Flights & Flow

Report for Algorithms Project #2 by Red Team

Red Team:

Danielle Banks Kevin Howlett Tim McCormack Kalani Stanton

Introduction

This project begins with a data source and a question: Using the data supplied by OpenFlights.org, how many people can be moved from one city to another? A simple enough question to answer with the use of networks and max-flow calculation algorithms, such as Ford Fulkerson or Edmonds Karp; however, this seemingly simple task was complicated by a lack of necessary data and the lack of normalization within the data we were given. We would need to solve these issues within our data prior to answering the questions, which require that we make a few application-driven decisions about our implementation of this flow network.

To guide these decisions, we took a step-back from the task at hand and began by considering the realistic applications for this project so that we could tailor our implementation to fit the needs of whomever would find this program useful. Ultimately, we designed this program to be a tool for logistical planning across any and all industry sectors that need to move large quantities of people from one location to another within a brief period of time. Some practical examples include transporting an entire branch of a company to a centralized location for a conference, or a mass relocation of employees due to an acquisition or merger. With this in mind, the following report discusses the organization of data, implementation of algorithms, and all decisions made therein with the intent of maximizing its usability and relevance in practical applications.

Acquiring and Organizing Data

As was alluded to in the introduction, not all of the necessary data were provided at the onset of this project. The flight routes and airplane data, supplied by OpenFlights.org were enough to build out a directed-unweighted network, with vertices and edges supplied within the *routes* data; however, in order to build a *flow-network,* we would need to associate *capacities* with these edges. Unfortunately, during this process of associating the capacities with edges, we discovered some redundancies in the “Equipment” column of the *routes* data frame that we needed to address before the capacities could be assigned. The section that follows contains a discussion of how we collected our seating capacity data, our method for assigning edge capacities, as well as how we addressed the issue of redundancies in the data with a practical, application-oriented solution.

Data Acquisition

OpenFlights

The primary data source for this project was OpenFlights.org, from which we pulled three data sets that contain data on routes, planes, and airports respectively. Two of these data

sets are directly relevant to the task of constructing a flow-network, despite one of them not being very useful to that end, while the other is applied in this project to enhance the usability of the end-product.

routes.dat.txt

Contained within this data set are the values needed to construct the network. The Source ID and Destination ID columns contain the identifiers for the nodes, while the rows that contain pairs of these values that represent the edges of the network. Furthermore, this data set contains a number of other columns that can be used to identify the nodes, such as IATA codes for airports, and eventually associate capacities with each edge via the "Equipment" column, which contains codes that represent the aircraft that routinely fly the route indicated by their corresponding row.

	Airline Code	AirlineID	Source	SourceID	Destination	DestinationID	cShare	NumStops	Equipment
0	2B	410	AER	2965	KZN	2990	NaN	0	CR2
1	2B	410	ASF	2966	KZN	2990	NaN	0	CR2
2	2B	410	ASF	2966	MRV	2962	NaN	0	CR2
3	2B	410	CEK	2968	KZN	2990	NaN	0	CR2
4	2B	410	CEK	2968	OVB	4078	NaN	0	CR2

planes.dat.txt

This data set was not very useful, as it contained little more than a nominal reference for the "Equipment" column in the routes data; however, cross-referencing the names of the aircraft with the codes used to identify them in the *routes* data would have offered us a way to manually impute the capacity values on each edge. Ultimately, we used this method in conjunction with another, similar data set that contained the capacities in addition to the nominal data present in this data set.

	Airplane	IATA	ICAO
0	Aerospatiale (Nord) 262	ND2	N262
1	Aerospatiale (Sud Aviation) Se.210 Caravelle	CRV	S210
2	Aerospatiale SN.601 Corvette	NDC	S601
3	Aerospatiale/Alenia ATR 42-300	AT4	AT43
4	Aerospatiale/Alenia ATR 42-500	AT5	AT45

Airports.dat.txt

As mentioned previously, this data set was used to enhance the experience of interfacing with our algorithm. The nominal and geographic data in this file was used to collect the OpenFlights ID values according to the cities where these airports were located. From the association between the OpenFlights ID values and their respective cities, we constructed a dictionary that allows a user to pick a City and then passes the ID value into the algorithm for constructing the flow-network. This means that the flow-network is generated according to the specifications of the user without them having to look up the OpenFlight ID for every airport they are interested in using as a source or sink node.

