Get the Route for minimum flight delay between two cities

-- A optimization of linear programming

Team Member:
Shangxing Jiang
Liyan Chen
Yanghongbo Lu

Agenda

- Introduction the problem
- The Data Acquisition and preprocessing
- Modelling and Algorithm
- Linear Programming
- Results and Findings

Introduction the problem

- Overview
  - We model the problem of finding minimum-delay route between two cities as a shortest path problem with stochastic costs.
  - Graph Theory is used to abstract all the flights forming our model.
  - Dijkstra’s Algorithm is used to calculate the shortest path between Nodes(Airports).

Introduction the problem

- Background
  - *"The total cost of domestic air traffic delays to the U.S. economy was as much as **$32.9 billion** for 2007."*

    – U.S. Congress Joint Economic Committee

  - Nearly 20% flights delayed in US.
  - Flight delay is a problem both for travelers and airlines.

Year	Ontime Arrivals	Ontime (%)	Arrival Delays	Delayed (%)	Flights Cancelled	Cancelled (%)	Diverted	Flight Operations
2009	813,409	79.69%	186,743	18.29%	18,522	1.81%	2,075	1,020,749
2010	771,445	76.75%	191,949	19.10%	39,133	3.89%	2,552	1,005,079
2011	716,764	75.46%	189,787	19.98%	41,313	4.35%	2,052	949,916

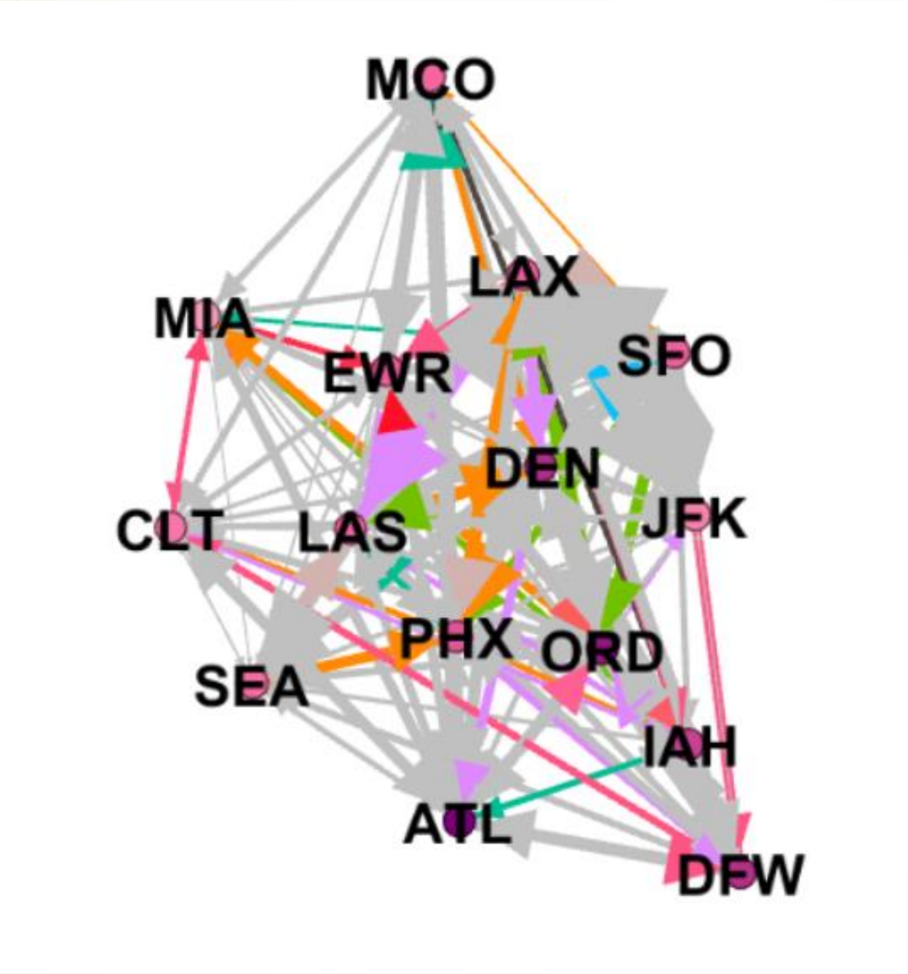
Introduction the problem

- Meaning of solving this problem
  - For airlines, identifying the causes of delay and predicting potential flight delays allow them to make smart schedules and improve their service.
  - For passengers, the ability to predict flight delay would allow them to make better informed decisions when making travel plans.

