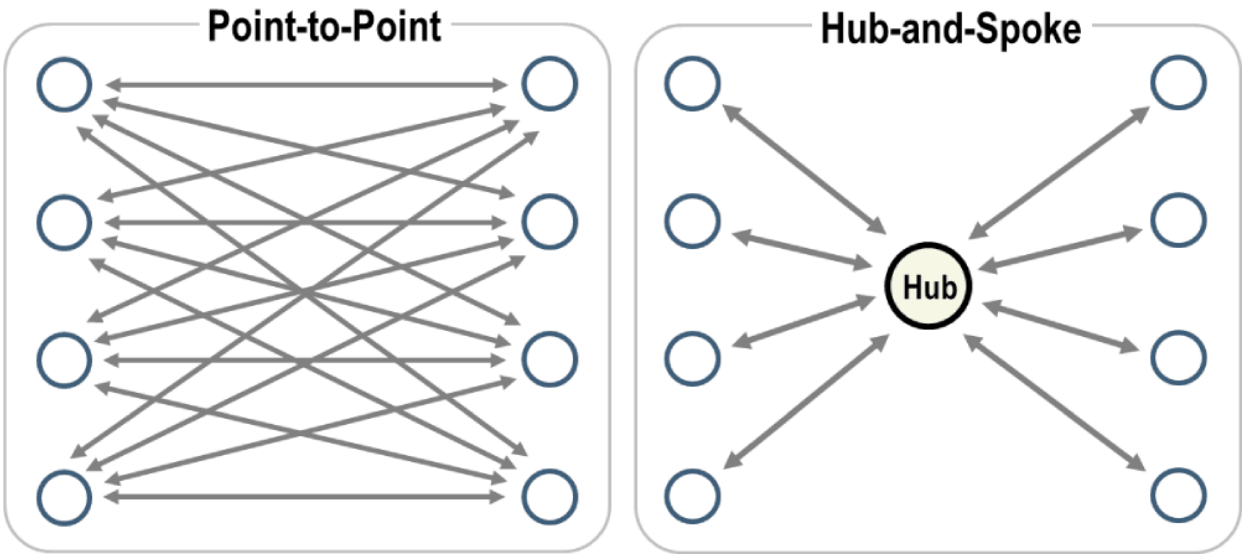
EE 578B Convex Optimization Project

Kyle Hadley

Introduction

Working within the aviation industry, one of the most interesting convex optimization problems that airlines face involves deciding which routes should be offered to their customers. There are many different factors that are involved in choosing the routes: passenger demand, cost of operating route, aircraft available to operate the route, etc.

Based on these factors, there have been predominately two approaches to planning airline's routes: (1) the hub-and-spoke model or (2) point-to-point.



The hub-and-spoke model mimics the spokes of a wheel spreading out from a single; typically airlines has a few select hubs (depending on an airline's size) that all flights fly through. An example of this is Alaska Airlines hub based out of the Seattle-Tacoma airport. The majority of the flights operated by Alaska Air flows through this airport. In this model, the airline chooses to connect as many cities to a single hub such that customers can access a larger network of cities, but are likely to require at least 1 or 2 layover flights. This enables an airline to operate larger aircraft between major hubs (e.g. Los Angeles and New York) and then utilize smaller, regional aircraft for any small flights from the hub to smaller cities (e.g. Log Angeles to San Diego).

The point-to-point model instead attempts to connect as many cities directly rather than through a single hub. The point-to-point model emphasizes flying between two cities directly, regardless of size. An example of this approach is RyanAir which operates many low-cost flights through Europe continent. RyanAir attempts to connect as many cities, regardless of the size of cities to provide easy to access flights. Airlines can often more easily ensure that their routes are meeting the customer

demand for a specific pair of cities as they don't need to be as concerned with ensuring "connecting" flights are supported.

There are certainly disadvantages to both solutions. A hub-and-spoke model often leads to higher facility costs and requires passengers to often make connecting flights to reach their final destination (ultimately leading to longer travel times). A point-to-point model also requires significantly more aircraft in a given fleet to support the same number of routes; sometimes this may mean that cities cannot be supported within a point-to-point model if an airline does not have adequate aircraft within their fleet.

