CS 5800: Final Project Report

Team Members: Yongzheng Li, Rohen Benjamin Varghese, Shichun Xuan

I - Introduction

Given the opportunity of this course project, the team of group 8 are interested in finding the most efficient way for them to plan flights based on various factors such as price, flight times, number of stops, etc. Flight planning can be a complex decision problem in real life, and the team aims to develop a solution that offers high adaptability to meet various needs by applying different algorithms from the course of CS 5800.

Shichun Xuan: When traveling by air I always wanted to find the best flight that suits my needs. Although the purpose of a flight is to get the passengers to their desired destination eventually, there are many factors to consider when making the decision. Things like price, flight time, number of stops are also crucial for the flight experience. And it is tough to balance and decide which to after sometime. Therefore, I wish through experiment and analysis we will be able to provide a feasible solution to this problem, with high adaptability to realistic needs.

Yongzheng Li: Graph algorithms have many applications in real day life. Dijkstra's algorithm has been widely used in traffic flow and navigation systems. Bellman-Ford has been used in the financial system. I did not learn graph algorithms before and I believe this project is a great opportunity for me to implement the algorithm in real cases. It not only helps me better understand the algorithms, but also improves my abstract thinking. Flight travel planning is a relatively simple case we would use and it is useful in traveling to find minimum time/cost between two airports.

Rohan Benjamin Varghese: In today's world people always want to find the most efficient way of doing things. I do as well and trying to find the best deals on flights is something that always gives me a headache. As a passenger I mainly consider the flight duration and cost while booking a ticket. In this project we try to solve that headache by finding the most efficient way to travel using Dijkstra and Bellman-Ford algorithms. The Dijkstra and Bellman-Ford algorithms are classic algorithms in computer science for finding the shortest path between nodes in a graph, which can represent, for example, road networks. For optimizing a flight travel system, Each airport can be considered a node, and each flight between airports can be considered an edge. The weight of each edge can be based on various factors such as flight duration or cost.

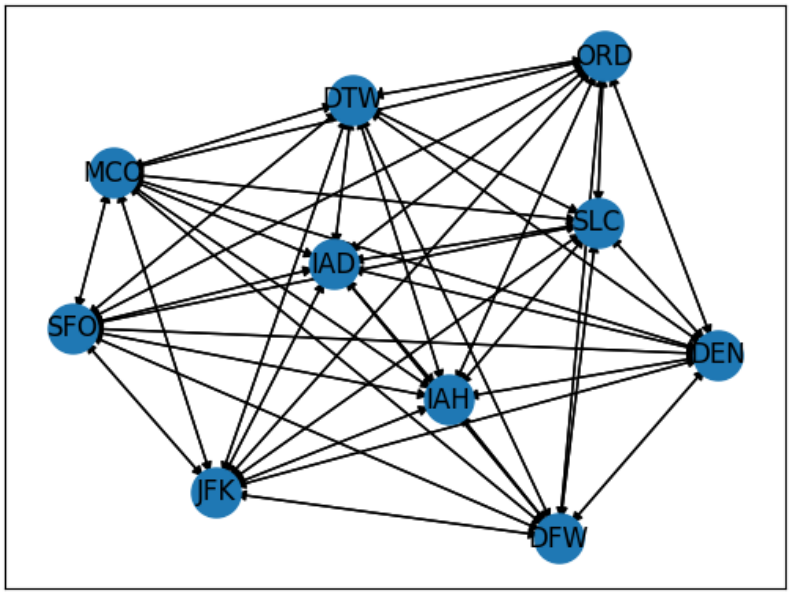
II - Technical Discussion & Analysis

1. Data

To enhance coverage of realistic scenarios, the team has selected the ten largest airports in the United States as the vertices of the graph. Each of these airports is connected to all the others by edges, resulting in a total of 90 edges within the graph. The weights for edge calculations include prices in dollars and durations in minutes, as described in the following section of this report.

During the data collection process, the team has taken extra care to reduce the impact of irrelevant factors, such as specific airlines, temporary discounts, and dates, in order to maintain data consistency and reliability.

Below is a representation of the resulting graph:



2. Method

The extracted data is cleaned, processed and the feautres ‘Origin’, ‘Destination’ and ‘Cost’ (representing Price/Time) are added into a list which is used as a graph. The ‘Origin’ and ‘Destination’ features act as the vertices and the ‘Cost’ serves as the weight. The NetworkX package is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks. Using Python’s NetworkX package the Dijkstra and Bellman-Ford algorithms can easily be applied to the graph

using the ‘all_pairs_bellman_ford_path’ and ‘all_pairs_dijkstra_path functions which operate using the below given pseudocodes.

