CSC111 Project: Ready for Departure!

Charlie Guo, Terry Tu, Owen Zhang, Peng Du

Thursday, April 15, 2021

Problem Description and Research Question

1. Problem Description:
   (a) Currently, countries all around the world are in a struggle with Covid-19. Due to the pandemic, almost all countries have some form of travel restrictions and quarantine rules. These new rules discourage or even ban people from travelling to other parts of the world, which leads tour industries around the world to a low ebb [1]. According to research, international tourism fell approximately 80 percent in 2020 [2].
   (b) However, as more and more vaccines are in production and available to use in the fight against COVID-19, the tide is quickly turning in favour of the return to a more normal lifestyle. Many of those who wanted to travel during the pandemic but were forced to stay at home are growing ever more anxious to travel. In a recent survey by IATA, 57% of those surveyed expect to be traveling within two months of the pandemic being contained [3].
   (c) During the pandemic, safety protocols and travel restrictions forced many airlines to cut seating and route availability in order to save costs as demands dwindled. However, as airlines begin to slowly transition back to pre-pandemic capacity, the surging demand for air travel will undoubtedly outpace airline availability, causing sudden spikes in prices, especially among popular destinations, such as New York and Beijing, as demand exceeds supply. [5]
   (d) In the meantime, many people are expected to have less money post-pandemic due to the financial downturn as a result of the economic shutdowns. For those who may be financially worse off but still need to do essential air travel, they may have the need to look for more cost-effective flight arrangements.
   (e) Therefore, in the midst of increasing demands for air travel with lower seating and route availability, surging prices, and the likelihood of more people looking to save money, it would be sensible to develop a program to assist people and their varying needs in navigating a much more complex post-pandemic air travel arrangement.
   (f) As such, it would be useful to develop a program to support people in picking the most suitable route for their ideal travel plan.
2. **Project Goal: Given the real-world data, how can we establish a flight route planning service that provides optimal(with least cost/least flying distance) connection route of flights for customized requirements?**

Dataset Description

1. We are using the flight route database as the real-world data, which can be accessed via https://www.kaggle.com/open-flights/flight-route-database. This data set contains the 59036 routes between 3209 airports on 531 airlines spanning the globe, as of January 2012. Each of the observations represents a flight route, with the information of airline and air line id, source airport and source airport id, destination airport and destination airport id, and the stops and equipment[11].

2. We are using the airport information includes the location, city database, standard code and other useful message from a rechargeable data set. The data set can be accessed through https://aviation-edge.com/aviation-database/. This dataset contain detailed about related data and we purchased and try to combined this data set with flight route data set to accomplish our project.

3. Note that the data set is missing the prices and flying times of flight routes. Firstly, We tried to use **Scrapy** to retrieve those data from the Internet; However, we found it is hard to get the price from airport website due to the website confidentiality agreement. Then we use the formula to calculate and simulate the rough price(more details see below).

Computational Overview

1. Pre-processing and Building the graph:
   (a) A directed and weighted graph is used to represent flight routes map and such graph is implemented by the System class.
   (b) A vertex in the graph corresponds to a real-world existing airport terminal. Each vertex, represented by the Terminal instance, includes the information of the airport's name, IATA code, ISO2 country code, coordinates, and popularity score.
   (c) An edge in the graph is directed from one vertex to another, representing a flight route between two terminals, with the first vertex in the edge being the departure terminal and the second vertex being the arrival terminal. The edge is represented by the Route class with the following attributes: departure and arrival terminals, airline, flight distance, flight number, and ticket price.
   (d) New Library - Pandas:
      i. Why Pandas: Use module code blocks to make data processing less code and more readable. More specifically, Pandas uses Numpy array, which has the following advantages [7] [8]
         - Uses vector and matrix operations to save coding time
         - Able to join and concatenate multiple data sets, meaning it can be as powerful as structured data queries
         - Uses single data type for each field at a time without type checking to obtain a faster execution time
         - Less Memory
      ii. Functions used from Pandas:
         - read_csv(): Load CSV data and convert it to a dataframe
         - drop(): Drop unrelated columns of the dataframe
         - merge(): Merge two data set into one based on specific columns.
      iii. Data types:
         - Numpy array
         - Dataframe: 2D mutable tabular data
   (e) Data Processing using Pandas: In this stage, features from each data set are extracted and processed in order to be a valid input for the graph. With the help of Pandas, we are able to utilize methods that can handle various tasks efficiently, such as column extraction, slicing, data set merging, etc. The below listed describes major processing computations.
      i. For all data sets, *read_csv* method will be called first to load CSV data and convert them to dataframe.

