CSC111 Project Phase 2: Dream Flight

Aaditya Mandal, Dinkar Verma, Faraz Hossein, Yousuf Hassan

Friday, April 16, 2021

Problem Description and Research Question

Our goal for this project is to find the most convenient* route between airports via flights.

Since 2017, there has been an average of over 100,000 flights per day, excluding post-COVID statistics (Asquith). The airline industry is very complex. At any airport, in a hub city, there are dozens of planes waiting to take-off and land at any given moment. Several different airlines will transport passengers to various cities across the globe. Then we must account for domestic airlines that fly passengers across a country or continent! If we try to visualize this, we can already imagine the enormous amount of direct plane routes (with no stoppages or layovers)! The picture gets even more complex when we factor in routes that have layovers in different cities! A powerful tool is required to visualize all this data in a clean and organized way. Enter: Graphs! Not only will graphs allow us to visualize this vast amount of information in an orderly fashion, but they will also allow us to perform meaningful operations on it! Some background information to familiarize yourself with this problem domain is the information required about airports, airlines, and flight routes. Starting with airports, the main pieces of information we should keep track of is their names (along with their codes) and their location (as longitude and latitude coordinates). For airlines, we must notice that not all airlines travel to all airports! Domestic airlines don't travel internationally, and some international airlines won't land at smaller airports (you must take a connecting domestic flight to arrive at your final destination). With this information, we can create more accurate, although more complex, flight routes. Another important detail about flight routes is that they don't travel in a straight path from one city to another! In fact, due to the Earth's curvature, the shortest and most efficient flight path that planes take will look curved. The circumference of the Earth is greatest at the equator, so a more efficient route is closer to the poles (Mazareanu)! Additionally, planes try to fly over land as much as possible because it is easier and safer to make emergency landings, as compared to the ocean!

In general, graphs are the most convenient way to visualize and model a network of airports. Each airport, represented by a vertex, is connected to other airports. This connection forms a flight route, representing an edge in our graph for the corresponding airports. Although most airports connect in this manner, this connection does not necessarily represent the fastest route to the corresponding location. Usually, a direct flight is the most efficient route, but certain constraints might lead the aircraft to use a "path" connecting multiple locations. Our problem now is more focused on actually creating these paths based on the user's inputs. It is quite simple to provide a direct flight from one city to another since this path is directly located in one row of our 'routes' CSV file. Hence, the problem becomes complex when we need to consider whether the user is fixated on using a desired Airline during the entire flight, regardless of whether there are 0, 1, or 2 stops during this flight. As the implementers, we can allow the user to choose their desired constraints on their flight from city A to city B such as choosing an airline and choosing specific cities they would like to stop at during this flight. We all love taking vacations and travelling the world, but sometimes airlines have delays and may not use the ideal route to reaching one's desired destination. It annoys us to spend so many hours taking connecting flights that may not be the fastest route. Hence, this motivates us to find the most efficient route an aircraft should take out of several possible routes connecting two locations. Although, this is only considering direct flights as our most efficient and ideal route for the passenger. Passengers may have a different plan of forming a pathway during their flight to explore certain cities around the world. This can be a hassle to figure out without a pre-existing program based on the user's demands and without programs that provide efficient routes based on these desires. Hence, we decided to create a flight visualizing software that finds efficient routes, based on the user's inputs, to travel between two or more cities.

* Note: Our definition(s) of "convenient" is explained in more detail in our computational plan.

Datasets

Starting off with our first data set, contained in the airline.csv file, is all the necessary data regarding airlines themself. It contains data regarding an airline's IATA which is used in our third dataset, its name, country of origin, whether the airline is active or not, and a few more columns which have been disregarded since they have not been used and are irrelevant. Within our program specifically, we have not actually used this data set for programming purposes, but it was crucial for testing. Our second data set deals with data based around airports around the world, which is contained in the airports.csv file. Disregarding the irrelevant columns, this data set covers the airport's name, city it is located in, the country, IATA which corresponds to the identification given to a specific airport, the airport's latitude and longitude values, and so on. Our program specifically applies the columns, Name, City, IATA, Latitude and Longitude. Our third and final data set covers the most important information regarding our program, the direct routes between airports around the world, where this data is contained in the routes.csv file. It holds information such as the airline's IATA, the source or starting airport's IATA, the destination airport's IATA. We have refrained from covering the other columns since they are not relevant to our implementation. For the program itself, we used the columns, Airline ,Source airport, and Destination airport.

Computational Plan

Overall, this project deals with flight routes that are directly applicable based on our datasets. Our data set contained in the airports.csv file, necessary data related to airports such as its name, location, IATA, as well its latitude and longitude coordinates necessary for plotting onto a map. This data is used to represent each vertex in our graph, as well as represent an airport in our map. Our data set contained in the routes.csv file contains the routes created by a certain airline from a source airport to a destination airport. These airports are directly the IATA codes contained in the airports data set. This routes data set is used to form the relationship between airlines and the routes is it capable of doing. Since each row in the routes.csv file represents a pathway between two airports, we can use this connection as our edges for our graph. As a whole, by using airports and routes, we can create a graph where airports represent each vertex, where an edge represents a connection between airports either based on a specific airline or simply without an airline.