0	1	2	3	4	5	6	7	8	9	10	11	12	13
1	Goroka Airport	Goroka	Papua New Guinea	GKA	AYGA	-6.081690	145.391998	5282	10	U	Pacific/Port_Moresby	airport	OurAirports
2	Madang Airport	Madang	Papua New Guinea	MAG	AYMD	-5.207080	145.789001	20	10	U	Pacific/Port_Moresby	airport	OurAirports
3	Mount Hagen Kagamuga Airport	Mount Hagen	Papua New Guinea	HGU	AYMH	-5.826790	144.296005	5388	10	U	Pacific/Port_Moresby	airport	OurAirports
4	Nadzab Airport	Nadzab	Papua New Guinea	LAE	AYNZ	-6.569803	146.725977	239	10	U	Pacific/Port_Moresby	airport	OurAirports
5	Port Moresby Jacksons International Airport	Port Moresby	Papua New Guinea	POM	AYPY	-9.443380	147.220001	146	10	U	Pacific/Port_Moresby	airport	OurAirports

Finding Capacities

The lack of data on seating capacities in the files from OpenFlights meant that we needed to decide on a method for collecting these data from additional sources. To do so would require that we either find a file that contained these data, or manually impute the data using the values in the *planes* data set. Although we were able to find a data set that contained seating capacities, not all of the values in the "Equipment" column were present; thus, we were forced to manually impute these missing values with what could be found using the IATA codes to search for the information online[1]. Once these capacities were found, we joined them to the *routes* data on the "Equipment" column. Unfortunately, the first time we attempted this we encountered issues due to the fact that the "Equipment" column contained observations of different lengths.

	Airplane	ICAO	Equipment	Capacity	Country
0	Aerospatiale/Alenia ATR 42-300 / 320	AT43	AT4	50	France
1	Aerospatiale/Alenia ATR 42-500	AT45	AT5	50	France
2	Aerospatiale/Alenia ATR 42/ ATR 72	\N	ATR	74	France
3	Aerospatiale/Alenia ATR 72	AT72	AT7	74	France
4	Aerospatiale/BAC Concorde	CONC	SSC	128	France

[1] Credit to Sara Haman for collecting and sharing a dictionary of these missing values.

Data Organization

Overall, the datasets we used were organized in a coherent logical manner, but required some minor cleaning and restructuring in order to fit the demands of this project. Although, the format of the *routes* data was conducive to the construction of a network, given that each row pertained to a single edge between a source and destination pair, the redundancy of having multiple values in the "Equipment" column required that we adjust how we chose to incorporate *capacities* into the data. Additionally, we wanted the algorithm to be user friendly so we decided to associate the airport identifier codes with their respective cities. Doing so allows the user to input the name of a city without having to reference any documentation to identify the numeric codes associated with their airports of interest.

Removing NA values

Within the data sets, there were multiple columns that contained na values (retained as "\\N" or "nAn"), which needed to be addressed based on where they were located. For NA values in the "Equipment" column, we chose to remove the rows altogether because imputing any value based on assumptions would likely lead to erroneous calculations in the long run and, in regards to the user-focused approach we took with this project, accuracy and precision are of utmost importance as this program will inform subsequent logistical decisions made by the user. We also removed any rows that contained "\\N" in the 'Source' or 'Destination' columns, since these are the columns that we used as identifiers for our vertices and passing null values into the algorithm for building a network would return an error. The removal of NA values resulted in a loss of only 18 entries from our original data set of 67,663 rows.

Multiple Values in "Equipment"

Under the assumption that there were no redundancies, our original plan was to simply join the capacities with the edges in the *routes* data on the "Equipment" column; however, upon discovering multiple values in this column for certain rows we needed to adjust our approach. Now, instead of only having one capacity value for each row, we chose to create three new columns containing the minimum, maximum, and average of the capacities associated with the values in the "Equipment" column. In rows where there is only one "Equipment" value, the capacities are the same across all columns; in rows with multiple values, the value selected can have a significant impact on the max-flow value derived from the network.