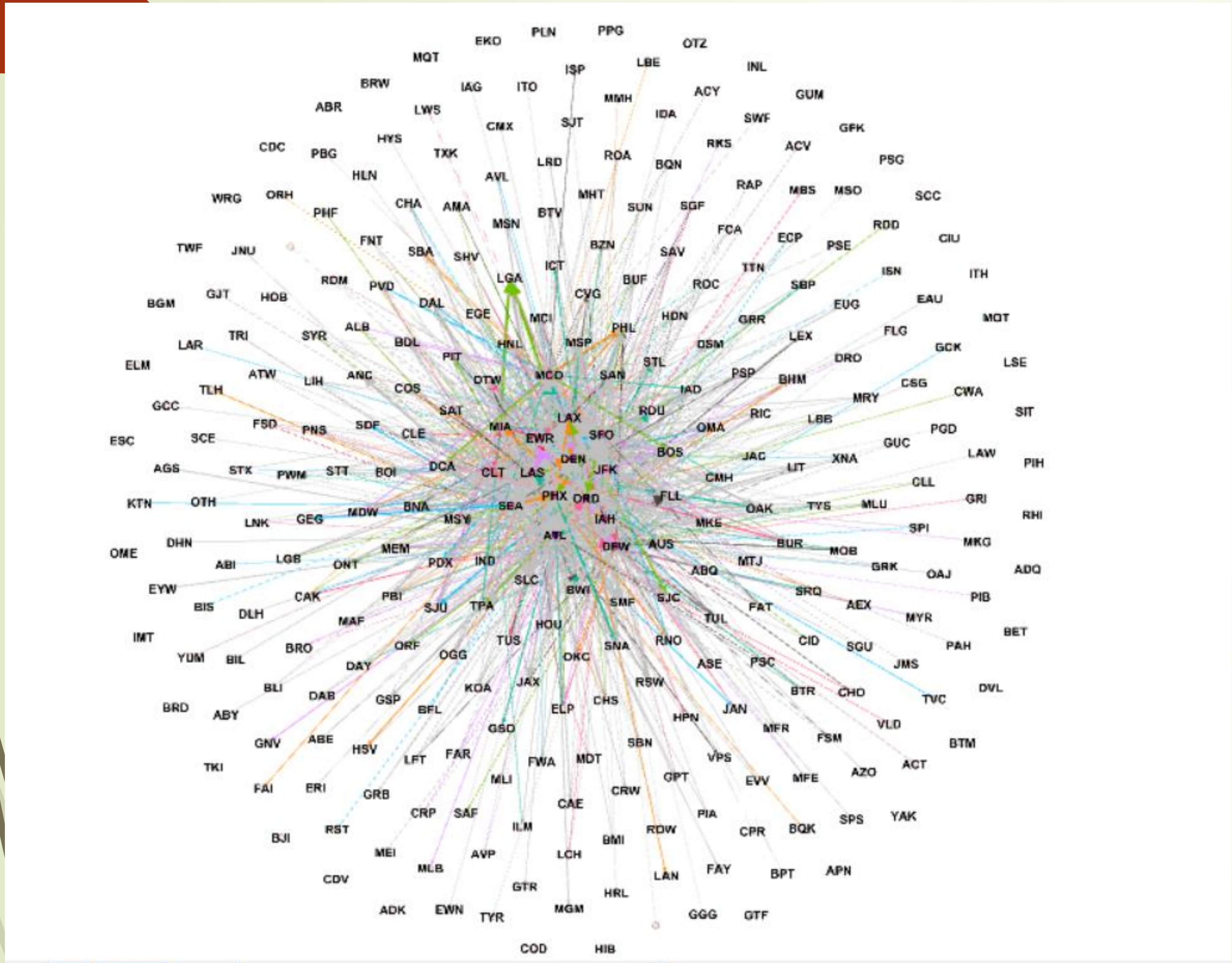
The Data Acquisition

- Source： The US Transportation Bureau
- Airline On-Time Performance Data
- Overview: data from 1980-2018.1
- The development of technology have influence on the delay
- Data: Choose data from 2017.1-2018
- Total: 585w+(too much!)
- Filter: Finding the 15th busiest airports in US as the origin
- Constraints: the total amount of the flight out as the degree
- Tools： Gephi and Excel
- Result: 19w+