Our program uses the routes.csv and airports.csv to create a graph of all possible routes connecting one airport to another. Although, it simply does not use these files as our entire graph for all of our functions. The size of routes is too large and requires a lot of running time to create a graph, using our load_route_graph function for this file. Hence, we are using 3 functions to create a smaller graph, by filtering out unnecessary rows in the routes file. These 3 functions cover the scenarios where the user chooses either zero stops (a direct flight), one-stop, or two stops. In our zero_stops function, we are only requiring the user to choose an airline or not since a direct flight will not have any layovers. Hence we can filter out each row in the routes data set, that does not either cover the chosen airline or does not contain the starting/destination city's airports. In our one_stop case, we require the optional input of an airline, starting and destination city, and an optional input of one specific layover city. Based on these inputs, again we simply filter out the unnecessary rows from the routes data set. Similarly in our two_stops function, the only difference here is that an additional optional layover city is used but this filters out fewer rows of our routes data set. This allows us to filter out unnecessary rows from the routes data set while reducing our running time significantly. Then, there are 3 more functions used corresponding to direct_trip, one_stop_trip, and two_stop_trip, that check and return all possible paths the starting and destination airport are connected, alongside the additional optional parameter of the user's desired airline. Based on this graph data, as well as our newly found paths, we are able to take this data and incorporate it with our visualization webpage which, as an over generalization, puts these newly found routes onto a world map. A detailed explanation will be provided as you keep reading on.

Instructions

Instructions:
Install all Python libraries listed under requirements.txt.
Download the zip file containing our three csv datasets which we have sent to the course email address using UTSend.

Description of program and explanation of features
When you first run the main file, the console should say "Dash is running on http://127.0.0.1:8050/." Click on the link and a web page will open in your browser.

On the left hand side of the screen, you will see two buttons on the top. The leftmost button corresponds to the information page and the rightmost button corresponds to the map page (current page). Below the buttons are dropdown menus that need to be filled out based on your desired trip. The right side of the page contains the world map to display flight routes!

To view the information tab, simply click on the blue button. To go back to the main screen, click on the map button.

Back to the main page...

- The first dropdown menu, number of stops, allows you to choose how many stops you would like to take throughout your flight. Based on your choice, 1 or 2 more dropdowns may appear below the origin city dropdown called stop 1 and stop 2.
- The origin city dropdown allows you to choose your starting location and below it you can select which cities you would like to stop at (assuming you don't want a direct flight). Then you can choose your destination city and your preferred airline.
- To view the flight based on your choices, simply click the view flight button.
- As a reminder, if the number of stops, origin city, or destination city dropdowns are left blank, no flights will exist (these three values are mandatory). Also, the stop cities and preferred airlines dropdowns are optional, meaning we will select the best flight regardless of these inputs (assuming you entered the mandatory parameters).

Here is a flight visualization with 0 stops from Toronto to Montreal

Additionally, You may zoom in or out on the map and hover over the dots (vertices), representing airports. This will provide detailed information about the corresponding airport, such as its name, IATA, the city and country it is located in, as well as its latitude and longitude coordinates on the map. If you hover over the centre of any displayed path, you will see the IATA of the two connected airports via the corresponding path. This value corresponds to one of the routes listed on the legend! Located on the right hand side of the screen, you will see the legend of the map. You may click on any of the listed routes to either hide/show them on the map. You may double click the corresponding route in this legend to only display this route on the map. At the bottom of the legend, the blue dot represents all airports of the corresponding routes listed above. You may either single click this button to hide/show all the airports, or double click to hide/show all the paths (edges) on this map. (INCLUDE IMAGES)***

Changes between proposal and final submission

The changes regarding our project plan simply start from our purpose itself. Our goal has changed from efficient routes to user desired routes to provide a more interactive experience instead of simply having direct flights that are the most efficient, from one airport to another. There are certain cases where the user does not care about choosing their layovers, hence we decided to direct the efficiency factor into those cases by choosing a set amount of efficient paths from one airport to another. Additionally, we have not implemented a plane travelling across the screen using the produced flight route since this was simply a visual quirk of the program, which did not provide much computational complexity to our program, but a better visual. Additionally, this feature is already available in the airplane itself during one's flight, allowing the user to rely on that fact. An additional constraint within our program is that the number of layover stops has been limited to a maximum of two, due to our worries on running time of our function and realistic scenarios. Although additional stops would provide further interaction with the user and our program, having three layovers during a flight is not realistic for an average passenger; hence refraining our program to allow having over two stops.