ii. To combine two data-frame includes routes and airports, we use the merge method in pandas and make the routes as the main key with the "source airport" column. Then we establish a left outer join between the routes and airports data-frame to merge a new data-frame contains the source airports and the destination airports for our next steps. Notice that the departure and arrival vertices makes up an edge in the graph.

iii. Due to the airports websites were hard to get the price message by limitations, we cannot use Scrapy to download the detailed message for the ticket price. Therefore, we use the formula of the how airport set the price depend on the distance and the countries to simulates the price for those airlines in our project. For instance, the price of airline inside a country will count as log(150, distance * 0.6) * distance * 1.1. Also, we add some random section to make the simulation realizable and the price will be more expensive when the airline pass different counties.

iv. Construct such graph class instance incorporating all vertices and edges that are previously loaded.

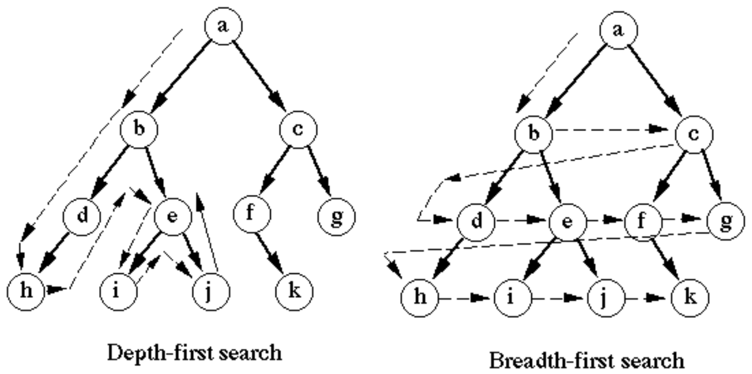
(f) Tidy data:

i. The processed data is stored in a Python nested list (2D array).

ii. Every row of the array has the following attributes all stored in string:

- airline: The airline of the route
- departure_code: IATA code of the departure airport.
- departure_name: Name of the departure airport.
- arrival_code: IATA code of the departure airport.
- arrival_name: Name of the arrival airport.
- departure_lat: Latitude of the departure airport.
- departure_long: Longitude of the departure airport.
- arrival_lat: Latitude of the arrival airport.
- arrival_long: Longitude of the arrival airport.
- departure_country: The departure country
- arrival_country: The arrival country
- departure_popularity: The popularity score of the departure terminal
- arrival_popularity: The popularity score of the arrival terminal

2. Search Algorithms for finding optimal flight routes from airport to airport.



DFS: the algorithm keeps looking for the next possible terminal to travel until it finds the target.

(a) Given the departure and arrival terminals, the function pushes the departure terminals into a stack.

(b) Pop off one element from the stack (FILO), Push all of its neighbour terminals into the stack.

(c) Repeat the above process until the target terminal is founded or there is no more terminals to explore (in this case, there are no routes between start and target terminals).

BFS: the algorithm looks for the all of the possible terminals that can be reached in the next step and keeps exploring step by step to expand the layer of searching.

(a) Given the departure and arrival terminals, the function pushes the departure terminals into a queue.

(b) Pop off one element from the queue (FIFO), Explore all of the reachable terminals within one flight. Push all of the explored into the queue.

(c) Repeat the above process until the target terminal is founded or there is no more terminals to explore (in this case, there are no routes between start and target terminals).

A* Search. Assume the departure and targeted arrival terminal are given, our goal is to search for a path that has the lowest cost.[9]

(a) In A-Star search, for each currently visiting airport (vertex), we are looking for its neighbours i.e. (there is a route from the currently visiting airport to its neighbour). Add the potential flight (edge) to a priority queue based on the sum of previous cost and the estimated future cost to the destination (Denote priority F = G + H respectively). Note that there are different ways of guessing the cost of getting to destination, and so the heuristics are domain specific.

(b) The previous cost g, represents the cost from departure terminal to the currently visiting terminal adding the chosen flight cost to that particular airport. The estimated future cost h, is a heuristic function defined by ourselves that tends to predict the cost from that neighbour airport to the destination airport e.g. (by distance/cost at the current planning phase). Specifically, the heuristic function has the following plans:

i. Given the longitude and latitude of current and arrival terminals, calculate the Euclidean distance from the current location to the destination as the value of the heuristic.

ii. Personalized function based on the estimated ticket price from current to destination terminals.

