MAX FLOW COAST TO COAST

A practical implementation of min-cut/max-flow on airline data from Openflights.com

Sara Haman, Conor Howlett, and Adam Lashley

TABLE OF CONTENTS

THREE BIG QUESTIONS

CAPACITY

How do we determine what the “maximum” number of passengers that can be transported is? What does “maximum” mean?

GRAPH

How do we transform the raw data structure into the a graph object which is optimal for our project?

MAX FLOW

What is the maximum number of passengers that can be transported from New York to San Francisco given trips with no more than one layover?

DESIGN DECISIONS

We decided to go with a two-sided approach in analyzing flight capacity:

MAX FLIGHTS

For each route (source airport to destination airport) we selected **the flight which could seat the maximum** amount of passengers

MASS EGRESS

Imagine is the apocalypse and we want to move as many people from NY to California as possible all at once. To answer this, we calculated the **sum of the capacities for all planes** that could possibly fly a route.

DATA CLEANING

In order to aggregate the data by the max and the sum capacities, we first had to manipulate it into an edge list.

Data structure:
Pandas data frames

Cleaning progression:

1. Normalize the table
2. Select the columns of interest
3. Layover handling
4. Merge the routes dataframe with plane capacities
5. Handle missing values
6. Convert to an edge list

Routes - Early Data

	Airline	AirlineID	Source	SourceID	Destination	DestinationID	Codeshare	Stops	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
5	2B	410	DME	4029	KZN	2990	NaN	0	CR2
6	2B	410	DME	4029	NBC	6969	NaN	0	CR2
7	2B	410	DME	4029	TGK	NaN	NaN	0	CR2
8	2B	410	DME	4029	UUA	6160	NaN	0	CR2
9	2B	410	EGO	6156	KGD	2952	NaN	0	CR2