This solution has a practical advantage over reducing the capacities to just a single value because it allows the user to select the value they wish to use in computing max-flow. We chose to implement this freedom of choice because of the many conditions under which someone may use this program. For example, if it is of utmost importance that all people arrive within the same time-frame, using the "Minimum Capacity" column would give the user greater certainty about the number of people that can be flown from source to target. In contrast, the "Maximum Capacity" value can be used in circumstances where the timing is less important; i.e. the plane used on that route may change every day, and if there are not enough seats on the plane flying one day, they can simply schedule all the flights on different days.

Lastly, the “Average Capacity” gives a probabilistic calculation of how many people can be flown from one city to another on any given day, under the assumption that each plane is equally likely to fly that route on a selected day. In giving control to the user, we effectively increase the precision with which the user can make plans using the max-flow values derived from our program.

Locating Airports

In an attempt to increase the usability of our program, we chose to simplify the process of selecting the source and destination city by generating a dictionary wherein the keys are cities and the values are the airport id codes used in the “Source” and “Destination” columns of the *routes* data. Prior to generating the dictionaries used in the final product, we joined these data on the “Source” and “Destination” columns that contained the IATA codes for the airports. Then, once the data were joined, we extracted the unique city values and iteratively constructed a dictionary for both the source cities and destination cities for the airports listed in the *routes* data.

Implementation Preparation

The final step in preparing our data sets for the network is generating a usable edge list that can be passed into the network construction algorithm. To do this, a simple grouping function was called on our data, resulting in a grouped dataframe, which was indexed according to “Source” and contained “Destination”, “Minimum Capacity”, “Maximum Capacity” and “Average Capacity” for each destination from this indexed location.

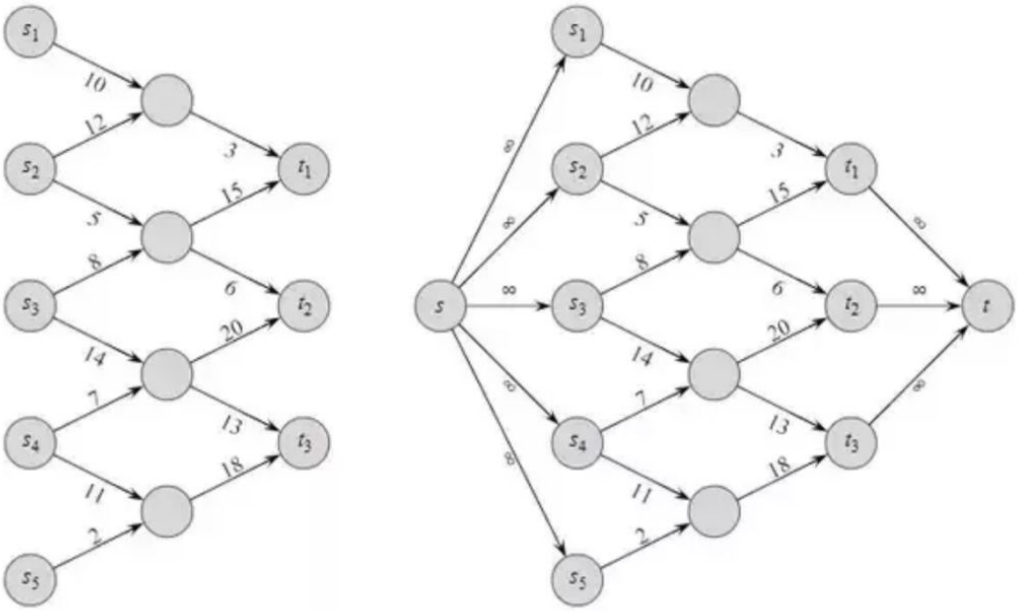
Constructing The Network

Network Components

Fundamental Structure

The network was built with a dataframe derived from *routes.dat.txt*. Each record in the file represents a single flight with a source and a destination. We used data frames from the pandas in Python 3 to store and subset the information from *routes.dat.txt*. This decision served us well because it allowed us to shape and subset information seamlessly. There were multiple constraints that were required of the network data frame. The first constraint was that the number of stops in a flight could not be greater than zero, or that the source had to be New York City and the destination had to be San Francisco. The reason for this subset is that we are allowed one and only one layover in the trip from New York City to San Francisco, in the case that a flight has one or more stops, it does not satisfy our initial condition. If there does exist a flight with one stop, then that flight may only be included if its source is NYC or its destination is SFO.