(c) For each move, we dequeue the least-cost flight (edge) from the priority queue to determine our next visiting airport. Repeat such pattern from departure terminal until we reach the destination and the route we travelled should be the potential optimal route for the trip.

(d) We used A* search to find the optimal flight since it is relatively fast and accurate, in comparison to other searching algorithm, such as DFS.

i. A* always finds a solution because in worst case, it checks all of the possible routes from starting point and as long as there is an existing route between starting terminal and destination terminal, the path should exist in the priority queue, which should be eventually dequeued (Assume we already handle the path checking so there is no infinite cycle)

ii. Fast: It expands the vertices (tries to travel to the neighbour terminals) based on the sum of the current cost/distance and the estimated remaining cost/distance. As long as the heuristic is non-trivial (as we already have a plan above), A* will encouraged to expand the vertex (terminal) that is closer(distance/lower cost) to our goal (lower heuristic value).

iii. Accurate: A* always expand the vertex with lower previous cost for arriving the current point. Suppose there are two vertices that have the same heuristic (estimate to have the same distance from the two vertices to the goal), A* will pick the vertex that has a lower cost from starting point to that vertex (same h and lower g value).

3. Route planning from 5 preferred departure to 5 desired arrival terminals. In this section, the function searches all 25 possible routes between 5 release and five arrival terminals. The route with the least cost is recorded, constantly updated, and eventually returned by the function.

4. World tour planning: Our project also has the functionality to plan for a world tour. Suppose that a user has a list of cities that he wants to travel to and is looking for the best route that starts from where he lives, passes all those cities, and eventually returns to where he lives. The route may be optimized according to flying time or total cost, depending on the user's preference. Our program takes the city of origin and the list of cities the user wants to travel to and returns the optimal route.

(a) Logistics: Since we are only traveling between the original and target cities, we only need to consider the sub-graph that contains the vertices of original and target cities and the edges between them. Our goal is to find the Hamiltonian circle with the least cost, i.e. the circle that contains all the vertices with the least total flying time/total cost. This problem is also known as Travelling Salesman Problem(TSP).

(b) Algorithm: Even though it takes exceptionally high time complexity to run a brute force algorithm to check through all the possible routes, we will still use dynamic programming to make the computation more efficient.

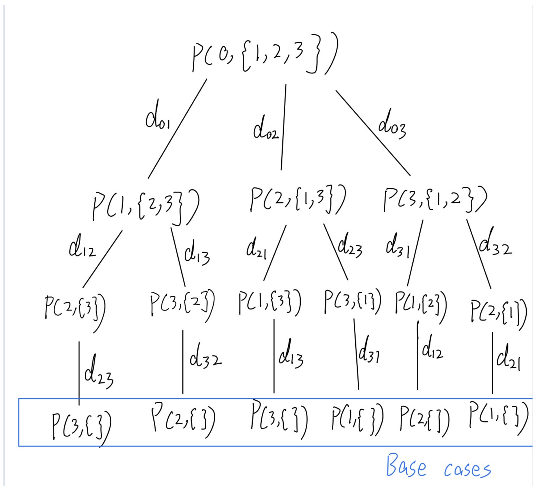
Assume that we start from the vertex s.
Define the function $p(i, V)$ as the shortest path(the path with the least cost) that starts at vertex i, visit all the vertices in set V and finally go back to s. For the parameters of this function, there are two cases:

i. V is empty. In this case, we do not have to pass any vertices, so all we need is to find the shortest path from i to s (d_{is}). This has been solved by the A-star search algorithm, so we can use it directly.

ii. V is not empty. In this case, the problem turns into multiple sub-problems, because we need to check through all the vertices in V, and find the vertex k that has the minimum $d_{ik} + p(k, V - \{k\})$ where d_{ik} denotes the shortest path from i to k.
Thus, our formula for $p(i, V)$ would be:

$$p(i, V) = \begin{cases} \mathrm{d}_{is} & \text{if } |V| = 0 \\ min(\mathrm{d}_{ik} + p(k, V - \{k\})) \text{ for k in V} & \text{if } |V| \neq 0 \end{cases}$$

As an instance, suppose that the user starts from city 0, wants to visit city 1, 2, 3 and eventually come back to 0, then the layers are as below:



Note that the base cases can be computed directly. We will need to figure out each of the nodes from leaves to root, and eventually find the desired $p(0, \{1, 2, 3\})$[10].