Attributes needed for the edgelist

Attributes needed to find the capacities

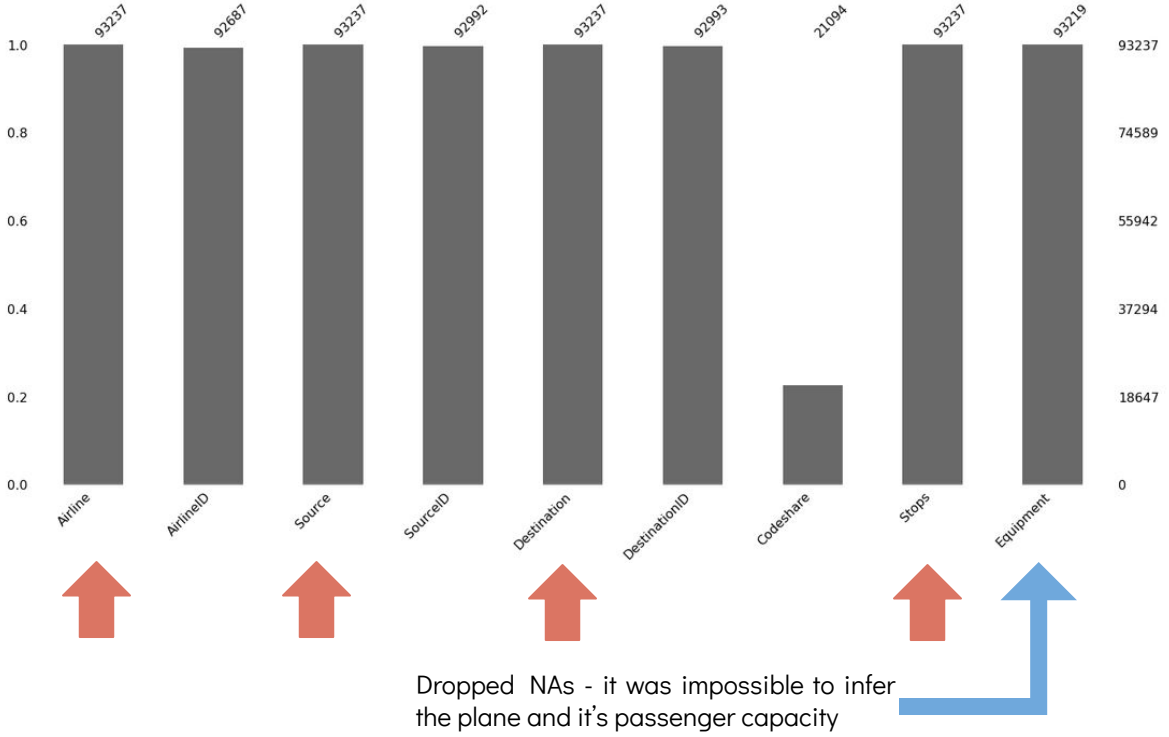
DATA CLEANING

In order to aggregate the data by the max and the sum capacities, we first had to manipulate it into an edge list.

Data structure:
Pandas data frames

Cleaning progression:

1. Normalize the table
2. Select the columns of interest
3. Layover handling
4. Merge the routes dataframe with plane capacities
5. Handle missing values
6. Convert to an edge list



CALCULATING CAPACITIES

	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	NaN	ATR	74	France
3	Aerospatiale/Alenia ATR 72	AT72	AT7	74	France
4	Aerospatiale/BAC Concorde	CONC	SSC	128	France
5	Airbus A300 pax	A30B	AB3	200	European consortium
6	Airbus A300-600 pax	A306	AB6	266	European consortium
7	Airbus A310 Freighter	A310	31F	NaN	European consortium
8	Airbus A310 all pax models	A310	310	198	European consortium
9	Airbus A310-200 Freighter	A310	31X	NaN	European consortium

Step 1. Found a supplemental data set containing ICAO codes and capacities. However, ~50 planes were still missing.

Step 2. Manually found plane capacities. Cross referenced two comprehensive online ICAO catalogues to find the planes.

CALCULATING CAPACITIES

```
## Plan:
#### Step 1. Find the seating capacity of each plane
#### Step 2. Create a dictionary of the planes and their capacities
#### Step 3. Use fillna to impute data where na's are, this is speedy on big datasets

# STEP 2.
dict = {'A81' : 85,  'AN4' : 44, 'BNI' : 9, 'CNC' : 19, 'DHP' : 6, 'DHT' : 20, 'BET' : 1,  '73M' :161.5,
        'SU9' : 1, 'PL2' :9, 'PAG' :9, 'YK2' :120, 'YK4' :24,  'PA2' : 9, 'CN1' : 1, 'CNA' : 1, 'BE1'  :19, 'CNT' : 14, 'MA6' : 1,
        '77W' : 396, '772' : 368,  'CRK' : 100, '787' : 310, '32B' : 244,  '32A' : 150,  '73C' : 1, 'CRA' : 1,
        '77L' : 317, '788' : 242,  '76W'  : 1, '74Y' : 1, '74H' :467, '73J' : 1, '73Q' : 1, '75W' : 1, 'F28' : 65,
        '74N' : 1, 'YN7' : 52, 'IL9' : 262, 'A58' : 75,  '75T' : 1,  'BH2' : 14, 'NDE' : 5, 'BEC' : 2, 'CNJ' : 10,
        'J32' : 19, 'AB4' : 247, 'PA1' : 9,  'BE9' : 16, 'M1F' : 1, 'YN2' : 17, '76F' : 1,  'CN2' : 19, 'SFB' : 1,
        '73R' : 150, '73N' : 149,  '77X' : 1, '33X' : 1,  '32C': 107}

# STEP 3.
full_routes.Capacity = full_routes.Capacity.fillna(full_routes.Equipment.map(dict))

# SEEING IF IT WORKED
print(full_routes['Capacity'].isnull().sum())

# IT DID!
# The NA count for capacity is now 0
```

Missing planes were primarily personal aircraft, Russian planes made by the company Yakolev, and helicopters.