Super-Nodes



Super-Nodes and Super-Sinks take into account the fact that multiple airports exist in San Francisco and New York City. The specification sheet describes the goal as transporting passengers from New York City to San Francisco, without specifying that a certain airport is to be used. With the assumption that all airports in a city are equally desirable the max-flow of the network can be increased if there are multiple source nodes, and multiple sink nodes. Max-flow is calculated by finding new augmenting paths, given that more edges are added to the network, there will be more opportunities for augmentation.

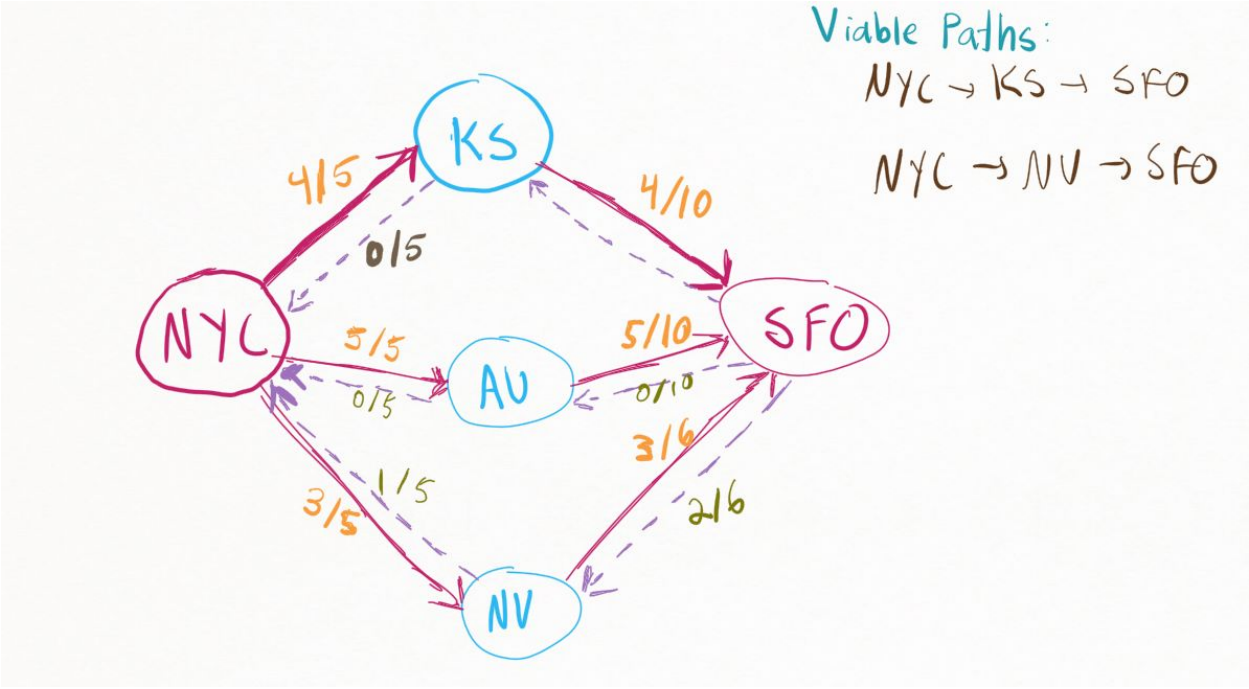
Network Construction

The network exists both as an edge list and as a capacity matrix. The edge list uses the unique identifiers for source and destination as the starting and ending points respectively, and it uses the capacity as a weight for the given edge. The edge list is created by iterating through the subset data frame described in the paragraph above.As an aside, our program also keeps track of which airports have been visited. Once the edge list has been created, the capacity matrix construction may begin. The capacity matrix is a square matrix, where the length of one row is equal to the number of visited airports. In the case of New York City to San Francisco, there are 259 visited airports. In such cases the capacity matrix dimensions are 259 x 259. A dictionary is also created that maps the column number to the unique identifier of an airport.