After introducing the recurrence relationship and base case of the DP-TSP method, one implementation technique that we apply is **Bitmasking**.

A. Suppose we have a collection of must-go airports which are numbered from 1 to N i.e. labeled as 1, 2, ..., N. According the recurrence relationship, we need to check every subsets of 1, 2, ..., N, in which there are 2^N subsets in total.

B. Specifically, these subsets can be encoded by a sequence of N bits (mask). For example, the mask 0101 means that the subset of the set 1,2,3,4 consists of elements 1 and 3.

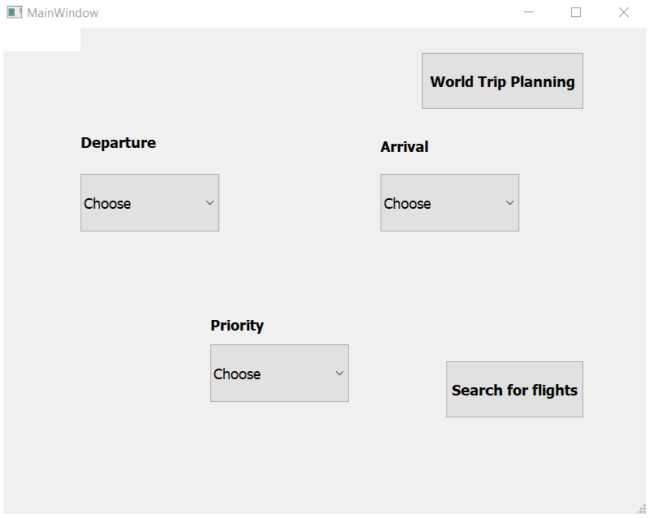
C. Our final goal is to calculate the value for the complete set i.e 1....1, which has N digits so that the shortest path that starts at initial terminals and contains all the must-go terminals will be obtained. While the algorithm implementation go over masks in increasing order of corresponding terminals, each mask is assigned a value and values of new masks are determined based on the already computed masks. For example, say $V = \{A, B\}$, the binary notation would be 11. The $p(A, 11)$ is determined by the computed masks - taking the minimum of $p(B, 01) + d_{AB}$ and $p(A, 10) + d_{AA}$.

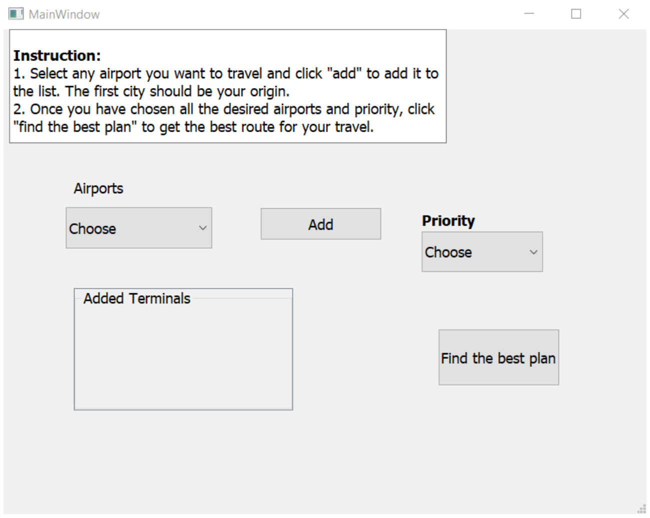
Simulation and Visualization

1. Since it is difficult to predict and forecast the future behaviour for our developed route planning system by itself, we created a simulation to test the methods of the flight planning system and report the effectiveness and correctness of our computation. Several users are simulated, and each of them is randomly using the features of

