Amanda Norton
Julian Palazzo
Sumitra Sankarasubramanian
Benjamin Weisman
Professor McDonald
Algorithms for Data Science
December 1, 2020

Assignment 10: Flow Networks and Airlines

Data Transformation

For this assignment, we were given two important pieces of data- routes.dat and planes.dat. Each of these files were essential for determining the network we ended up building over the course of the project.

We started off by taking routes.dat and planes.dat and basically combining them into a single excel file. Within routes.dat we were interested in the Airline, the Source Airport ID, the Destination Airport ID, and Equipment. We weren't interested in flights with Stops greater than 0 or Codeshare because of the way we implemented midpoints. We weren't interested in any columns that were repetitive data such as Airline ID, Source Airport ID, or Destination Airport ID.

We had our network ready to be made from the provided files, however we were missing the weights - the max plane capacity. The max plane capacity is essential for answering problems one and two and is required to find any max flow. We searched through many different official sources but eventually settled on using Miles Tweed's plane capacities that they so generously shared with us. The file contains the max plane capacities along with the plane code that matches with our Equipment column.

Next our goal was to replace equipment codes with their respective max capacities. The first step in data processes was to get all distinct equipment types within the routes.dat file. This was done in RStudio by importing the data and separating the multiple entries of equipment into their respective columns. Once this was done, using R commands, we extracted the unique equipment types from each column and added them into a list. Next we took Mile's dataset and merged his capacities into this master list of equipment. This is what file "Data Processing File 1.Rmd" does.

The next step of data processing took place in Python using Pandas. The next goal is to replace the equipment values with the values from the data Miles provided which are the max capacity of the airplane depending on the equipment. We were able to make a dictionary, in which we paired the equipment values to the capacity values and then using the replace function in Pandas, we were able to replace equipment types with their values in the original routes.dat. Next bringing this csv back into RStudio, we were able to then separate once again the equipment column into multiple columns to then mutate these columns into a sum of total capacities for each route. In order to accomplish this, in the new replaced_routes.csv file, we changed the equipment columns into integers causing then equipment that did not have an assigned capacity into NA. Once adding all of the columns, the mutate function set all NAs into 0s. Therefore, routes with a total capacity of 0 meant that their equipment type did not match with anything in Miles' dataset and thus we did not have the data for that route. We made the decision to replace the 0s in total capacity into the average capacities from Miles' dataset which turned to be 160. Therefore, even if this route had multiple equipment types, its total capacity would be 160. So with the mutate function, and through this backward way of data processing, we accomplished our goal of having all routes and their associated max capacities.

After we created our singular csv, we imported the data as a Pandas Dataframe. Pandas Data Frames allow for much easier manipulation and made creating our functions much more simple. We subsetted the data frame to only include data where stops were equal to zero.

Algorithms

We used maximum flow within NetworkX. The maximum flow function in NetworkX implements the Preflow-Push algorithm to find the maximum flow in the network. The Preflow-Push algorithm determines maximum flow by allocating flow locally between each pair of edge-sharing nodes; in contrast, the Ford-Fulkerson algorithm determines maximum flow by allocating flow globally between each complete path connecting the source and sink nodes. This algorithm is more efficient for determining maximum flow than both Ford-Fulkerson and Edmonds-Karp, with asymptotic complexity of $O(V^2E)$.

Functions:

CapacityCheck

Capacity check takes source, destination and carrier as arguments and returns the total capacity of an edge between the source and destination, for a given carrier.

ArlineCheck

Airline check takes source and destination as arguments and returns a list of airlines with at least one route between the source and destination.

RouteCheck

Route check takes source and destination as arguments and returns the number of direct routes between source and destination.

routeCheckR

Route Check R takes source, destination, and dataframe as arguments and returns the number of direct routes between the source and destination. Dataframe is specified to allow subsetting of routes for any attribute value. In Part 2 we subsetted by airline to increase route-finding efficiency.

capacityCheckR

Like routeCheckR, capacityCheckR performs the same function as capacityCheck. The only difference is this function takes dataframe as an additional argument, allowing users to subset the routes dataframe to find routes' capacities more efficiently.

Filtering Routes

In the dataset we had 6 NewYork airports.
(https://en.wikipedia.org/wiki/Aviation_in_the_New_York_metropolitan_area)

We chose to use these airports as per the FAA and Wikipedia designating these airports as serving the New York Metropolitan area.

