produced was to optimal.





Conclusions

The simple heuristics run much quicker than Christofides but yield significantly worse tours. On smaller data, none of the algorithms take an much time to run. On larger data, there is significantly more stratification in runtime with Christofides taking much longer than the other two algorithms and smallest insertion not taking much time at all. Regardless of the size of the dataset, Christofides yields a significantly better tour than either of the other algorithms which have similar accuracy to each other.

Choosing which algorithm to use would depend heavily on the application.

- Anything with small enough data should use Christofides for the increased accuracy and minimal runtime blowup
- Larger data would require more analysis. If it is vital to get an accurate tour, then Christofides is preferred, but if the tour is needed quickly, Smallest insertion would likely be satisfactory.

Acknowledgements

I would like to thank our Comps advisor Professor Layla Oesper for her guidance throughout the project and Mike Tie, Technical Director in Computer Science, for his technical support.