Pseudocode for Bellman-Ford Algorithm

```
function BellmanFord(Graph, source):
    // Step 1: Initialize distances from source to all vertices as
infinite and distance to source itself as 0
    dist[source] = 0

    // Step 2: Relax all edges |V| - 1 times
    for i from 1 to |V|−1:
        for each edge (u, v) in Graph.edges:
            if dist[u] + weight(u, v) < dist[v]:
                dist[v] = dist[u] + weight(u, v)

    // Step 3: Check for negative-weight cycles
    for each edge (u, v) in Graph.edges:
        if dist[u] + weight(u, v) < dist[v]:
            return "Graph contains a negative-weight cycle"

    return dist[]
```

Pseudocode for Dijkstra’s Algorithm

```
function Dijkstra(Graph, source):
    create vertex set Q

    for each vertex v in Graph:
        dist[v] ← INFINITY
        prev[v] ← UNDEFINED
        add v to Q
    dist[source] ← 0

    while Q is not empty:
        u ← vertex in Q with min dist[u]

        remove u from Q

        for each neighbor v of u:  // only v that are still in Q
            alt ← dist[u] + length(u, v)
            if alt < dist[v]:
                dist[v] ← alt
                prev[v] ← u

    return dist[], prev[]
```

The ‘all_pairs_bellman_ford_length’ and ‘all_pairs_dijkstra_length’ functions provide us with minimum cost between the vertices as well. Once the algorithm is applied, Python’s Pandas package is used to export the results to a data frame for better presentation.

3. Result & Analysis

The table presented below illustrates the optimal travel routes between any pair of selected airports. Cells highlighted in green indicate instances where there is a potential enhancement in travel efficiency by choosing an alternative route instead of opting for a direct flight.

	DEN	DFW	DTW	IAD	IAH	JFK	MCO	ORD	SFO	SLC
DEN	DEN	DEN-DFW	DEN-DTW	DEN-SLC-IAD	DEN-IAH	DEN-JFK	DEN-MCO	DEN-ORD	DEN-SFO	DEN-SLC
DFW	DFW-DEN	DFW	DFW-MCO-DTW	DFW-IAD	DFW-MCO-IAH	DFW-JFK	DFW-MCO	DFW-ORD	DFW-SFO	DFW-MCO-SLC
DTW	DTW-DEN	DTW-MCO-DFW	DTW	DTW-IAD	DTW-IAH	DTW-JFK	DTW-MCO	DTW-MCO-ORD	DTW-MCO-SFO	DTW-MCO-SLC
IAD	IAD-DEN	IAD-DFW	IAD-MCO-DTW	IAD	IAD-IAH	IAD-SFO-JFK	IAD-MCO	IAD-SFO-ORD	IAD-SFO	IAD-DEN-SLC
IAH	IAH-DEN	IAH-MCO-DFW	IAH-MCO-DTW	IAH-IAD	IAH	IAH-JFK	IAH-MCO	IAH-MCO-ORD	IAH-SFO	IAH-MCO-SLC
JFK	JFK-DEN	JFK-DFW	JFK-MCO-DTW	JFK-IAD	JFK-IAH	JFK	JFK-MCO	JFK-ORD	JFK-SFO	JFK-SLC
MCO	MCO-DEN	MCO-DFW	MCO-DTW	MCO-SLC-IAD	MCO-IAH	MCO-DTW-JFK	MCO	MCO-ORD	MCO-SFO	MCO-SLC
ORD	ORD-SFO-DEN	ORD-DFW	ORD-MCO-DTW	ORD-IAD	ORD-IAH	ORD-JFK	ORD-MCO	ORD	ORD-SFO	ORD-MCO-SLC
SFO	SFO-DEN	SFO-DFW	SFO-ORD-MCO-I	SFO-IAD	SFO-ORD-IAH	SFO-JFK	SFO-ORD-MCO	SFO-ORD	SFO	SFO-DEN-SLC
SLC	SLC-DEN	SLC-MCO-DFW	SLC-MCO-DTW	SLC-IAD	SLC-IAH	SLC-JFK	SLC-MCO	SLC-MCO-ORD	SLC-SFO	SLC

The table below provides the potential costs associated with alternative travel paths. Departing airports are listed in the columns, while arrival airports are represented in the rows. In our assessment, a lower value of the price-to-time ratio is considered to indicate the shortest path. In the case of routes with transitions, we calculate the sum of the ratios for the two paths involved.