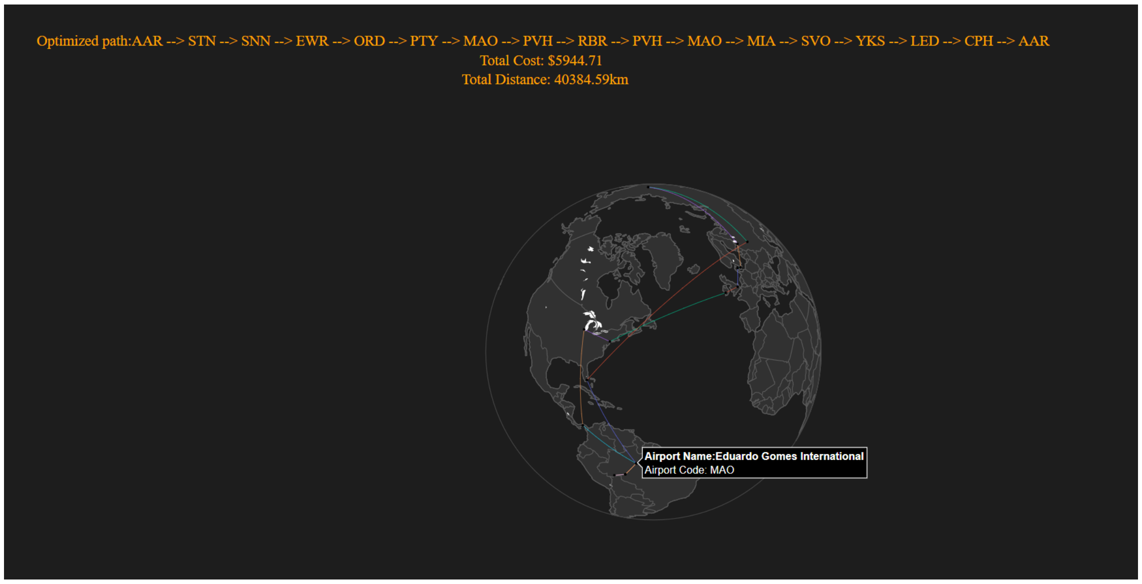
our project to plan for their trips, with different departures, destinations and preferences. The overall structure is similar to the one that was introduced in CSC110, and a Plotly interface is used to visualize the simulation. On a world map, each of the simulated users will be represented by a coloured dot, and their trips will be traced using a coloured line. The simulation aims to justify that our program always works properly and does not stick in any situation. Note that when running the simulation, several interactive HTML webpages of 3D Globe routes are simultaneously generated.

2. Visualization:
   (a) We have implemented an GUI that allows the users to utilize the functionalities of our project.
      i. Package used: The GUI is implemented with PyQt5, which is one of the most powerful and popular cross-platform GUI library. It provides a wide range of GUI objects that developers can take and use, as well as the high flexibility when setting buttons and connecting different pages. Comparing to pygame, the GUIs implemented by PyQt5 are much smoother and less buggy.
      ii. User Manual:
         A. The GUI will pop up after running the file ”main.py”.
         B. On the main page of GUI, users can select a departure terminal and an arrival terminal, as well as the priority(the heuristic that our algorithms use to optimize the routes). Then, users can click on ”search for flights” and the GUI will call the A* algorithm to find the optimal flight route.
         C. If the users want to use the World Tour Planning, click on the ”World Trip Planning” button, and a new page will pop up. On the new page, users can select the airport they want to travel to and click ”Add” to add it to the list. Note that the first airport added to the list should be the starting point, and the rest are the stops in the world tour. After adding all the desired airports, select the priority(the heuristic that our algorithms use to optimize the routes) and click on ”Find the best plan.” Then, the GUI will call the implemented algorithms to find the optimal plan for the world tour.
         D. In either way, a web page will pop up with the optimal flight route and airports plotted on the globe and the route information displayed at the central top. Please refer to the next section for how we have visualized the result.
      iii. Behind the scene:
         A. When looking for the optimal flight route, the program reads the selected item on the three selection boxes and uses these texts as the parameters to call the corresponding functions to get the optimal flight route.
         B. Since the dynamic programming approach of world tour planning only consider the input airports, there may be no direct flights between two stops, in which case the dynamic programming approach cannot find the plan. When this happens, the brute force approach of world tour planning is called because it considers all the airports and can always find the optimal strategy despite its high time complexity.




Instruction:
1. Select any airport you want to travel and click "add" to add it to the list. The first city should be your origin.
2. Once you have chosen all the desired airports and priority, click "find the best plan" to get the best route for your travel.