Discussion

Firstly, our computational work regarding creating routes based on user inputs directly helps us visualize them onto a map. Although, this is only considering all of the possible routes a passenger could take to get from one city to another. The more layovers a flight contains, the greater the number of possible routes is possible. Our program allows any possible path to be created and computes its distance and time, which is used to compare against other possible paths. In retrospect, no matter how many layovers there are, the user can visualize a set number of efficient paths they can take from one city to another. Our program does not simply output all possible paths, but the most efficient paths, without the user's desired layovers. Now, if they decide to choose their layovers, our job becomes direct since we can visually provide them if such a path exists. A certain path may not exist if all airports from the starting city do not connect via a path, to the destination city's airport. Additionally, since our program allows the user to choose their desired airlines, we need to further filter all possible routes, to figure out if this airline can make the trip or not. The more we rely on the user's inputs, the less of a chance there is that they may choose an efficient route. Hence, our program uses the haversine calculator as a step in computing efficient routes, when the user does not decide on specific layover cities. This is the computational power behind allowing the user to choose efficient routes, or choose their desired routes if they exist. Hence, this allows us to conclude that our program nicely covers a realistic amount of scenarios in the travelling lives, as well as accomplishing our goal.

Moving on, we faced a limitation within plotly for the add_trace function. More specifically, we wanted to add a trace to the plotly.graph_object, as a graph_object.Scattermapbox(). When we added the edges between each node, which represented flight routes, we wanted to create a label that would display information about the flight, such as the flying distance, the duration of the flight, and things of that nature. However, Scattermapbox was not capable of having labels for graph edges; it only had them for nodes. So, to overcome this, we calculated the coordinates of the center of each graph edge and created a transparent graph node at that spot. The only purpose of that node was to act as a label for the graph edge. A major we faced was regarding the size of our routes.csv file. Due to its sheer size, our functions were quite difficult to implement with considerably low running time. We had to create multiple different algorithms in figuring out whether an airport is connected via a path (can make the trip) based on a set number of layovers. We tackled this problem by simply filtering down the routes.csv file based on the user's inputs in our two_stops function, contained in the main.py file, to allow us to loop over much a smaller number of rows than the ones contained in the routes.csv file.

There was a limitation with mapbox, such that it would not draw a flight route over the Pacific Ocean. This was because, the mapbox map was not able to go into 'negative' coordinates, so it had to draw the flight route always to the East (towards positive coordinates).

Finally, some next steps for further improvement of our program may cover things such as improving our visualization and allowing the user to input as many layovers as they would like. If a user loves travelling and they come up with crazy ideas of taking a tour around the world, the program needs this feature to allow the user to input additional layovers of their choosing. In terms of our visualization, we could add back our original idea of using our adding a plane travelling across a route on the map. This would require an additional dataset since we need to take into account the speed of the aircraft during the flight, based on its model and its altitude. We could incorporate

real-life weather updates from all around the world, to allow the aircraft to take detours or go around places with terrible weather. This would turn our program into an AI since we would require countless amounts of data allowing us to predict a safe and efficient route, or even update the route in real-time as a very beneficial feature. Additionally, we could explore how routes would work in terms of outer space (limited to our solar system to start). This would require additional research on the physics of outer space. We would need to consider things such as asteroids that may come into the trajectory of our path, the speed of our aircraft, a hypothetical aircraft that could take us from one planet to another in a realistic timeline instead of countless years, using fuel while gaining distance efficiently during our route, and so many more. Graphs are applicable in so many areas of exploration and it excites us to explore further beyond our current program.

References

"Airports, Airlines, and Routes - Dataset by Tylerudite." Data.world, 4 Feb. 2021, data.world/tylerudite/airports- airlines-and-routes/workspace/pro ject-summary?agentid=tyleruditeamp;datasetid=airp orts-airlines-and-routes.

Asquith, James. "Why Planes Don't Fly In A Straight Line On A Map." Forbes, Forbes Magazine, 25 Feb. 2020, www.forbes.com/sites/jamesasquith/2020/02/24/why-planes-dont-fly-in-a-straight-line-on-a-map/.

Mazareanu, E. "Airline Industry Worldwide - Number of Flights 2004-2021." Statista, 2 Dec. 2020, www.statista.com/statistics/564769/airline-industry-number-of-flights/statisticContainer.

"Plotly Python Graphing Library." Plotly, plotly.com/python/.
"Tutorial." Tutorial - NetworkX 2.5 Documentation, 22 Aug. 2020, networkx.org/documentation/stable/tutorial.html.

"Basic Callbacks: Dash for Python Documentation." Plotly, dash.plotly.com/basic-callbacks.

"Dash Core Components." Plotly, dash.plotly.com/dash-core-components.

"Dash HTML Components." Plotly, dash.plotly.com/dash-html-components.