Deriving Answers

Computing Max-Flow

In our simulation the max-flow value represents the maximum number of people that can be transported from New York City to San Francisco. The algorithm we used to calculate the max-flow of our network was Edmonds-Karp. This algorithm is a specific form of Ford-Fulkerson that utilizes Breadth-First Search when searching for potential augmenting paths. Breadth-First Search traversals of graphs can be simplified by the notion that parent nodes are visited before child nodes. Augmenting Paths can be defined as paths that contain all non-forward edges or all non-empty backward edges. Another characteristic of these paths is that the amount of flow into a node must be equivalent to the amount of flow out of a node. The subgraph illustration below describes which paths would be considered viable or augmentable given the current residual. A residual graph is a skeleton of the initial graph that updates the flow-capacity ratio after each iteration of the Edmonds-Karp algorithm.



The Edmonds-Karp Algorithm begins by setting the flow rate of all edges equal to zero. The capacities of edges remain constant throughout our simulation. A breadth-first search traversal is performed to see if there exists a potential path from the source node to the sink node. If no path is found, then the algorithm will exit. In the case that a path is found, the minimum flow rate is calculated. The min-flow rate is also known as the bottleneck capacity, it represents

the maximum number of people that can travel that particular path and successfully end at the sink node. The bottleneck capacity is added to the max flow rate and then the algorithm iterates once more. Our Edmonds-Karp algorithm was implemented using a capacity matrix described above. This is classic across computer applications, where a graph is represented via a square matrix. Each row-number represents a unique airport.

Interfacing With the Algorithm

The most important function of our application is the ability to calculate the max flow rate of any two cities. Each input is used as a parameter that corresponds to a source and sink of the graph. From this information the *routes.dat.txt* file is subsetted to include only flights that travel from source to sink with at most one layover. The edge list as well as the capacity matrix can then be constructed via the subsetted data frame. Our Edmonds-Karp algorithm operates with the capacity matrix. The head of a sample capacity matrix is available below. With the robustness of our implementation of calculating the max-flow rate we were able to use it to find the optimal airline. To solve the optimal airline problem we subsetted the *routes.dat.txt* file like before, however this time we partitioned the file based on unique airlines. Each airline had its own capacity matrix, that is fed into our max flow function. This process is illustrated below.

	0	1	2	3	4
0	0.0	0.0	0.0	0.0	0.0
1	1934.0	0.0	248.0	403.0	670.0
2	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0

To produce our solutions to the problems, we created two functions to take in a routes data frame and some set of arguments, and output a number which represents the estimated number of people that can be transported. The first function, *maxFlow*, takes in the routes dataframe, an airlineID (default to *None*) a list of starting airports, a list of ending airports, and a capacity estimate method as arguments. It then performs data cleaning, graph creation, and calculates and outputs a max flow value, using Edmonds-Karp. The following figure is an example, which shows how many people can be transported from the New York Metropolitan Area to the San Francisco Bay Area using all airlines, the biggest plane available for each route, and all the airports in the immediate area. The max number of passengers that can be transferred is 48875.

```
sources=['JFK','LGA','EWR'] # Major airports in NY Metropolitan area
sinks=['SFO','OAK','STS','SJC'] # Major airports in SF Bay area

maxFlow(routesDF, sources=sources, sinks=sinks, capEstimate='MaxCapacity')
# 48875 people using major airports and largest available plane for each route
```

```
48875.0
```

We then produced estimates from this function, subsetting for every airline carrier. The top sorted results are shown below. The carrier most effective at transporting passengers from NYC Metropolitan Area to SF Bay Area is United Airlines, with a high estimate of 10250.