LGA	LaGuardia
JFK	John F Kennedy
ISP	Islip-Macarthur
SWF	Stewart
HPN	Westchester
EWR	Newark

3 San Francisco airports. (destination)

We chose these three airports based on the wikipedia page for airports servicing the San Francisco Bay Area, but left out the airport in Santa Rosa as it did not have direct transportation to San Francisco like the other airports.

SFO	San Francisco
SJC	San Jose
OAk	Oakland

Midpoints

List of all airports that have San Francisco as the destination airport.

Graph:

We used a Networkx Directed Graph.

Adding Edges:

Direct Edge - An edge was added for each airline with a direct route from any NewYork airport to any San Francisco airport.

Edges with Midpoint - An edge was added from NewYork to Midpoint and from Midpoint to San Francisco with the same carrier on both legs.

Implementation

We added edges to our graph in two steps. We began by adding all direct routes, if they exist, from all NY airports to all SF airports. To do this, we used routeCheck to determine if a direct route exists; if so, we added an edge between the nodes. Each edge has a source node from the ny_airports list and a target node from the sf_airports list, as well as a “capacity” and a “carrier” attribute. We used capacityCheck to assign a value for each edge’s “capacity” attribute.

After adding all direct edges to our graph, we began adding paths with one stop between any NY airport and SF airport.

Using our list of NY airports and midpoints (ie. airports with at least one route to any SF airport), we performed the routeCheck function on each source/destination pair to determine if at least one route exists. If there is a route between the NY and midpoint airports, we know that there is a path between the NY and SF airports, since all midpoints have routes to at least one of the SF airports. We assume that paths from NY → Midpoint → SF are only valid if the same carrier flies on both legs; therefore, to identify routes where the airlines match on both legs, we used the airlineCheck function to generate two lists: one of airlines that fly the first leg and one of airlines that fly the second leg. We then isolated airlines that appear in both lists (ie. have routes from NY → Midpoint and from Midpoint → San Francisco). We then iterated through this list of carriers, adding one edge from the NY airport to the midpoint airport and one edge from the midpoint airport to the SF airport. Again, we used the capacityCheck function to assign the appropriate capacity value to each edge's "capacity" attribute. We also added a "carrier" attribute to each edge.

Since our implementation treats all NY airports as potential source nodes and all SF airports as potential sink nodes, we found maximum flow from each NY airport to each SF airport. For each source/target pair, we appended the maximum flow capacity to a list called "allFlows". After appending the maximum flow value for each NY-to-SF pair, we simply added the flow values in the "allFlows" list to find the total maximum flow out of all New York airports to all San Francisco airports.

For part 2, we began by gathering a list of airports that appear in our graph from part 1; this was done in a similar fashion to how the edges were added in part 1, the only difference being instead of adding the edge to our graph, we appended the carrier to our list called "airline_list". After removing duplicates from our list of airlines, we iterated through the list and subsetted our original routes dataframe to only include routes for each airline in airline_list. In the same iteration, we initialize a graph and add edges for direct and one-stop routes using the same process as for our graph in part 1. Still within each-carrier iteration, we found the maximum flow between all nodes in that carrier's graph, assigning the maximum value to a new variable called "flowC". At the end of each iteration, we compared the "flowC" value to our value for the maximum flow of each carrier's directed graph; we call this variable "solution". If the "flowC" value is greater than our "solution" value, "solution" assumes the value of "flowC" After checking the maximum flow values for each carrier's graph and finding the maximum of these values, we simply identify the carrier whose graph had the greatest "flowC" value and the carrier which can transport the most passengers from New York to San Francisco.

How we validated our solution-

We validated our solution by manually calculating the maximum flow and comparing it with the result from our function.

Future Extensions

When working on this assignment, all of us had a realization that these algorithms are very similar to how airlines determine how to best distribute and use their resources. Airlines, and many other companies, employ similar algorithms to maximize profits. Essentially, companies use exploratory data analysis to avoid any wasted resources. By knowing maximum flow between different states, an airline could decide whether or not it needs more or less planes at a given state at a given time. As another example, a company could use maximum flow to decide whether a branch of stores in a specific state need more or less of x given resource thereby improving efficiency. Another idea we had was to create a python script in the future that runs our application automatically and feeds in real-time data.