The 15th busiest airports in US



ATL
CLT
DEN
DFW
EWR
IAH
JFK
LAS
LAX
MCO
MIA
ORD
PHX
SEA
SFO

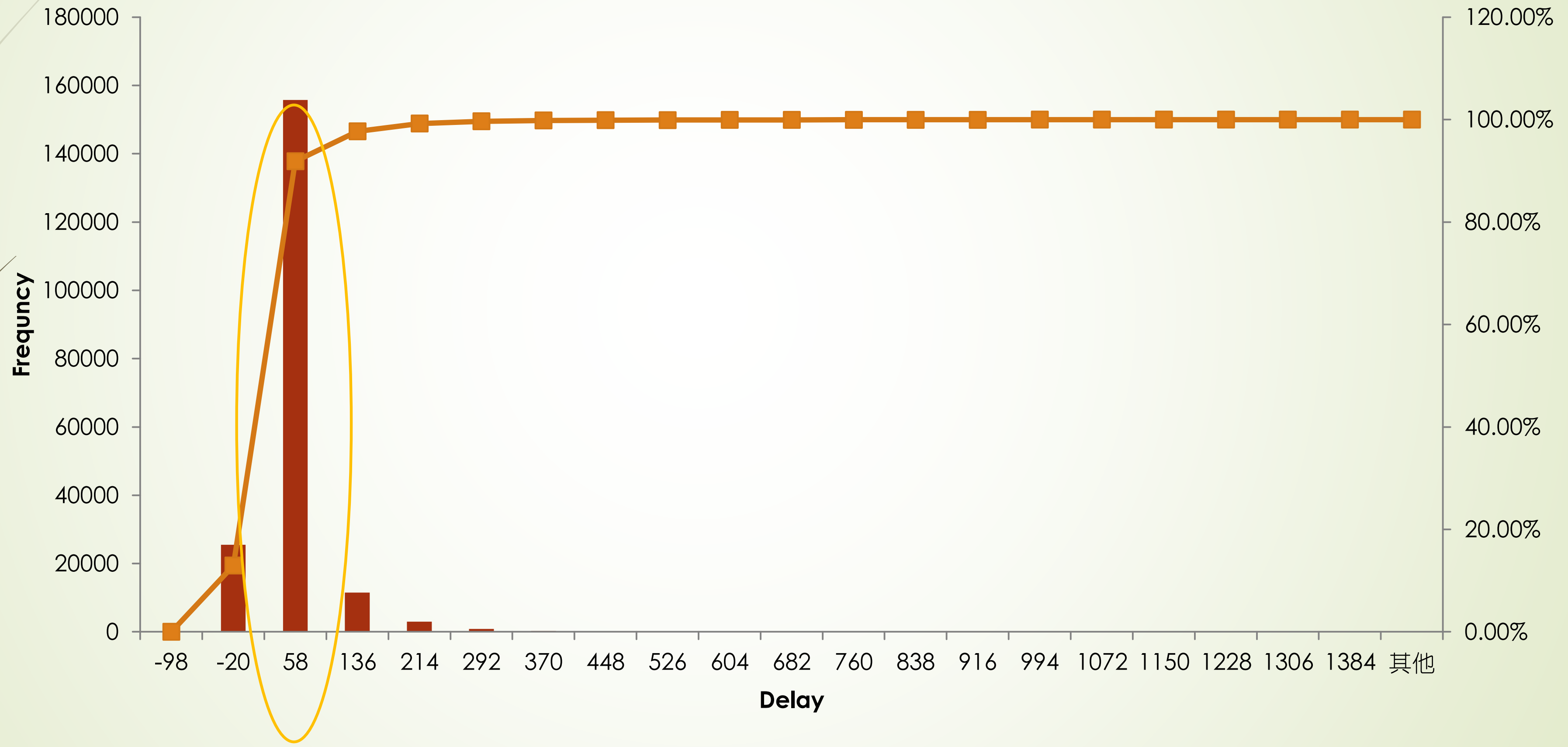


•Total airline of the 15 airports

Origin	15
Destination	229
Count	197389
Max delay	1455
Min delay	-98
Ave delay	8.44516918