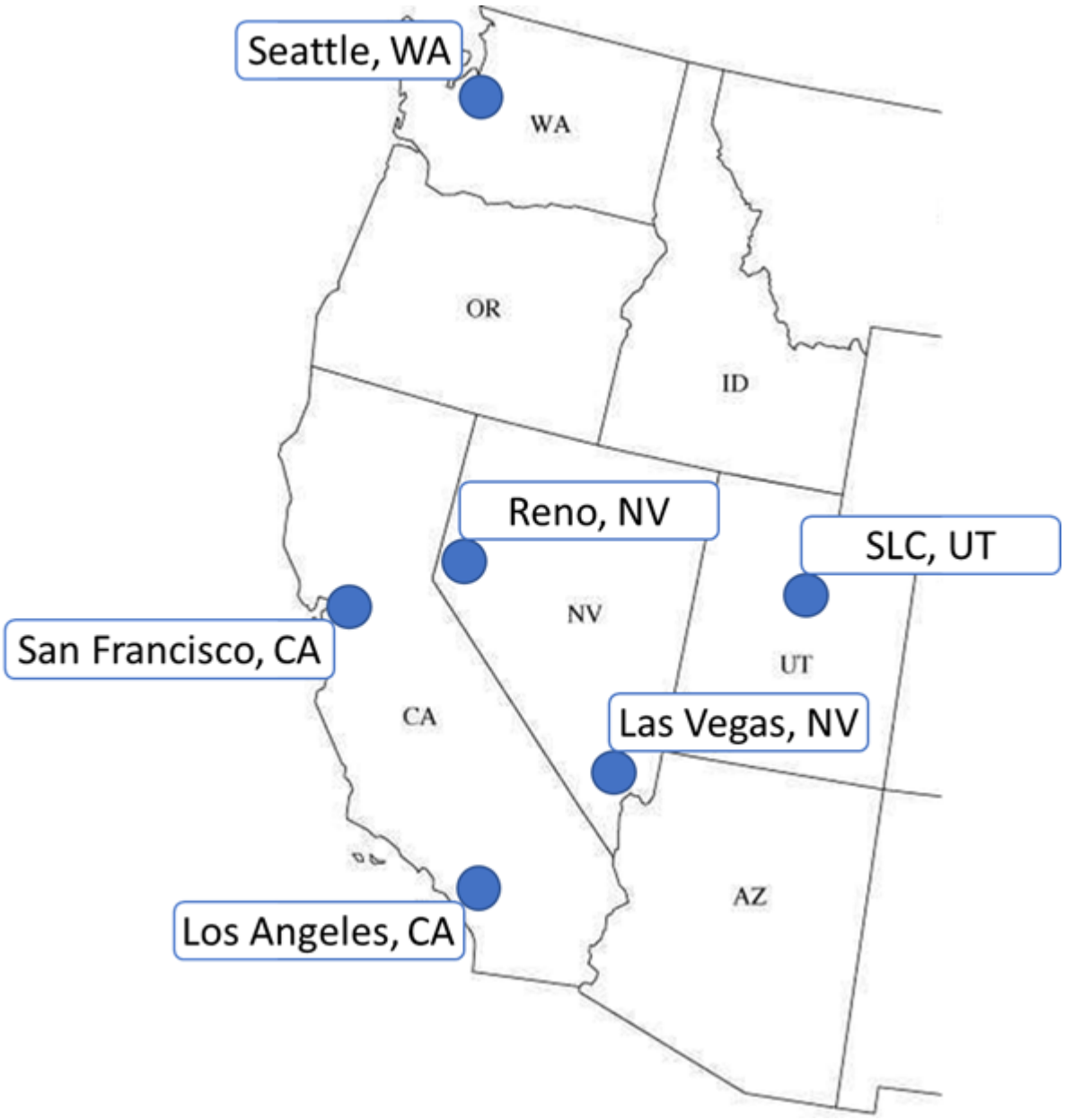
Through my project, I hope to provide the foundation of a tool that could allow airlines to more quickly plan their routes based on customer demand. The model will compare multiple networks connected cities - following the hub-and-spoke model, point-to-point, and a combination of the two - to identify the optimal routes to maximize customer reward.

