CSC111 Project Report: Random flight trip generator

Keito Hara, Keishi Suzuki, Yicheng Wang

Monday, March 31, 2025

Problem Description and Research Question

Project Goal: Based on the budget, preferred region, and direction of the user, generate possible flight routes for their future vacations.

Planning a trip can be overwhelming, especially when considering multiple factors such as budget, destinations, and available flight routes. Travelers often struggle with selecting a destination that aligns with their financial constraints and travel preferences. This challenge becomes even more complex when multiple people are involved, each with different interests and budgets.

With the vast number of airports and flight connections across the United States, an optimized travel recommendation system could significantly simplify trip planning. Our project aims to address this issue by designing a graph-based travel route recommendation system that suggests potential destinations based on a user's starting airport, budget, and travel preferences.

Our model represents airports as vertices and flight routes as weighted edges, where the weights correspond to distance and airfare costs. The system will take user inputs, including the starting airport, maximum budget, and additional constraints such as the preferred region (e.g., Pacific, Mid-Atlantic) and direction (e.g., East, West). Using graph algorithms, our system will compute the most feasible travel options that meet the user's criteria.

Our goal is to develop an interactive tool that assists travelers in making informed decisions by suggesting flight routes that optimize cost and travel convenience. By leveraging real-world airfare data and graph-based optimization techniques, we seek to create a system that enhances trip planning and offers users a personalized travel experience.

Computational Overview

For this project we used a dataset of airline flight routes with the average airfare between each airport. For the purpose of our project, we removed data which lacked some data values such as coordinates.

Based on this dataset, we constructed a graph where every vertex represents an airport and the edges represent the possible paths between each airport. For each vertex (airport), add the attributes of the airport code, the city the airport is in, and the coordinates of the airport. Also, in each vertex, the neighbor attribute will be a dictionary with the neighboring airports as its keys and a list as its value with the information about the flight route going from the vertex to its neighbor (including distance, average fare, lowest fare).

Figure 1: Sample Data

1. Vertices
   (a) Airport code
   (b) The city the airport it is in
   (c) Latitude of the airport
   (d) Longitude of the airport
2. Neighbors dictionary
   (a) Outer dictionary Key: Neighbor_vertex (airport)
   (b) Attribute:
      i. Distance
      ii. Average fare
      iii. Lowest fare

The dataset that we based the graph on is the US Airline Flight Routes and Fares 1993-2024 from the Kaggle website (See Figure 1). For the exact columns used, please check the function load_review_graph.

First, we only read the columns that we need and stored them into the graph as relevant attributes in each vertex. For every row, which represents a flight route, if either origin or destination is in the graph, then add the destination and its flight information to the neighbor attribute in the origin vertex. Otherwise, create a new vertex for both origin and destination with the needed attribute information (such as airport code). This is done with the add_vertex and add_edge functions in the Graph class.

Based on the starting position and budget, we created a function possible_airports_budget in the Vertex class that gets the set of all airport codes that can be reached from the starting airport based on the given budget. This is achieved by exhaustively searching all airports through recursion. This function is callable in the Graph class through the function airport_budget.

Another function possible_airports_region in the Vertex class returns the set of all airport codes that can be reached from starting airport based on the given budget and is in the specified region. This additionally uses a helper function in the project2_statev2.py which returns the region the airport is in based on the airport code.

Another function possible_airports_direction in the Vertex class returns the airport code that is the furthest away from the starting airport in the specified direction that is reachable by the given budget. This uses a helper function is_in_direction to get whether a airport is in the given direction and only adds that airport into the set of airports in the specified direction. Then, once the recursive calls are finished, all of the airports in the set will be put into a loop to return the airport that is the furthest away.

After either returning a set of airports or a single airport depending on the situation, the user is given the option to ask more questions, whether it is about the path between two airports or which city the