THE GRAPH

The edge list was transformed into a matrix of dictionaries with identifier numbers, which allowed attributes to be retained. The graph was instantiated as a class object. The network is **directed** moving left to right with no cycles.

```
def createDict(el):
    '''

    Creates a dictionary with these constraints:

        key: the airport id
        value: unique int to represent the airport

    This is for use in an adjacency matrix/graph

    '''
    airports = []

    for i in el:
        for j in i:
            if (type(j) == str) and j not in airports:
                airports.append(j)

    numbers = list(range(len(airports)))
    airportDict = dict(zip(airports, numbers))

    return airportDict
```

THE GRAPH

The edge list was transformed into a matrix of dictionaries with identifier numbers, which allowed attributes to be retained. The graph was instantiated as a class object. The network is **directed** moving left to right with no cycles.

```
def newEL(el):
    '''
    Duplicates the airport edgelist, but uses the new integer identifiers

    '''
    new_el = []

    for i in el:
        edge = []
        for j in i:
            if type(j) == str:
                edge.append(airportDict[j])
            else:
                edge.append(j)
        new_el.append(edge)

    return new_el
```

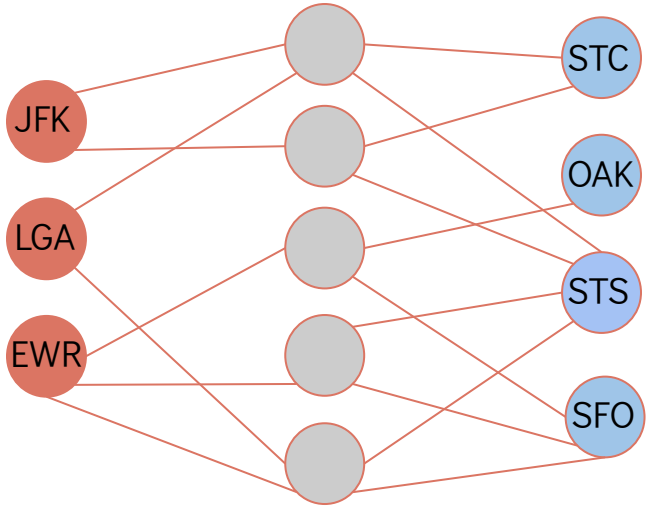
THE GRAPH

The edge list was transformed into a matrix of dictionaries with identifier numbers, which allowed attributes to be retained. The graph was instantiated as a class object. The network is **directed** moving left to right with no cycles.

```
def display(A):
    '''

    Creates a zeros matrix, size (number of nodes) by (number of nodes)

    Appends the nodes to their locations with their respective capacity values

    Prints the solution for most people that can travel from New York to San Francisco

    Else, returns the max flow using Ford Fulkerson if solving for the best carrier

    '''

    A = np.zeros((len(airportDict),len(airportDict)), dtype =float)

    for i in newEL(el):
        A[i[0]][i[1]] = i[2]

    g = Graph(A)

    if (aggType == 'Max') and (carriers == False):
        print("The maximum passengers that can fly from New York to San Francisco utilizing a single plane is: %d " %
              g.FordFulkerson(airportDict['Super Source'], airportDict['Super Sink']))
        print()

    if (aggType == 'Sum') and (carriers == False):
        print("The maximum passengers that can fly from New York to San Francisco utilizing every availible plane is: %d " %
              g.FordFulkerson(airportDict['Super Source'], airportDict['Super Sink']))
        print()

    else:
        return g.FordFulkerson(airportDict['Super Source'], airportDict['Super Sink'])
```

