**Module 6: Portfolio Milestone**

**Traveling Salesman Toolbox**

Joshua Humphreys

Colorado State University Global

Course Code:CSC506

Dr. Brian Holbert

June 9, 2024

**Module 8 Portfolio Project**

For my portfolio project I selected the traveling salesman problem (TSP). The Traveling salesman problem states: given a set of cities and the distances between them, what is the shortest route that visits each city, starts and ends at the same city, and doesn’t visit any city more than once? The obvious applications of solutions to this problem are efficient route planning. Delivery vehicles, hospital systems moving patients, garbage trucks, traveling salesmen, etc. can all benefit from finding efficient ways to complete a cycle (a path where the start and end points are the same). However, less obvious applications of the traveling salesman problem include network design, project management, robotics, and manufacturing to name a few. Effectively, any time something needs to start in one state, transition between states, end in the state it began, there is a cost associated with transitions, and transition costs vary depending on the states they connect, then the traveling salesman problem applies.

Finding the optimal solution to the TSP is extremely difficult and likely cannot be done in polynomial time. Furthermore, verifying that a given route is, in fact, optimal is also very difficult and cannot be done in polynomial time. The applicability of the TSP as well as the difficulty in finding optimal solutions are what make the TSP so interesting. Because an optimal solution cannot be found in polynomial time, it’s necessary to sacrifice some optimality so that a solution can be reached. In other words, a good enough solution is better than no solution.

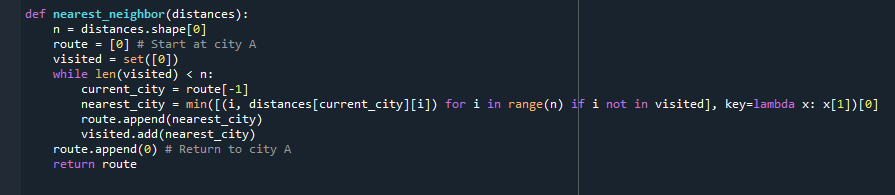
The goal of this project was to compare four different heuristic algorithms for solving the TSP. Heuristic algorithms are those algorithms that can provide a good enough solution to a problem in a reasonable time but cannot guarantee the solution is optimal. The algorithms I selected for this project were the Nearest Neighbor (NN), Genetic, Minimum Spanning Tree (MST) and 2-Opt. In the following pages I will discuss my research methodology, analysis, and outcomes.

**Methodology**

The algorithms selected for analysis each approach the TSP in different ways. The Nearest Neighbor takes a greedy approach. That is, the NN makes locally optimal choices in hopes that a globally optimal solution is reached. The NN adds nodes to a path by selecting the node with the shortest edge connected to the most recent node added to the path. The genetic algorithm attempts to replicate how genetic traits are passed from generation to generation. Genetic algorithms start with an initial population that reproduces to create a new population from which the “fittest” reproduce and so on. The MST produces a minimum spanning tree by adding the shortest edge connecting a node not yet in the tree (but not necessarily connected to the most recently added node) until all nodes are in the tree. A path is then generated from the tree by traversing the tree in pre-order. The 2-Opt algorithm is an optimization algorithm that checks if swapping nodes produces a shorter path.

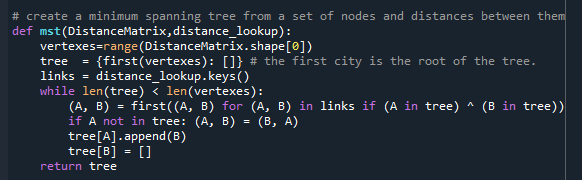
**Figure 1.**

*Nearest Neighbor Algorithm*

**

**Figure 2.**

*Minimum Spanning Tree*

**

**Figure 3.**

*2-Opt Algorithm*

*A computer screen shot of code

Description automatically generated*

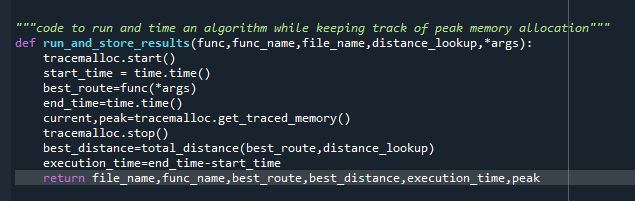
I started my research by finding implementations of each algorithm written by others. During my research, I found an implementation of a genetic algorithm (Baranowski, 2023) that also included functions necessary to load a set of included test graphs. This implementation formed the basis of my implementations. I adapted implementations for the other algorithms to accept the input used in the genetic algorithm. The genetic implementation also included mechanisms to time the algorithms which I used for each algorithm as well.

