Christopher Moroney

Arias

CSC 3430

Due 13 March 2020

Extra Credit Assignment

A classic example of an NP problem is the Travelling Salesperson Problem (TSP). In this problem, a salesperson is travelling to many different cities, and needs to find the cheapest overall price in order to reach every city. We can use a Graph data structure to resemble the TSP, in which nodes are cities or intersections, and the edges, which are directed and weighted (this is a must, or else this problem could possibly end up becoming non-NP, or rather could be run with a greedy algorithm), are roads with traffic time or airfares with flight costs or distance. A real-world application of the Travelling Salesperson Problem is for a person to be able to reach multiple destinations using a maps application, such as Google Maps. For instance, if a person needs to reach City A, B, and C, which order of cities is best for the Travelling Salesperson to take that is most efficient? Suppose the person also does not want to take toll bridges, and prefers to avoid the highway unless absolutely necessary. TSP becomes an NP problem because the algorithm needs to look at almost every single subset in order to satisfy the conditions of the user, as well as to find the cheapest option to visit every node. Another real-world application is that the Travelling Salesperson would also have to reach home, so the algorithm would have to take the person home efficiently only after he or she has reached every city already. This particular problem is very important to solve because in the business industry, time can oftentimes be more valuable than just money. A company with just money and no time is just rich, but a company or person with time has potential. Saving people’s time is incredibly important for business, education, and personal reasons for meeting deadlines and requirements as necessary. Being able to navigate time and expense also can influence people's decisions on when to take specific flights, or when to drive to locations as necessary, so long as it is the cheapest possible method for the sakes of the user.

Solution

The solution is based on the ***Multi-Fragment algorithm***, which provides a methodology for approximate solutions to the travelling salesman problem. I will demonstrate it using a 10-city subset. While this cart doesn’t show *directed* edges, the problem should consider that because the cost of flying from City A to City B is not necessarily the same as from City B to City A.



Real-world data can be gathered for each of these edges, taking the cheapest possible airfare between them. I won’t populate them all here, but show a few examples.



The algorithm requires identifying the cheapest airfare in the system. In this case it is Seattle to Portland, which I’m highlighting in red.



I now create a graph that joins these together to indicate that I’ve indicated that I’ve done this one.



I now find the *next* closest two cities, and join them together where joining them **does not result in a closed tour**. So for example, it will be LA->Phoenix.



So my tracking graph has these nodes added:



I continue to repeat this process, finding the shortest, unlinked cities, avoiding closed tours. If any city has more than 2 edges, I pass on it until I reach a point where I can minimize the overall number of edges to visit every city. My graph then would look like this.



Note that this algorithm repeatedly uses the cheapest fare between cities to find the next pair of cities and link them into the graph.

This is a simplification for single direction connection...the process would have to be repeated for 2-way airfares being different as described above.

Works Cited

*Multi-Fragment*, users.cs.cf.ac.uk/C.L.Mumford/howard/Multi-Fragment.html.