	ORD	DEN	DFW	MCO	IAD	IAH	SLC	SFO	JFK	DTW
ORD	0	0.56	0.35	0.4	0.72	0.68	0.7	0.27	0.49	0.53
MCO	0.14	0.43	0.16	0	0.48	0.28	0.3	0.41	0.36	0.13
DFW	0.25	0.67	0	0.29	0.55	0.57	0.59	0.46	0.51	0.42
SFO	0.25	0.3	0.27	0.5	0.45	0.66	0.43	0	0.29	0.63
DTW	0.3	0.5	0.32	0.16	0.64	0.44	0.46	0.57	0.52	0
IAH	0.36	0.35	0.6	0.44	0.64	0	0.74	0.63	0.46	0.54
DEN	0.53	0	0.41	0.45	0.42	0.46	0.53	0.28	0.38	0.45
SLC	0.55	0.25	0.57	0.41	0.67	0.69	0	0.53	0.4	0.54
JFK	0.59	0.54	0.61	0.82	0.9	0.58	0.52	0.45	0	0.66
IAD	0.94	0.91	0.87	1.07	0	0.83	0.66	0.76	1.05	0.93

We observed that both algorithms yield comparable outcomes in this dataset. Specifically, Dijkstra's algorithm runs quicker on the dataset for the majority of cases when compared to the Bellman-Ford algorithm. Nevertheless, it's worth noting that the Bellman-Ford algorithm is better suited for situations where negative weights are a possibility. While our current weight assignment methodology does not introduce negative weights, they may arise if we choose to involve additional factors into the weight calculation ratio in the future.

The time complexity of the Bellman-Ford algorithm is O(V.E) where V is the number of vertices and E is the number of edges in the graph. This is because it relaxes all E edges V-1 times.

The most efficient implementation of Dijkstra's algorithm uses a Fibonacci heap, reducing the time complexity to O(V log V + E).

III - Conclusion

1. Limitation and Weakness

While we have successfully generated feasible results that can aid in real-life decision-making processes, we acknowledge that our current algorithm may not be sufficient to cover all factors that can influence flight planning. To enhance both the accuracy and practicality of our solution, it is crucial to extend the scope and incorporate additional real-life aspects, including airline, time of the day, layover durations, baggage fees, and more. By accounting for these factors, not only we can significantly improve the credibility of our algorithm's outcomes, but also allow for more customizable and context-aware decisions in flight travel planning.

2. Team Reflections

Shichun Xuan: The final project of the course grants us a chance to practice and utilize what we learned in class and help me to understand how we could apply algorithm into realistic problems especially when it is related to my life experience. Meanwhile, by recognizing the limitation of this project, I find myself more motivated for improvement and the development of more robust and effective software solution in the future..

Yongzheng Li: This project helps me understand how to find appropriate data for the algorithm. Due to time constraints, we do not put in depth on improving the algorithms. I am interested in taking future courses related to graph algorithms so that I can learn more about advanced graph algorithms. Also, I would like to have more projects related to graph algorithms.

Rohan Benjamin Varghese: Working on the project gave me an in depth understanding of graph algorithms. Applying the algorithm on real time data provided me with the understanding of choosing the right weights and vertices. I believe this project could relieve me of my headache of trying to find the most efficient way to travel in the future. With more data and time, the graph algorithms can really help me or other passengers find more efficient ways of flight travel.

IV - Cite References

Libraries:

NetworkX: NetworkX is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks.

Functions:

- DiGraph : A DiGraph stores nodes and edges with optional data, or attributes.DiGraphs hold directed edges.
- all_pairs_bellman_ford_path: Compute shortest paths between all nodes in a weighted graph
- all_pairs_bellman_ford_length: Compute shortest path lengths between all nodes in a weighted graph.
- all_pairs_dijkstra_path: Compute shortest paths between all nodes in a weighted graph
- all_pairs_dijkstra_length: Compute shortest path lengths between all nodes in a weighted graph.

Pandas: Pandas is a python package that is used to visualize the results obtained from the code in a tabular form (data frame).

Sites:

https://www.kayak.com/explore
https://networkx.org/documentation/stable/reference/functions.html