(b) We have also visualized the flight routes that are found by our algorithm, as well as the relevant information.
   i. Package used: The visualization part is implemented by plotly, a front end visualization package that supports python.
   ii. Logistics:
      A. Each of the airports in the optimized flight route is plotted as a dot on the globe according to their longitudes and latitudes.
      B. Every two consecutive airports are connected by a line, indicating that we are flying from one to another.
      C. The detailed information of the route is displayed at the central top of the web page, which includes the stops, the total price and the total flying distance of the route.
   iii. Interactivity:
      A. When putting the mouse on a specific airport(dot) on the globe, the name and IATA code of that airport will be displayed.
      B. By using the left/right button of the mouse to hold and drag on the globe, users can rotate the globe in whichever the ways they want.
      C. By using the scroll button of the mouse, users can zoom in and zoom out to find the best view of the globe.


Optimized path:AAR --> STN --> SNN --> EWR --> ORD --> PTY --> MAO --> PVH --> RBR --> PVH --> MAO --> MIA --> SVO --> YKS --> LED --> CPH --> AAR
Total Cost: $5944.71
Total Distance: 40384.59km

Airport Name:Eduardo Gomes International
Airport Code: MAO

Software and Data

Software Instruction:
To run the project, the audience will need to have Python 3.9 installed on the computer. All the required packages are included in 'requirements.txt' file, and please make sure that all the requirements are satisfied before running our main program. Some of the packages are quite large and may take some time to install, so we kindly ask you for your patience and understanding.

Data set Instruction:
As discussed in the previous sections, two data sets are used in our project. The URLs of them are as follow:

1. Flight Route: https://www.kaggle.com/open-flights/flight-route-database
2. Airports: https://aviation-edge.com/aviation-database
3. Our shared zip file is shared via https://send.utoronto.ca/pickup.php with **Claim ID**: 7FwBA5AvyqJRMxby and **Claim Passcode**: Eh9eyv8zbqvoFFa2

Once our uploaded datasets are downloaded, please ensure that they are under the same directory as other project files. To be specific, all the data sets and files we uploaded on the Markus should be in the same folder. In our project folder, run the file called main.py. For specific instructions on running our main module (GUI), please refer to the introduced User Manual under the visualization part.

Changes

1. After discussing with the group members, we noticed that it is extremely hard to scrape the price for every flights. As an alternative solution, we proposed a rule of estimating the price for flight routes based on flight distance, whether the flight is cross-country, and some random multipliers.
2. Based on the feedback from the TAs, we added two new searching algorithms BFS and DFS to compare with the A* search algorithm.
3. Over the basis of knowing that solving the world tour requires a solid comprehension on TSP, still we have managed to apply two algorithms - one brute force and one dynamic programming, to solve the NP TSP problem.
4. From the discussions with our group members, we explicitly defined that the customization to the world trip by the client can either prioritize on price or distance.

Report and Discussion

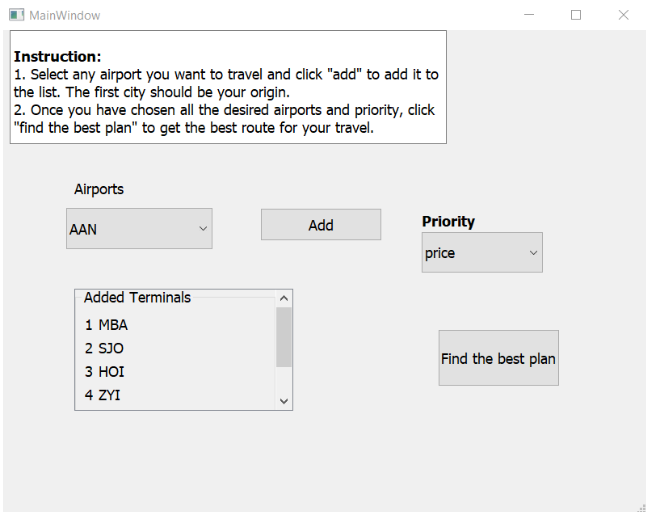
1. The central data structure for this project is Graph which represents a large set of network data. Without constructing Terminal, Route, and System classes, it would be impossible for us to develop various searching algorithms to complete the route planning service.
2. Application scenarios
   (a) Terminal-to-Terminal: User can choose specific departure and arrival terminals, the system will produce an optimized route for the user based on either price or distance.
   (b) Best City Pairing: Allows for user to input multiple departure cities and destination cities, which successfully outputs the best city pairing based on user's customized requirements to get to the ultimate destination.
   (c) World Tour: User can choose their departure and arrival terminals as well as the cities they want to travel. The system will produce an optimized route for the tour based on priority on price or distance.
3. In our project, the major computational algorithms are applied in the subsequent sub-problems. Since we used real world data set that includes tens of thousands of entries, it would be worthwhile to analysis the running time of each searching algorithm.
   (a) Searching for optimal route between two locations. All three algorithms DFS, BFS, and Astar have the same upper bound of time complexity, with $O(|V| + |E|)$, where $|V|$ is the total number of terminals and $|E|$ is the total number of routes. However, Astar is able to solve the problem with the highest efficiency since it knows which is the best route that can be taken from its current state and how it needs to reach the target airport. Given two arbitrarily chosen airports, the three models search for the routes with varying running times and the table below shows the performance of Terminal-to-Terminal Search.

