OMGTSP

Jamie Fuente

July 2020

Problem Statement

Given a complete weighted graph G = (V,E,w), return a cycle containing all $|V|$ nodes with minimal weight.

Why the TSP?

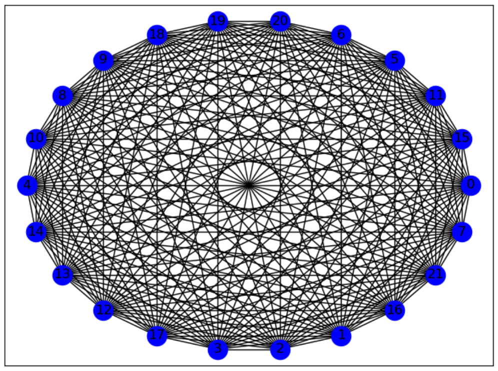
I've chosen to study the TSP because it lies at the intersection of the many diverse fields I encountered while an undergraduate student. Understanding this problem and building a good solver requires a subtle understanding of optimization, linear algebra, graph theory and computer science. Furthermore, at the time of writing, there is no efficient solution to the TSP. It is whats known as a NP-Hard problem (More on that later). Even so, there exists modern software that can solve the TSP quite quickly. The TSP lies on the boundary of what is and is not possible to do with a modern computer. For all these reasons and more, I believe the TSP is an excellent starting point for any student wanting to learn more about mathmatics and CS.

Hopes for this project

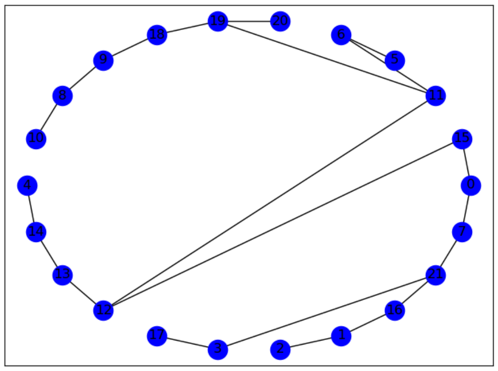
I started this project for several reasons. As I studied linear, non-linear and integer optimization at Rice, most of the algorithms were presented only in the abstract. Coding projects simply required the use of black-box solvers like Gurobi. I think truly understanding something requires doing it oneself, so I decided to start this project. The first stage of this project was mostly researching the algorithms and getting up to speed. The second phase, where I am now, requires defining project structure implementing the algorithms in an easy to debug language, namely python. The next phase will be identifying the most computationally intense parts of the algorithm and porting them to C++ or C. Several data science and optimization projects I've worked on require calling C++ libraries from python, so I am excited to learn more about this process. Lastly I plan to optimize and parallelize my code. These last two phases will blend nicely with the spring semester courses I pan to take in computer systems and parallel computing.

Part 1: The Heuristic

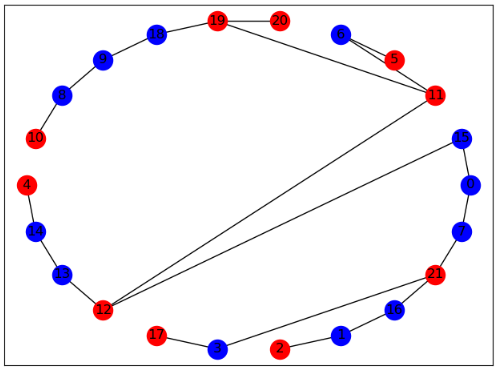
Christofides Algorithm is currently the best known polynomial time algorithm for approximating the solution to the TSP. The algorithm consists of 4 stages. Consider the complete graph ulyssess22.



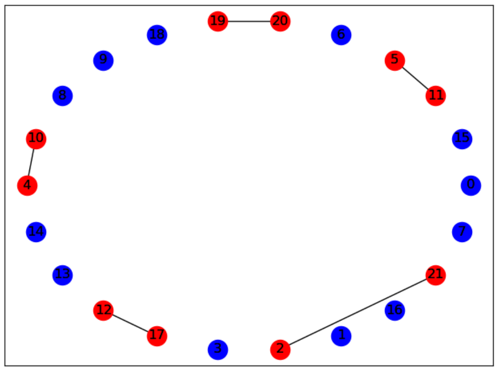
First, we construct a minimum spanning tree.



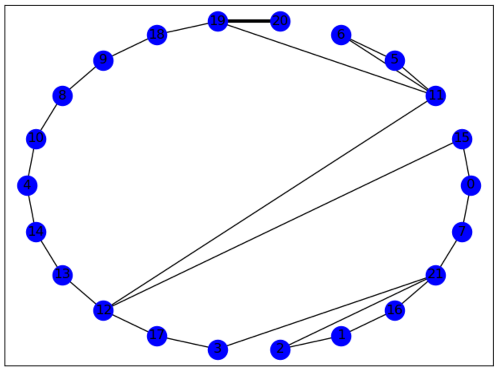
By the handshaking lemma, we know that this MST will have an even number of nodes that are adjacent to an odd number of edges. These node are identified bellow.



Second, we want our subgraph to be even. This means every node should be adjacent to an even number of edges. Luckily, there are an even number of odd nodes, so we can just match them up. Because our ultimate goal is a tour of minimum weight, this matching should have minimum weight. So we find a minimum weight perfect matching between the odd nodes. For more on this algorithm see MWPM.pdf



Third, begin to construct a tour. Because our subgraph is now even, we can find a Euclidean tour. This is a tour in which ever edge is visited exactly once.



Fourth and finally, we transform the Euclidean tour into a Hamiltonian tour (a tour in which every node is visited once). We do this by shortcutting. For example, in the above tour one can see that 21 is visited twice. Once in the path [7,21,16] and another in [3,21,2]. To short cut, we simply take all but one of these paths and cut out 21. If we keep [7,21,16], then [3,21,2] becomes [3,2]. Since 3 and 2 were once connected to 21 but are now connected to each other, they maintain the same number of edges. 21 however is adjacent to two less edges, so every node in the graph has either the same number or less edges than before shortcutting. We repeat this process until every edge is visited exactly once.