Next, I adapted each algorithm to only produce a route as output. This allowed me to build a 2-Opt algorithm that would accept output from the other algorithms. In this way I was able to test two hybrid approaches that combined the 2-Opt optimization with the Nearest Neighbor and the MST. The result of these adaptations was a set of algorithms that accepted similar input and produced similar output. I then wrote driver code that prepared each test graph and iterated through each algorithm.

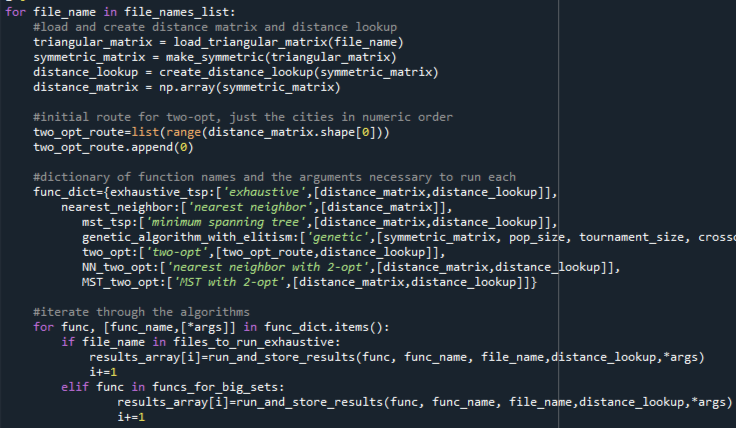
After my initial implementation, I noticed a few inefficiencies in the driver code. Namely, I had written a function for each algorithm that ran and timed each algorithm. I was able to move the timing functions to a single function, thus eliminating significant clutter. I also added the capability to record peak memory allocation and eliminated some redundancy in performing some calculations. To make the functions work in as standard of way as possible, I used a dictionary with algorithm names and arguments. The figures below demonstrate the highlights of how arguments were passed to each algorithm in a standardized way.

**Figure 4.**

*Function to run each algorithm*

****Figure 5.**

*Code to run each algorithm on each test graph.*

******

**Analysis**

The first run highlighted an obvious problem with the 2-opt algorithm. As can be seen in Figure 3 below, the 2-opt algorithm consistently poor results in very little time. Investigation into the issue revealed an error in my use of iterators to select pair for the 2-opt swap. After resolving the issue, the 2-opt swap functioned quite well. The correct 2-opt function is displayed in Figure 3 above.

After running the algorithm, I performed a few simple data cleaning steps that can be found in the attached Jupiter notebook. I then used Excel pivot charts to produce some revealing visualizations. Figures 6 and 7 below demonstrate that the time and space complexity associated with the 500 city set genetic algorithm are so much larger, that visualization of trends in the other algorithms and sets is difficult if they are included. Figures 8 and 9 then demonstrate time and space complexity of the remaining algorithms over the smaller test graphs.

**Figure 6.**

*Execution Time by Number of Cities with Genetic Algorithm and 500 city set.*

*A graph with numbers and text

Description automatically generated with medium confidence*

**Figure 7.**

*Peak Memory by Number of Cities with Genetic Algorithm and 500 city set.*

*A graph with different colored lines

Description automatically generated*

**Figure 8.**

*Execution Time by Number of Cities without genetic or 500 city set*

*A graph with different colored lines

Description automatically generated*

**Figure 9.**

*Peak Memory by Number of Cities without genetic or 500 city set*

*A graph with different colored lines

Description automatically generated*

It’s important to note that even though the 500-city set and genetic algorithms time and space complexity were significantly greater than the other algorithms and sets, the growth rate of both time and space were still within reason. That is, neither grew exponentially. In the previous charts we can see that the time and space complexity of each algorithm grew steadily with respect to increasing test set size. Figure 8 demonstrates what initially seems like strange behavior in the time complexity of the MST with 2-opt. However, it’s important to remember that the 2-opt algorithm depends heavily on the conditions of the initial algorithm. If there are many available swaps, the 2-opt algorithm will take significantly longer. This is likely the source of the behavior we see with the 101-city set. Figure 10 below demonstrates that the MST produced the second longest path while the MST with 2-opt produced the second shortest path for the 101-city set.