Group	Dalay	Frequency	Percentage	Accumulation
1	-98	1	0.00%	0.00%
2	-20	25536	12.94%	12.94%
3	58	155777	78.92%	91.86%
4	136	11539	5.85%	97.70%
5	214	3014	1.53%	99.23%
6	292	882	0.45%	99.68%
7	370	294	0.15%	99.82%
8	448	129	0.07%	99.89%
9	526	42	0.02%	99.91%
10	604	44	0.02%	99.93%
11	682	36	0.02%	99.95%
12	760	29	0.01%	99.97%
13	838	19	0.01%	99.98%
14	916	12	0.01%	99.98%
15	994	17	0.01%	99.99%
16	1072	8	0.00%	99.99%
17	1150	3	0.00%	100.00%
18	1228	2	0.00%	100.00%
19	1306	2	0.00%	100.00%
20	1384	2	0.00%	100.00%

Histogram



- **Concentrated in [58,136)**

Delay description

- Reference： American Airlines, Delta Air Lines, Southwest Airlines, United Airlines, Frontier Airlines, JetBlue, Alaska Airlines, Hawaiian Airlines, Spirit Airlines, Virgin America
- Delay in plan: 30-60 mins(Taxi-in, Taxi-out, Traffic control, Loading)
- Acceptable delay: <=3 hrs (Weather, Traffic control)
- 98% flights in US have acceptable delay.
- However, delay still makes traveler annoyed!
- **Traveler need: Selecting a min delay airline!**

Graph Theory

- A graph $G = (V,E)$ is a set of vertices V, and edges E. For nodes $a, b \in V$, the notation $(a, b) \in E$ indicates that there is an edge from a to b in G.

- If direction and weights are introduced to our graph. G = (V, E, W)
- i.e. All edges have their directions and associated weights.

Data Preprocessing

- Graph: directed
- Nodes: Airports
- Node id: get from The US Transportation Bureau
- Edges: Airline
- Source: Origin airport (15)
- Target: Destination airport(229)
- Weight: Distance or Delay?

	A	B
1	AIRPORT_ID	AIRPORT
2	10135	ABE
3	10136	ABI
4	10140	ABQ
5	10141	ABR
6	10146	ABY
7	10155	ACT
8	10157	ACV
9	10158	ACY
10	10165	ADK
11	10170	ADQ
12	10185	AEX
13	10208	AGS
14	10257	ALB
15	10279	AMA
16	10299	ANC
17	10333	APN
18	10372	ASE
19	10397	ATL
20	10408	ATW
21	10423	AUS
22	10431	AVL
23	10434	AVP
24	10469	AZO
25	10529	BDL

