Luv Dabhi, Douglas Hernandez
Shiraj Pokharel
CSC 4520
01 December 2023

# The Traveling Salesman (Georgia Edition)

The Traveling Salesman Problem (TSP) stands as one of the quintessential challenges in the field of optimization and algorithm design. Rooted in graph theory, TSP asks what is the shortest possible path that visits every city given in a set exactly once and returns to the origin city. This seemingly straightforward problem becomes quite complex when you start considering how many different paths are possible from every combination of cities you can visit. This is what is known as an NP problem, where it is easy to verify a solution in polynomial time, but not easy to solve the problem in that time. For our project, we wanted to tackle this NP-hard problem with 10 different cities in Georgia while considering a multi-faceted approach to see what algorithm would give us the most efficient and accurate solution. We will be examining the time complexity of each using V as vertexes (cities) and E as edges (distances).

The first approach we considered was a brute force method. The brute force method exhaustively explores all possible permutations of the cities to find the shortest tour. As shown in the file TSP.java, it generates all possible permutations by choosing an arbitrary city and swaps the order around, calculates the total distance for each permutation, and identifies the tour with the minimum distance. Although we can find an answer like this, this approach has a time complexity of O(V!) due to the permutations of all 10 cities, making it inefficient for large instances of n cities.

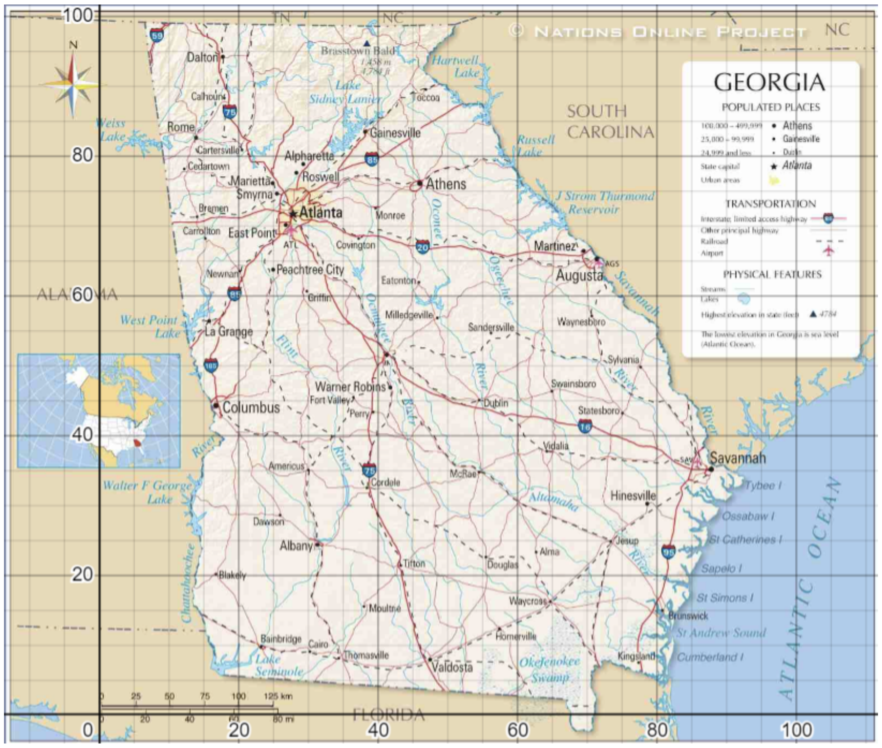
After noticing this, we then turned to approximation algorithms such as nearest neighbor algorithm. The nearest neighbor approximation algorithm is a heuristic that offers a practical approach for TSP,  efficiently constructing a tour by iteratively selecting the nearest unvisited city step-by-step to the current city, gradually forming a tour. The TSPnear.java file, in its execution, initializes the tour from any starting city, systematically adding the nearest unvisted city until the entire route is covered. While this algorithm does not guarantee an optimal solution, it provides a quicker and reasonably accurate solution with a time complexity of $O(V^2)$. Just like the previous method, the tour and total distance are printed at the end of the execution, helping us compare the accuracy of our solution when using these different methods.

Since the brute force method isn't efficient for larger amounts of data, and the nearest neighbor algorithm was suboptimal in nature, we took a different approach. The algorithm we tried next was Prim's algorithm. Prim's algorithm is typically used for finding the minimum spanning tree in a graph with weights. As shown in the java file TSPprims.java, we adapt Prim's algorithm to construct a graph structure out of the cities and then find the MST of the graph. This is done by choosing an arbitrary city and then iteratively choosing the edge with the shortest distance to an unvisited city from all available edges and repeating this until all cities have been visited. By this method, we can achieve a run time complexity of O(E*log(V)), which is much quicker than our two previous methods. The result is a tour that should be an accurate and quick solution for the shortest possible path, which is printed along with the total distance of the tour.

In conclusion, we have explored three different approaches to solving the Traveling Salesman Problem. The brute force method is an exhaustive search to find the optimal solution. While for our data set, we could use it to find a solution, it is computationally expensive with larger input sizes. The nearest neighbor approximation algorithm is a faster heuristic that provides a reasonably accurate and quicker solution; however, the time complexity still is not ideal. Prim's algorithm provides the quickest solution between our 3 methods, but it may not always yield the optimal tour. The choice of algorithm we use depends on the specific requirements of the problem and what we're willing to sacrifice, considering factors such as time complexity and the desired level of optimality. If we were to design a different algorithm for this project, we have the idea of the algorithm somehow choosing the starting city at a somewhat optimal location, such as somehow figuring out what city is the closest to the center of the map. This would make it so that the city has the shortest distance to access all other cities when trying to construct a tour, thus finding the shortest possible path much quicker.

Works Cited + Work Distribution (bottom) ↓

1. https://www.youtube.com/watch?v=1pmBjIZ20pE&ab_channel=AlphaOpt
2. https://www.techtarget.com/whatis/definition/traveling-salesman-problem
3. https://www.engati.com/glossary/brute-force-search#:~:text=brute%2Dforce%20search%3F-,What%20is%20Brute%2Dforce%20search%3F,search%20or%20generate%20and%20test.
4. https://www.tutorialspoint.com/design_and_analysis_of_algorithms/design_and_analysis_of_algorithms_approximation_algorithms.htm
5. https://www.youtube.com/watch?v=GC-nBgi9r0U&t=221s&ab_channel=JohnSong
6. https://www.brainkart.com/article/Exhaustive-Search_8013/
7. https://levelup.gitconnected.com/prims-algorithm-visually-explained-fb69a786f352



8. https://www.nationsonline.org/oneworld/map/USA/georgia_map.htm

Work Distrbution

Research: Luv, Douglas

Report: intro - Luv, Conclusion - Douglas, Body - Luv & Douglas

Code: TSPnear.java - Luv & Douglas, TSP.java - Douglas, TSPprims - Luv

Editing: Luv, Douglas