THE GRAPH

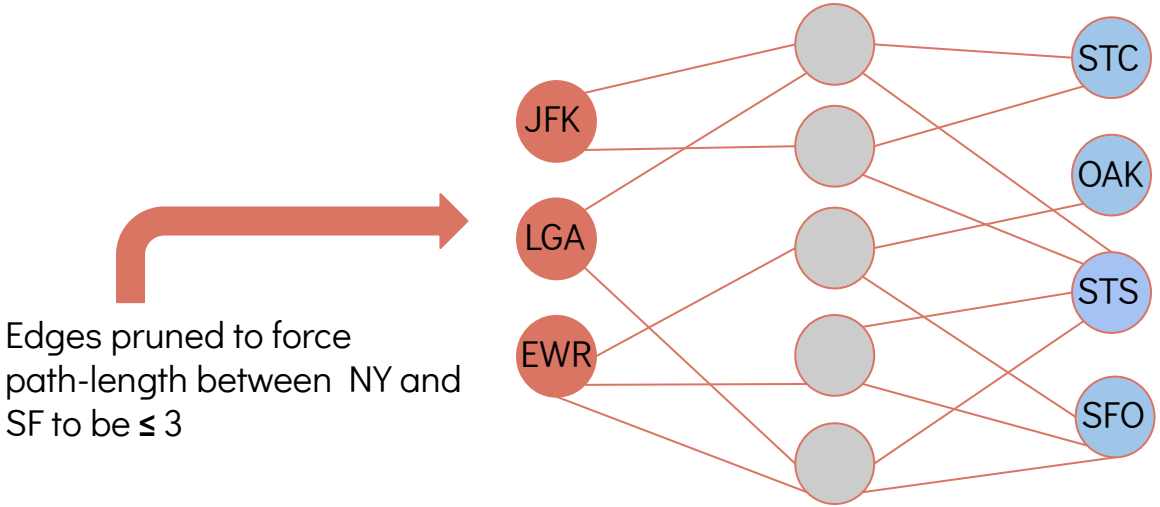
The edge list was transformed into a matrix of dictionaries with identifier numbers, which allowed attributes to be retained. The graph was instantiated as a class object. The network is **directed** moving left to right with no cycles.



However, having multiple sources and multiple sinks created a resource allocation problem.

THE GRAPH

The edge list was transformed into a matrix of dictionaries with identifier numbers, which allowed attributes to be retained. The graph was instantiated as a class object. The network is **directed** moving left to right with no cycles.

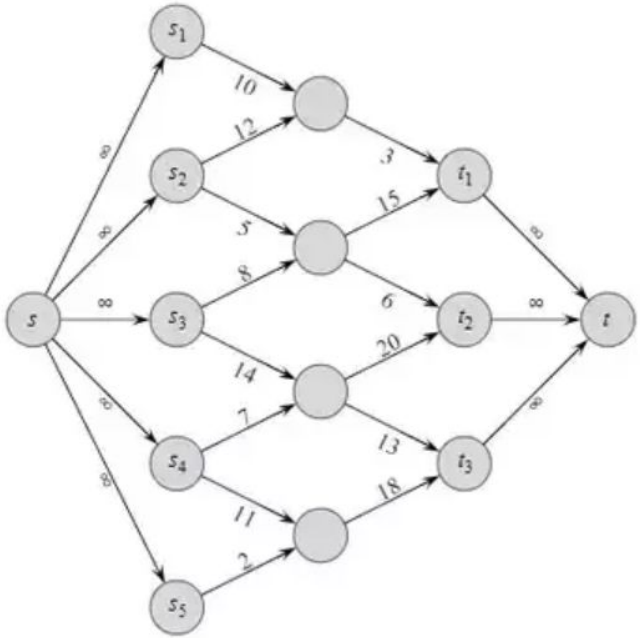


Edges pruned to force path-length between NY and SF to be ≤ 3

However, having multiple sources and multiple sinks created a resource allocation problem.

