Peyton Scherschel

Algorithms Project 1 Report

01/29/2020

1. Details of implementations:
   1. Nearest Neighbor Approach:

For the nearest neighbor approach I implemented the concept of taking a starting point, then iterating through all other points checking how far away each point is, and then choosing the closest point to be my next starting point, where I repeat the process ignoring the points I have already visited, until I get to the last point which I will return to the first point from. I accomplished this efficiently by recursively going through a list of points. Each recursive call adds the current start point to the end of my path history and then in one for loop that iterates through only the remaining points I find the closest point. Once I have checked all remaining points to find the closest, I recursively call the function again until there are no unvisited points remaining at which point, I add the original point to the path again. My algorithm is efficient as it only checks the points that need to be checked and does so in one for loop.

* 1. Exhaustive Search Approach

For the exhaustive search approach, I implemented the idea of generating all permutations of paths that start and end at the chosen first node. Once I got all the permutations, I checked each one for the total path length to find the shortest one. I did this one efficiently by generating all n! permutations up front and then iterating through them all once. This is the only and fastest way to check all n! permutations. In code I used the itertools python library to generate all n! permutations of my list. It is important to note that I remove the first element that we have defined to start and end with, so I essentially generate (n-2)! Permutations in comparison to the final path length. This saves lots of time, especially as n gets larger. Once I get all the permutations that I need to check I iterate through them and their items once to calculate their total path lengths and chose the smallest one.