Algorithms	Average Running Time (s)
DFS	11.414328145
BFS	2.6441678169
A*	0.0439383180

   (b) An optimal series of routes that starts and ends at same location, includes all specified cities exactly once. A brute force approach with A* aid is applied so that it evaluates every possible tour and return the best one while the dynamic programming method yields a much more efficient solution as all vital information is remembered and carried forward. For the DP method, there are at most $n \cdot 2^n$ sub-problems, with each one taking linear time to solve. Thus, the total running time is $O(n^2 \cdot 2^n)$. Over the basis of knowing the difference in time complexity, a simulation is conducted with different input size and the following results are obtained. Given a series of arbitrarily chosen airports, the table listed illustrates the performance of World Tour Search in terms of running time in seconds.

Number of Terminals	AStar-TSP (Brute Force)	DP-TSP
3	0.686384947	7.85869e-05
4	2.751648952	0.00013445
5	14.13248426	0.00037607
10	TLE	0.06464656
15	TLE	2.84143637
20	TLE	172.685251

4. For the front-end part, the GUI essentially makes our algorithms approachable for any plain users. With the implemented GUI, users can easily input the cities/airports they want to travel between and their heuristic preferences. After running the algorithms behind the scene, the best plan would be visualized clearly and interactively. This matches perfectly with our project goal of establishing a flight route planning service for customized requirements, and it works in a pretty similar way as Google Flights and Expedia.

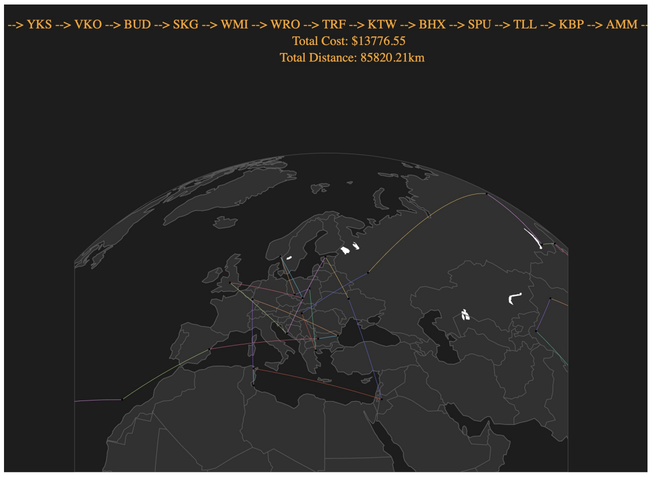


Instruction:
1. Select any airport you want to travel and click "add" to add it to the list. The first city should be your origin.
2. Once you have chosen all the desired airports and priority, click "find the best plan" to get the best route for your travel.

Airports: AAN

Priority: price

Added Terminals
1 MBA
2 SJO
3 HOI
4 ZYI



--> YKS --> VKO --> BUD --> SKG --> WMI --> WRO --> TRF --> KTW --> BHX --> SPU --> TLL --> KBP --> AMM --
Total Cost: $13776.55
Total Distance: 85820.21km

5. Limitations:

   (a) While BFS and A* are able to give the optimal customized flight routes between cities, the DFS method does not guarantee to have such optimal solution, instead, it only provides a viable routes between two cities.

   (b) Since the world tour is a NP hard problem, both of the algorithms that we implemented are not able to solve the problem in polynomial time, with one being $O(n!)$ and the other being $O(n^2 \cdot 2^n)$

   (c) Although DP-TSP is able to solve the World Tour problem much faster than the brute force one, still it requires the input terminals to form a Eulerian Graph with some routes between these terminals Note that a Eulerian graph is a connected sub-graph that has a closed trail (a walk with no repeated edges) containing all edges of the sub-graph. When the selected terminals don't match such requirement, there won't be a way to travel all of the cities once, without visiting other cities. As an alternative solution, the brute force algorithm will be applied, but only a limited amount of input cities are allowed to have a reasonable running time.