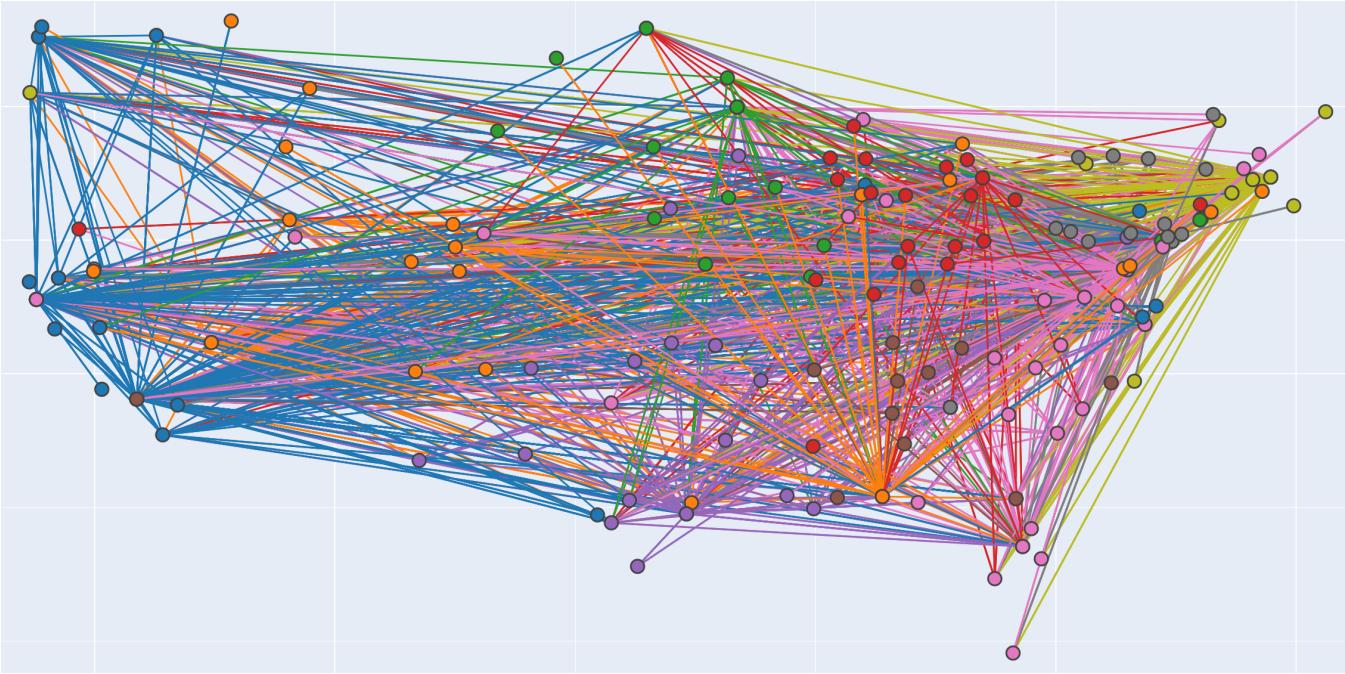

Figure 2: Visualization of entire graph

airport is in.

We also made another function that returns that shortest path between two airports based on either cost or distance (shortest_budget_path and shortest_distance_path). This function is based on Dijkstra's Algorithm.

Visualization implementations:

The visualization is based on NetworkX graph, for each vertex in the original graph we add the attributes including city, region, geo-coordinates. And for each edge, we store the distance, fare, and lowest fare in the edge and these edges are created as undirected based on the neighbours among the original graph.
After creating the graph, we then calculate the positions for each vertex, and this ensures the graph can show a realistic 'map' for all the airports, increasing the precision and readability of the graph.
Besides, we add hover to each vertex and edge so when the mouse is hang on a vertex or an edge, related info like city, position, name of the airport will be shown.
There are two functions for the visualization, the first one (visualization_graph) creates a big picture of the graph and every vertices are well coloured depending on their regions, and the other function (visualize_graph_with_path) is base on the same graph, but fade the colours of those unrelated vertices and edges but emphasizing the vertices and edges that are in the given path with distinctive colours. And the path is put on the top layer.

Instruction

When the main.py file is run, the visualization of the entire graph will show up first (See Figure 2).

Then a text that asks your starting airport will show up in the terminal.

Type your desired starting airports airport code. Then, type your budget.
Then, you will have the option of entering your desired region. If you enter yes, you will be given the result of possible_airports_region.
If you enter no, you will have the option of entering your desired direction. If you enter yes, you will be given the result of possible_airports_destination. If you enter no to even this, you will be given the result of possible_airports_budget.

Then, you will be given the option of asking for more information.
If you enter path, you will be asked to enter another airport code and whether you want the cheapest path or the shortest path, as this will return the path between the starting airport (that you entered at the start) and the new airport.
If you enter city, you will get the city name that the airport that you entered is in.
If you enter new trip, there will be a new iteration of the while loop and the whole process will start all over again.

Also, at any point of this, if you enter end, the code will terminate.

Changes from original plan

There have been some changes to how the Vertex and Graph class will be made. However, overall in terms of the project goal, I don't think that there has been a major change to it.

Discussion

I think that we were able to satisfy our project goal as we were able to return meaningful outputs for the user and create useful visualizations so that it would make it easier for the user to plan their future vacations.
Some limitations are that, due to the large size of the dataset, we are not able to completely understand what happens for example, when there is data on the same flight but in a different year. Another limitation are that for the visualization of the entire graph, there are too many edges that overlap with each other, making it difficult to identify a single edge.
As a further exploration, we could maybe search for datasets that are not restricted to just the US and improve on our visualization so that there is more freedom for the user.

References

Jikadara, B. (n.d.). US Airline Flight Routes and Fares 1993-2024 [Data set]. Kaggle.
"W3schools.Com." W3Schools Online Web Tutorials, www.w3schools.com/dsa/dsa_algo_graphs_dijkstra.php. Accessed 30 Mar. 2025.
"W3schools.Com." W3Schools Online Web Tutorials, www.w3schools.com/python/python_regex.asp. Accessed 30 Mar. 2025.
"Graph." Graph Objects in Python, plotly.com/python/graph-objects/. Accessed 30 Mar. 2025.