Savanna Moss & Celeste Rosendale

The Pizza Pilgrimage: a Simplified Traveling Salesperson

The traveling salesperson problem is known as a Millennium Problem, meaning that if anyone were to ever solve it, then you would receive $1,000,000. In short, you are a salesperson trying to visit every city without visiting the same city more than once, and spending the least amount of money on travel, whether that be in actual cost or overall mileage. The final project is more or less a 'baby' version of this problem, trying to take in a graph and use our proposed algorithm to find the smallest cost to travel around to each city at least once and then return home. The project was broken down into 3 parts: an algorithm to solve the pilgrimage (where the graph obeys the Triangle Inequality), an algorithm for any graph, and a path cost calculator to take the graph you find and determine your total cost to get back home. The algorithms will of course be heuristics, as to get a perfect answer for every graph even at this level is near impossible. Our goal is to come up with a solution, that, to the best of our abilities, gives the smallest cost possible to travel around our graph and return home.

Our project would be classified as an NP-Hard problem, which on the scale of P - NP, is not the easiest thing in the world to accomplish directly, especially within a short time period, but is possible. P problems are relatively simple to solve, and can be done so quickly with the aid of a computer. NP problems are deceptively easy, and even with a computer could have a very long running time that without an improved algorithm, you may not receive that perfect answer very quickly, but in fact the running time would be so great we wouldn't theoretically find our solution. The Pizza Pilgrimage best represents NP-Hard because it can be at least as difficult as NP-Complete (being able to reduce the time necessary for any problem knowing there is a better time for one problem in the NP complexity level), and is not necessarily in NP.

When first introduced to the problem, we decided to try Kruskal's algorithm as our base form of thought and improve upon that in our code to try and find our lowest cost solution. We chose Kruskal's based on a realization that if you allow yourself to backtrack across smaller edge weights, you can save overall costs on travel. Of course, a true minimum spanning tree (MST) created using any of the algorithms you research or learn about does not have cycle allowance, but taking this rule out of finding your shortest path can actually reduce your cost if you allow yourself to travel back to areas to cut the total down. We looked into many different algorithms, mainly detailed in the Heuristics for the Travelling Salesperson Problem paper, written by

Christian Nilsson, and decided to take ours and run with it to see how it would work out given a varying number of graphs to work with.

The algorithm we coded doesn't necessarily follow one blueprint, but can best be described as a combination of Tabu-Search and Kruskal's. First, we create an "MST" using a modified Kruskal's sort to get all of our shortest edges that best connect everything together. The reason why it's not really an MST is that when we create our graph, we allow technical cycles to be created to better link everything together and have more options looking forward. On top of this, we allow one edge per vertex so we have *n* edges for *n* vertices to use. We only use this MST when finding our best path, up until the last step.

Next is the Tabu-Search. Tabu-Search is an addition to a neighborhood search that will go through the connections a node has to find the best candidate to move onto next. The issue that this algorithm usually runs across is that it can often get stuck in a cycle, so we just implement the Tabu-Search to better traverse through our graph. We use the Tabu-Search by using the shortest available edge (removed by a 2-opt move), and making our next move based on our next location from this edge. We go through our MST to find our best scenario, travelling from one place to the next, with our best weights in mind (prioritizing new vertices/towns that we haven't visited yet), and travel until we have hit every town at least once. At our final town, before returning back home, we then calculate two things. We test if the direct path home is shortest, or if the MST travelled to get to the point is the shortest. They could be the same, either way we attempt to find our best path all the way home. Using this as our final test also takes care of graphs that satisfy the Triangle Inequality.

For most cases, our program runs well, outputting the best-case path to take to ensure the shortest cost. The only issue is running time with an algorithm like this. Worst case scenario: using an algorithm like Tabu-Search will result in a running time of $O(n^3)$, which is slower than using a 2-opt search going through the graphs. The general process is longer, despite having less edges to work with, it's still basically recalculating at each neighbor, which would ultimately add up to the total amount of edges, and possibly a bit more with any possible back-tracking. On top of this processing, when calculating the route back home, we're adding that onto the total as well, not including when first creating the graph to then decide our MST. The algorithm works well over most cases given, for some graphs the cost is a bit more than what we could come up with on our own, but never by a large margin - it stays within the difference of the largest edge we use. Meaning that our best solution we can find, if it was different from what our algorithm found, the difference was typically only the weight of the largest edge we had included in our "MST." Overall well done, minus a few hiccups on some larger graphs, but it is a heuristic algorithm, so it won't solve everything perfectly, but will give its best solution.

References:

Loeffler, John. “P Vs NP, NP-Complete, and an Algorithm for Everything.” *Interesting Engineering*, 4 May 2019.

Nilsson, Christian. “Heuristics for the Traveling Salesman Problem.” *Linköping University, Sweden*, 2003.

“Kruskal’s Algorithm – Minimum Spanning Tree (MST) – Complete Java Implementation.” *Algorithms, TutorialHorizon*.

“Graph – Find Cycle in Undirected Graph using Disjoint Set (Union-Find).” *Algorithms, TutorialHorizon*.