**Figure 10.**

*All algorithms ranked by Best Distance.*

*A colorful hexagon with many lines

Description automatically generated*

Figure 10 also demonstrates the inconstancy with which the genetic algorithm performed in this experiment. On the other hand, the MST with 2-opt and NN with 2-opt consistently produced routes with the shortest distances. Coupled with their relatively low time and space complexities, especially as compared to the genetic algorithm, we see that these hybrid approaches are quite effective.

**Outcomes**

This experiment demonstrated that the MST and Nearest Neighbor consistently produced results in the least amount of time while the 2-opt and Nearest Neighbor used the least amount of memory. The MST with 2-Opt and NN with 2-Opt consistent produced routes with the shortest distances. The time and space complexity of the genetic algorithm was so much larger than the others that making comparison between them is unreasonable. With that said, the genetic algorithm did produce some of the shortest routes in the test sets with smaller initial sets. The genetic algorithm does contain the most complexity of all the algorithms tested here. The genetic algorithm also has the greatest potential for tuning as it relies upon a set of parameters that govern tournament size, generations, and rules for how routes “reproduce”.

**Conclusion**

Due to their speed, relatively low space complexity, and consistent production of short results, the MST with 2-opt and NN with 2-opt are the algorithms best suited for deployment with dynamic data. That is, if the algorithm is to be used to find routes in real-time, then these algorithms likely produce a good-enough route and can do so in reasonable time. However, if speed is not of the essence and it is necessary to produce a route that is likely as close as possible to optimal, then it may be worth tuning a genetic algorithm.

**References**

Baranowski, M. (2023). *Genetic Algorithm for TSP Solver.* Github. <https://github.com/SonnyFixit/Travelling_salesman_problem/tree/main>

Bhagavan. (2023, March 26). *Solving the traveling salesman problem in python using the nearest neighbor algorithm.* Medium. <https://medium.com/@suryabhagavanchakkapalli/solving-the-traveling-salesman-problem-in-python-using-the-nearest-neighbor-algorithm-48fcf8db289a>

Bòna, M. (2011). *A walk through combinatorics* (3rd ed.). World Scientific Publishing Co. Pte. Ltd.

Cañada, T. (2023, January 23). *Introduction to the traveling salesman problem.* Medium. https://tonicanada.medium.com/introduction-to-the-travelling-salesman-problem-5ace44932cb5

Davis, A. (2022, May 18). *Traveling salesman problem with the 2-opt algorithm.* Medium. <https://slowandsteadybrain.medium.com/traveling-salesman-problem-ce78187cf1f3>

Gao. Y. (2020, February 14). *Heuristic algorithms for the traveling salesman problem.* Medium. <https://medium.com/opex-analytics/heuristic-algorithms-for-the-traveling-salesman-problem-6a53d8143584>

Geeksforgeeks. (2024, April 30). *Traveling salesman problem using genetic algorithm.* <https://www.geeksforgeeks.org/traveling-salesman-problem-using-genetic-algorithm/>

Geeksforgeeks. (2022, November 28). *Approximate solution for traveling salesman problem using MST.* <https://www.geeksforgeeks.org/approximate-solution-for-travelling-salesman-problem-using-mst/>

Helsgaun, K. (n.d.) *An effective implementation of the lin-kernighan traveling salesman heuristic.* Department of Computer Science, Rokslide University.

Kumar, A. (2024, March 08). *Genetic algorithms.* Geeksforgeeks. [https://www.geeksforgeeks.org/genetic-algorithms/#](https://www.geeksforgeeks.org/genetic-algorithms/)

Norvig, P. (2023).  *The traveling salesman problem.* Github. <https://github.com/norvig/pytudes/blob/main/ipynb/TSP.ipynb>

Quang, T.L. (2024, March 18). *Traveling salesman problem: Exact solutions vs. heuristic vs. approximation algorithms.* Baeldung. <https://www.baeldung.com/cs/tsp-exact-solutions-vs-heuristic-vs-approximation-algorithms#:~:text=Lin%2DKernighan%20is%20a%20local,a%20local%20minimum%20is%20reached>.

Sethi, R. (2020, August 26). *Hamiltonian paths and cycles.* Medium. <https://medium.com/stamatics-iit-kanpur/hamiltonian-paths-and-cycles-4f233bfbc53a#:~:text=Sir%20William%20Rowan%20Hamilton%20was%20an%20Irish%20mathematician%20and%20the,also%20called%20Hamiltonian%20paths>!).