Distance or Delay

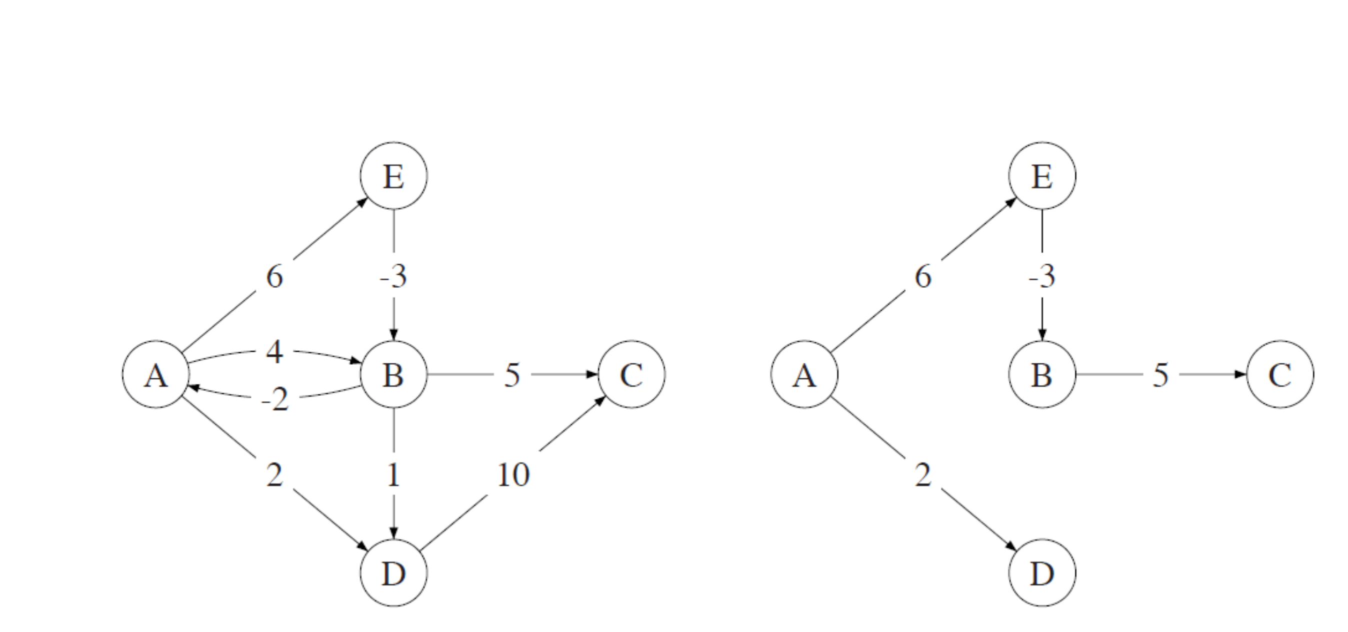
- Problem: Delay may be negative
- Assume: (JFK,IAD)=-3,(IAD,EWR)=2,(JFK,EWR)=3
- The program may choose (JFK,IAD,EWR)=-1 **WRONGE!**
- Airtime is more important when choosing a flight
- Airtime has positive correlation with distance
- Result: Choose **distance** as the first constraints and than consider the delay.

- Airline: a route from origin to destination
- Flight: there may be several different flights following a same airline offered by different companies.
- Using the **pivot table** to find **distance** of different airlines and **average delay** for all the flights.

	A	B	C	D	E	F	G	H
1	AIRLINE_I	FL_NUM	ORIGIN_A	DEST_AIR	ARR_DELA	AIR_TIME	DISTANCE	
2	20416	231	10397	11697	92	84	581	
3	20416	251	10397	13204	-18	57	404	
4	20416	252	10397	11042	41	78	554	
5	20416	363	10397	12266	5	104	689	
6	20416	403	10397	11697	69	83	581	
7	20416	404	10397	12892	6	273	1946	
8	20416	481	10397	13495	-16	71	425	
9	20416	541	10397	15304	0	60	406	
10	20416	556	10397	10821	53	74	577	
11	20416	565	10397	13204	-3	62	404	
12	20416	600	10397	14100	-9	80	666	
13	20304	5868	10397	13930	-9	89	606	

The Shortest path problem

- predecessor function π : V →V , mapping a vertex with has the shortest path to the origin
- function *d* : *V* →R , the weight of the shortest path between the origin node and another node
- **Required:**π, *d*, origin n_o, destination n_d
- *n* ← n_d
- *path* ←list(n_d)
- **repeat**
- *n* ← π(*n*)
- append *n* to *path*
- **until** *n* = *io* n_o
- reverse *path*
- **return** *path*, *d*(n_d)



Dijkstra's Algorithm

- **State label** c： 0 not been visited, 1 visited
- **Require:** G = (V ,E,c,w), origin $n_o, FIFO\ queue\ Q\ for\ storing\ the\ visited\ nodes$
- **for** v **in** V **do**
- π(v) ← v,d(v) ← ∞,c(v) ← 0
- **end for**
- $d(n_o) \leftarrow 0$,set Q ←V
- **while** Q not empty **do**
- *n* ←minQ,remove *n* from Q
- **for** v ∈ V adjacent to *n* **do**
- relax(π,*d*,*w*, tail *n*, head v)
- **end for**
- **end while**
- **return** π,*d*