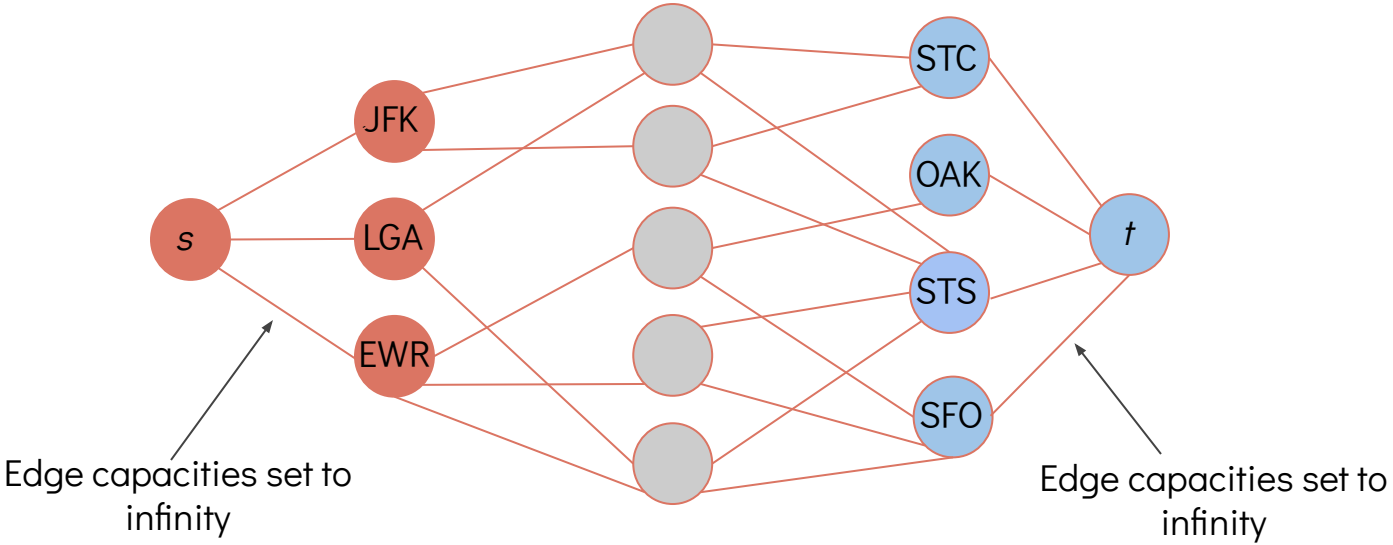
‘SUPER’ VERTICES

Created in a function which allows the user to choose which airports they want to travel to and from.



‘SUPER’ VERTICES

Created in a function which allows the user to choose which airports they want to travel to and from.



ALGORITHMS

```
def BFS(self,s, t, parent):
    '''
    Implements breadth first search
    '''
    visited =[False]*(self.ROW)

    queue=[]

    queue.append(s)
    visited[s] = True

    while queue:

        u = queue.pop(0)

        for ind, val in enumerate(self.graph[u]):
            if visited[ind] == False and val > 0 :
                queue.append(ind)
                visited[ind] = True
                parent[ind] = u

    return True if visited[t] else False
```

Time complexity: O(V+E)

MAX FLOW PROBLEM

Goal: to route as much flow as possible from *s* to *t*.

BREADTH FIRST SEARCH

Used to prune the network to only relevant flights.

FORD FULKERSON

Edmonds-Karp algorithm implementation for finding augmenting paths

ALGORITHMS

```
def FordFulkerson(self, source, sink):
    '''
    Implements Ford Fulkerson on a graph, taking the given source and sink,

    Returns the maximum flow for the given source to sink pair

    '''
    parent = [-1]*(self.ROW)

    max_flow = 0

    while self.BFS(source, sink, parent) :

        path_flow = float("Inf")
        s = sink
        while(s !=  source):
            path_flow = min (path_flow, self.graph[parent[s]][s])
            s = parent[s]

        max_flow +=  path_flow

        v = sink
        while(v !=  source):
            u = parent[v]
            self.graph[u][v] -= path_flow
            self.graph[v][u] += path_flow
            v = parent[v]

    return max_flow
```

Time complexity: $O(|V||E|^2)$

MAX FLOW PROBLEM

Goal: to route as much flow as possible from *s* to *t*.