	AirlineCode	AirlineID	MaxPeopleLowEst	MaxPeopleAvgEst	MaxPeopleHighEst
464	UA	5209	8137	9251	10250
153	DL	2009	2939	3371	3870
89	AA	24	3062	3226	3433
475	US	5265	2460	2601	2779
110	B6	3029	1492	1518	1544
292	LH	3320	966	1109	1232
507	WN	4547	1134	1152	1170
278	KL	3090	606	691	847

We also created a function called *maxFlowCity* which can take in almost all the same arguments as *maxFlow*, with the key difference of taking a city name for the *startCity* and city name for *endCity* arguments, instead of airport codes. Using this method, we obtained a smaller estimate (36805 passengers), since there was only one airport in SF city, and two in NY city.

```
maxFlowCity(routesDF, startCity='New York', endCity='San Francisco', capEstimate='MaxCapacity')
```

```
36805.0
```

The following figure shows the estimates for each airline carrier, using *maxFlowCity*, going from ‘New York’ to ‘San Francisco’.

	AirlineCode	AirlineID	MaxPeopleLowEst	MaxPeopleAvgEst	MaxPeopleHighEst
464	UA	5209	3379	3536	3674
153	DL	2009	2257	2810	3415
89	AA	24	2615	2776	2956
475	US	5265	2250	2351	2466
110	B6	3029	1394	1420	1446
292	LH	3320	751	821	890
95	AF	137	470	596	824
278	KL	3090	516	638	772
102	AS	439	604	661	717

Validation

As a means of validating our results we decided to construct a network using the python library networkx. This particular library is extremely powerful with regards to the implementation of various shortest path and max flow rate algorithms. You can simply construct a network via an edge list, and perform operations on the network. We were aware of the power of this tool, and were careful to use it as a means of data validation, and not as a tool for development. The same subsetting process of removing multiple layover and irrelevant flights took place before the construction of the networkx network. We utilized the `addEdge` functionality provided by networkx to iterate through the subsetted dataframe, adding sources, destinations, and capacities. One simple call to a function then confirmed our earlier findings. This value, 31432, matched the max flow we calculated using our custom graph class/algorithm with ‘avgCapacity’ as the capacity estimate method and JFK and SFO as our start and end airports, respectively.

```
nx.maximum_flow_value(G, "3797", "3469") # 31432 people can be transported w/ only 1 layover

31432.0
```

To validate our custom graph class and Edmonds-Karp algorithm, we also produced a small sample graph to which we knew the max flow answer. We calculated max flow using our algorithm and obtained the correct solution.

```
graph = [[0, 16, 13, 0, 0, 0],
        [0, 0, 10, 12, 0, 0],
        [0, 4, 0, 0, 14, 0],
        [0, 0, 9, 0, 0, 20],
        [0, 0, 0, 7, 0, 4],
        [0, 0, 0, 0, 0, 0]]

g = Graph(graph)

source = 0; sink = 5

print ("The maximum possible flow is %d " % g.FordFulkerson(source, sink))
```

```
The maximum possible flow is 23
```

Future Directions

Airline Efficiency

The Airline industry is a multibillion-dollar industry with people in charge of every aspect of the industry from logistics to maintenance to flight and crew allocation. In this project, we explored the maximum capacity for each airline given and starting location to a target location. When exploring the project a little further we could implement Ford- Fulkerson algorithms to explore the maximum capacity of each airline per hour. In doing so, we can maximize the number of people that can travel from one destination to the other during certain times. This information can be helpful for airlines to make helpful business decisions. Companies can then use the maximum capacity per hour values to determine flight pricing for certain times. The more people that can be transported during a specific time frame may be more expensive and the fewer people traveling airfare may be more expensive.

Examining maximum capacity per hour poses the problem of resource allocation. How can airlines distribute time, fuel, and human capital efficiently? Airline unions only permit pilots to work no more than 9 hours per day and no more than 14 hours for flight attendants. On average, planes need servicing after 125 hours and require over 20hours of manpower, and require fuel in between long flights. Fortunately, when supplied with the plane capacity, fuel tank capacity, coupled with the rest of the airline data we can use Ford-Fulkerson to determine the maximum fuel capacity in gallons for a given flight.

A project extension could be used to maximize airlines profits and maximize airline resources and ultimately saving companies bottom lines each year.