Optimization Model

$$\min\{\sum_{(i,j\in E)} w_{ij}\gamma_{ij} : \gamma_{ij} \in \Omega\}$$

$$\Omega = \{ \sum_{\substack{(i,j\in\epsilon)\\ Flights\ out\ of\ i}} \gamma_{ij} - \sum_{\substack{(i,j\in\epsilon)\\ Flights\ into\ j}} \gamma_{ij} = b_i, \forall i \in V\}$$

- w_{ij} is the weight of each edge, here stands for the distance of each airline
- $\gamma_{ij} \in \{0,1\}$: yes-or-no *decision variables* representing whether airline (i, j) is included
- b_i = -1 origin, 1 destination, 0 otherwise
- When an airline is chosen, find min delay **FL_NUM** of this airline
- Go over the shortest path to get a recommended flight set

Sample Code （Python2.7）

```
class Graph:
    def __init__(self):
        self.vert_dict = {}
        self.num_vertices = 0

    def __iter__(self):
        return iter(self.vert_dict.values())

    def add_vertex(self, node):
        self.num_vertices = self.num_vertices + 1
        new_vertex = Vertex(node)
        self.vert_dict[node] = new_vertex
        return new_vertex

    def get_vertex(self, n):
        if n in self.vert_dict:
            return self.vert_dict[n]
        else:
            return None

    def add_edge(self, frm, to, cost = 0):
        if frm not in self.vert_dict:
            self.add_vertex(frm)
        if to not in self.vert_dict:
            self.add_vertex(to)
```

Sample Code （Python2.7）

```
import heapq

def dijkstra(aGraph, start, target):
    #print '''Dijkstra's shortest path'''
    # Set the distance for the start node to zero
    start.set_distance(0)

    # Put tuple pair into the priority queue
    unvisited_queue = [(v.get_distance(),v) for v in aGraph]
    heapq.heapify(unvisited_queue)

    while len(unvisited_queue):
        # Pops a vertex with the smallest distance
        uv = heapq.heappop(unvisited_queue)
        current = uv[1]
        current.set_visited()

        #for next in v.adjacent:
        for next in current.adjacent:
            # if visited, skip
            if next.visited:
                continue
            new_dist = current.get_distance() + current.get_weight(next)
```