BREADTH FIRST SEARCH

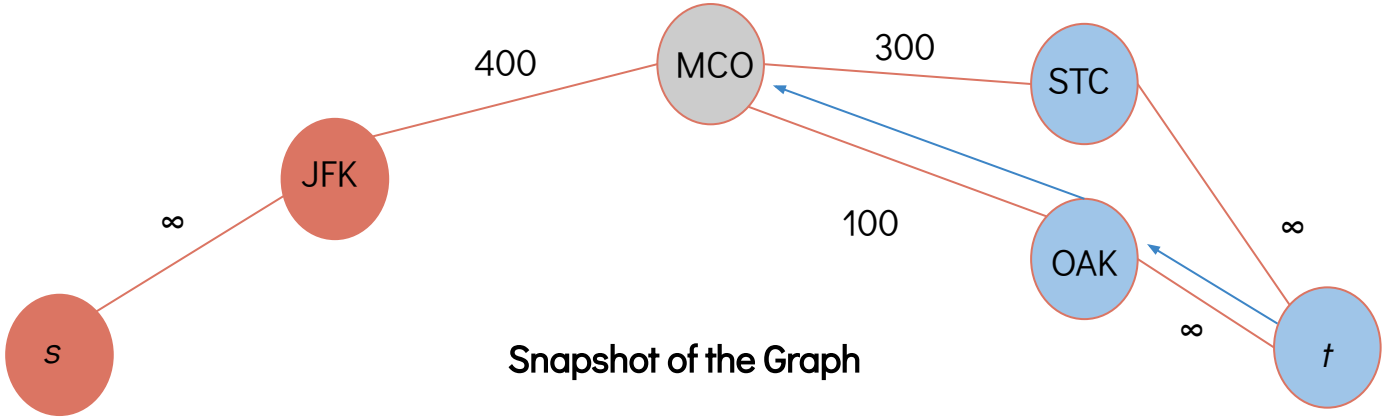
Used to find the augmenting paths.

FORD FULKERSON

Edmonds-Karp algorithm implementation for finding augmenting paths

The maximal number of people that can be moved from NY to San Francisco?

SINGLE FLIGHT



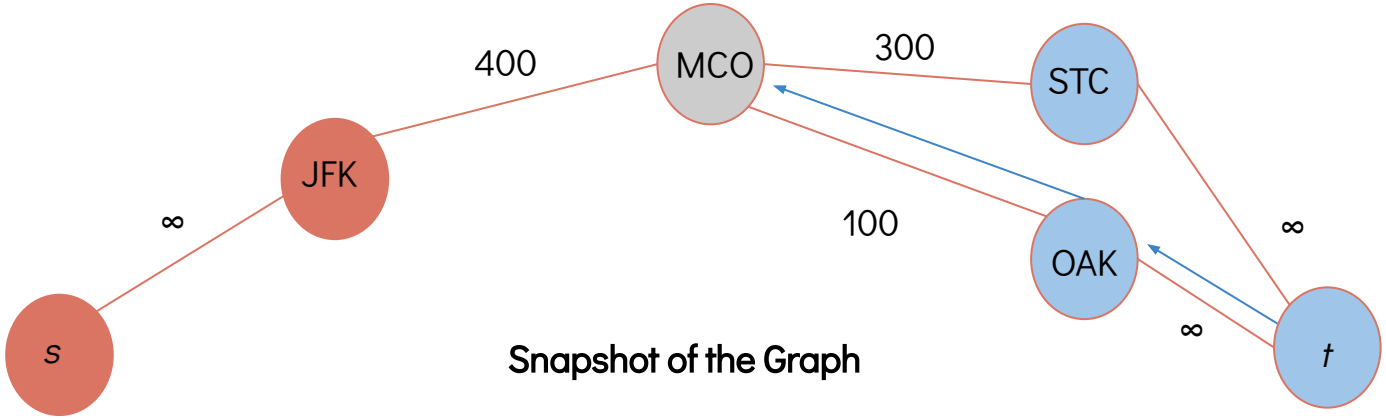
Snapshot of the Graph

```
The maximum passengers that can fly from New York to San Francisco utilizing a single plane is: 18901
```

Average runtime: 3.79 s

The maximal number of people that can be moved from NY to San Francisco?

