Hayden Donofrio

Lab 3 Write-up

A screenshot of a cell phone

Description automatically generated

Here is a lot of the raw data that I have gathered from the Brute Force and the Dynamic programming approaches to the TSP. As shown here. The Dynamic Programming algorithm performs phenomenally faster than the brute force algorithm. This is mainly due to the lowering amount of combinations that are needed to be tested as the cached values in the dynamic approach save many different later operations that the dynamic approach calculates many times. This is shown in both time and the nodes explored as the test size increases, but accuracy remains the same.

This graph puts it into scale how much the dynamic algorithm improves over the brute force approach. It looks like the dynamic approach is hardly increasing at all while the brute force approach looks exponential, perhaps even factorial.

I have derived the time of my Brute Force algorithm to not be the typical (n-1)!/2, but to be n!/4. I believe this is because of one of the approaches in my code where I include the operation to return back to the original node rather than calculating that after finding the best path. Had I removed that last part, we would most likely see a typical (n-1)!/2 time complexity.

This is a comparison of the execution time of the dynamic programing vs 2^(n-1)\*(n-1)^2. As we can see this is a pretty close comparison to what the recommended performance for dynamic programming. This is significantly faster than n!, but this still would take a very long time for charts of 50+ cities rather than just 12. But for what we have now, this is not a bad approach.

**CONCLUSION:**

Due to what we have gathered so far, I have come to the conclusion that the brute force is within the asymptotic bounds of 0(n!) while the dynamic programming approach is within O(2^n). Because of this we can conclude the dynamic programming approach will nearly always be a better approach than the brute force approach for any reasonable calculation.

**DESIGN:**

For my design for my project, I focused on a decorator pattern for the execution and a builder/ factory pattern for the handling of the algorithm/graph information. I chose a decorator since it would be very simple to create a new algorithm and tack it on to the base decorator while still having a good interface. Or if we wanted to add new ways to handle the salesman problem, we can tack that on the salesman class. The decorator makes it very easy to delegate issues to classes and figure out which method you want to execute when.

For the Factory/Builder I used that since I thought it would be very easy to use the builder as a way to hold the matrix data, as it was a lot more difficult to delegate such data with a decorator pattern. That is one flaw in my design, it is very difficult to pass along large data members to a child class without having to add extra methods to the decorator. However, I managed to use the factory for easy creation of the algorithms and it was overall very fast and efficient for handling data, although there was a bit of headway along the way. My design will lead to very easy adding of algorithms or adding new ways to solve TSP