   (d) In the current simulation, sometime the system will output routes that are not user friendly. In other word, people might think the generated route are not reasonable. For example, when I prioritize on the price, sometimes the system will return a route that transfer between flights too many time just in order to decrease the price by a tiny margin.

6. Conclusions and Future Work

Overall, we have established a flight route planning service that provides optimal flight routes for customized requirements. In this project, we proposed various searching algorithms to develop a customized flight route planning service. Through testing, it is proved that the A* algorithm yields the best performance among the three constructed models for city-to-city search, with the least running time. In contrast, for world tour search, the dynamic programming algorithm outperforms brute force with A* aid in a much faster way. The simulation calls all of the searching methods with randomly generated input. Subsequently, it successfully outputs the result with an interactive graphical interface that demonstrates flight routes on a three-dimensional globe. In terms of future works, to improve the searching algorithms, it is worthwhile to design and implement approximation algorithms that can approach the World Tour problem with polynomial time, such as Annealing and Guided local Search, provided that they don't always produce an optimal solution. In addition to optimizing our searching algorithms, it would be more practical to add more functionalities to better serve the needs of the users, such as selecting multiple priorities when requesting optimal routes, showing more features of the arrival cities on the global map, such as landmarks and population, and possibly recommending accommodations near the arrival terminal.

References

[1] N. G. Uğur and A. Akbıyık, *“Impacts of COVID-19 on global tourism industry: A cross-regional comparison,” Tourism management perspectives, Oct-2020. [Online].* Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7474895/. [Accessed: 09-Mar-2021].

[2] “Rebuilding tourism for the future: COVID-19 policy responses and recovery,” OECD. [Online]. Available: http://www.oecd.org/coronavirus/policy-responses/rebuilding-tourism-for-the-future-covid-19-policy-responses-and-recovery-bced9859/. [Accessed: 09-Mar-2021].

[3] B. T. Group, “IATAs poll of recent travellers shows growing confidence in return to air travel,” Travelweek, 09-Mar-2021. [Online]. Available: https://www.travelweek.ca/news/iatas-poll-of-recent-travellers-shows-growing-confidence-in-return-to-air-travel/. [Accessed: 16-Mar-2021].

[4] A. Behsudi, “Tourism-dependent economies are among those harmed the most by the pandemic,” International Monetary Fund. [Online]. Available: https://www.imf.org/external/pubs/ft/fandd/2020/12/impact-of-the-pandemic-on-tourism-behsudi.htm. [Accessed: 09-Mar-2021].

[5] M. B. Pitrelli, “Many thought airfares would spike in the age of coronavirus. That’s not happening yet,” CNBC, 23-Sep-2020. [Online]. Available: https://www.cnbc.com/2020/09/23/airfares-airlines-arent-raising-prices-amid-covid-19-pandemic.html. [Accessed: 09-Mar-2021].

[6] J. Walton, “Will coronavirus make flying more expensive?,” BBC Worklife. [Online]. Available: https://www.bbc.com/worklife/article/20200528-will-the-price-of-flights-increase-due-to-coronavirus. [Accessed: 09-Mar-2021].

[7] API Reference. (2017). Retrieved March 15, 2021, from `https://pandas.pydata.org/pandas-docs/version/0.21.1/api.html`

[8] James, J. (Director). (2018, June 10). *Python: NUMPY — Numerical Python Arrays Tutorial* [Video file]. Retrieved March 15, 2021, from `https://www.youtube.com/watch?v=8Mpc9ukltVA`

[9] Hahmann, T. (n.d.). CSC384 Introduction to Artificial Intelligence, University of Toronto, Fall 2011, Torsten Hahmann. Retrieved March 15, 2021, from http://www.cs.toronto.edu/ torsten/csc384-f11/

[10] J. (2009, October 30). Tsp(Traveling Salesman Problem)–dynamic programming. Retrieved March 13, 2021, from https://blog.csdn.net/joekwok/article/details/4749713

[11] OpenFlights. (2017, August 29). Flight route database. Retrieved March 15, 2021, from https://www.kaggle.com/open-flights/flight-route-database