MASS EGRESS



Snapshot of the Graph

```
The maximum passengers that can fly from New York to San Francisco utilizing every availible plane is: 66188
```

Average runtime: 3.01 s

Which *carrier* can transport the greatest number of individuals from NY to San Francisco?

BOTH

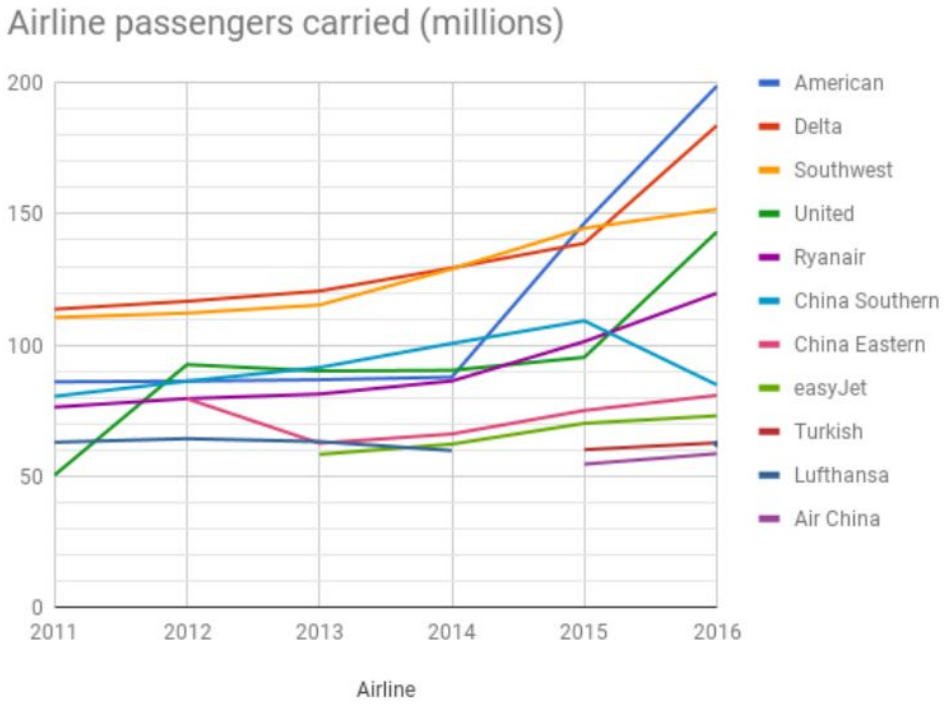
WINNER: United Airlines

```
The airline that can transport the most passengers using their largest available planes is: UA
The total passengers that can be transported is: 9732
The airline that can transport the most passengers using all available planes is: UA
The total passengers that can be transported is: 18845
```

Average runtimes: 7.45 s and 7.48 s

DISCUSSION

- The accuracy of the max flow algorithm was validated by taking subsamples of the graph and running it on them and comparing the output to the predicted results.
- Given our interpretation of maximum capacity, the outputs were not far from what was expected.
- The runtime over many iterations remained consistent.
- Although American Airlines has the largest fleet of any U.S. airline company, United Airlines has the third largest but is the most represented in the OpenFlights data.



Source. IATA chart accessed through Wikipedia. (https://en.wikipedia.org/wiki/Largest_airlines_in_the_world)

QUESTIONS?