Sample Code （Python2.7）

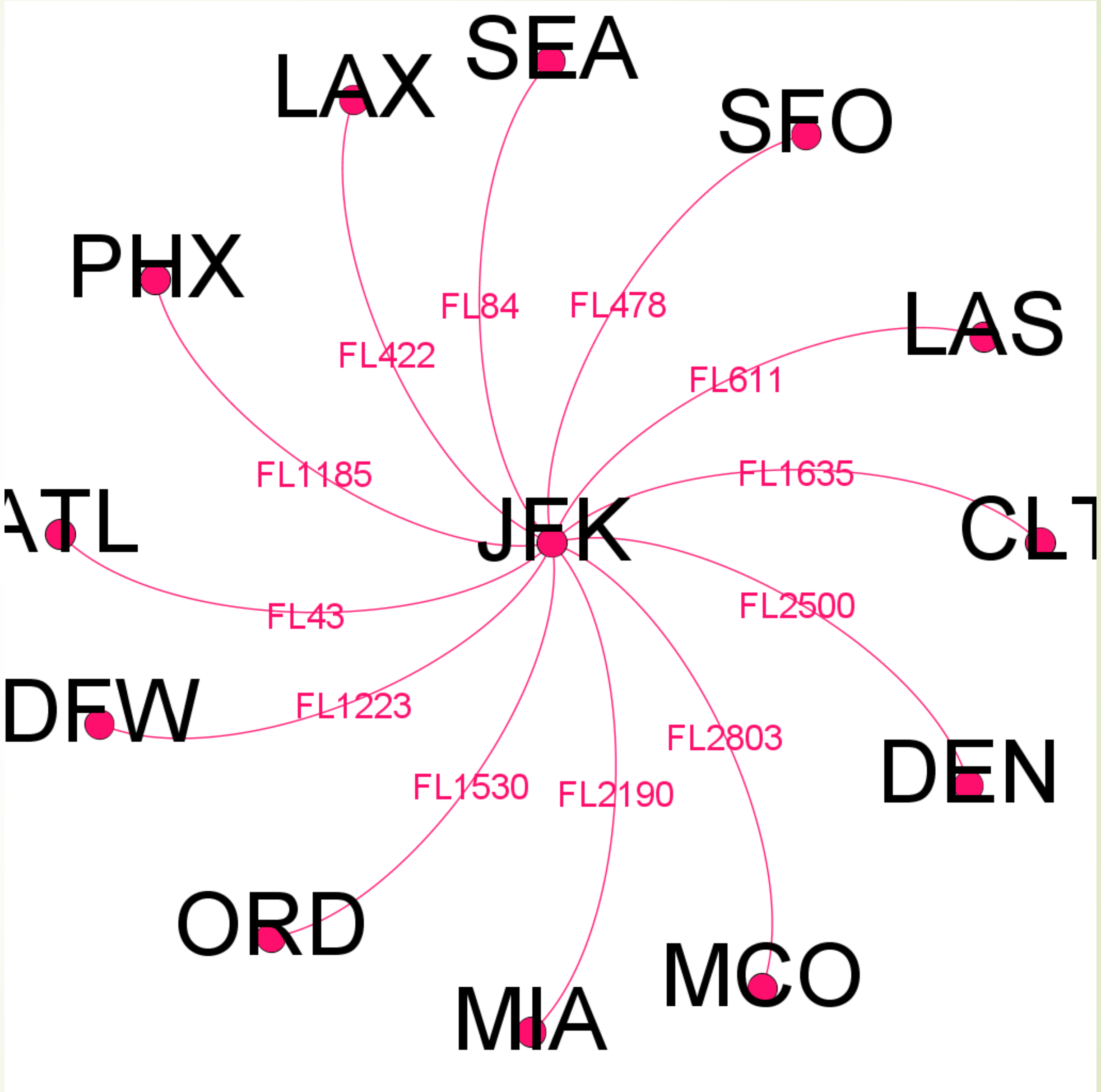
```
def find_min_delay(x, y):
    a=unique_index(dep, x)
    b=unique_index(arr, y)
    c=[]
    for i in range(len(a)):
        for j in range(len(b)):
            if a[i]==b[j]:
                c=np.append(c, a[i])
    common=np.empty(len(c),dtype=object)
    for i in range(len(c)):
        common[i]=int(c[i])
    arr_d=1000000
    for i in range(len(common)):
        if int(arrival_delay[common[i]]) < arr_d:
            arr_d=int(arrival_delay[common[i]])
    return arr_d
```

Test Result

- We did a test to find out recommended flights with min delay between JFK to the other 12 airports

oringin_airport	oringin_airport_ID	arrival_airport_ID	arrival_airport	min_delay	reconmmended flights	shortest_path:none for none stop flight
JFK	12478	ATL	10397	-28	43	12478','10397'
JFK	12478	CLT	11057	-31	1635	12478','11057'
JFK	12478	DEN	11292	-50	2500	12478','11292'
JFK	12478	DFW	11298	-32	1223	12478','11298'
JFK	12478	LAS	12889	-51	611	12478','12889'
JFK	12478	LAX	12892	-62	422	12478','12892'
JFK	12478	MCO	13204	-39	2803	12478','13204'
JFK	12478	MIA	13303	-37	2190	12478','13303'
JFK	12478	ORD	13930	-42	1530	12478','13930'
JFK	12478	PHX	14107	-48	1185	12478','14107'
JFK	12478	SEA	14747	-56	84	12478','14747'
JFK	12478	SFO	14771	-47	478	12478','14771'

Flight Map



LAX
SEA
SFO
PHX
LAS
ATL
JFK
CLT
DFW
DEN
ORD
MIA
MCO
FL84
FL478
FL422
FL611
FL1185
FL1635
FL43
FL2500
FL1223
FL2803
FL1530
FL2190

Thanks!

With any problems, send email to:

Shangxing Jiang sjiang25@stevens.edu
Liyan Chen lchen39@stevens.edu
Yanghongbo Lu ylu44@stevens.edu