Sample Airline for Analysis

To help develop the preliminary aspects of the model, I'll be imagining that my model is to be used for an upstart airline - *Concave Airlines* - to help them identify how they should offer routes between various cities to support customer demands. *Concave Airlines* is based out of Las Vegas, Nevada and is focused on mainly offering routes along the west coast and mountain states of the US, but also is interested in potentially supporting other major cities across the US and in North America.

The following cities are currently of interest to be serviced by the airline:

- Las Vegas, NV
- Reno, NV
- Salt Lake City, UT
- Seattle, WA
- Los Angeles, CA
- San Francisco, CA



Problem Setup

Initial Model

We can model our routes as a graph and calculate the minimization akin to the approach taken when modeling shortest path problems with edges formulated. Thus, we can think of the model generically as minimizing the cost of mass flow such that:

$$\min_x c^T x$$

$$\text{s.t. } Ex = b, x \geq 0$$

Within our example, the nodes of our graph represents the cities our airline will service and each edge represents a flight route that our airline will support between two pair cities.

Graph Structures

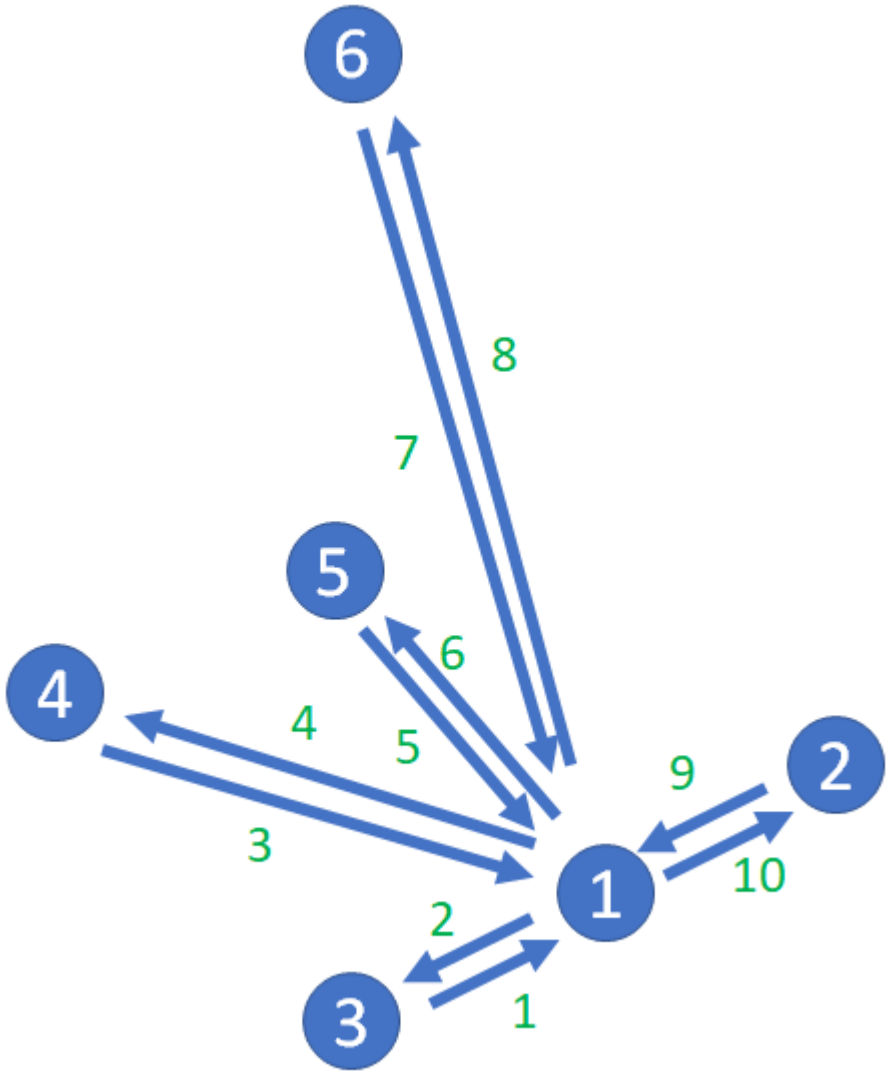
In order to leverage this general linear program, we must must have a defined graph that can be used to define the various variables and constraints in the model. However, as we discussed before, there many combinations of flight routes that *Concave Airlines* could service. It may be advantageous to follow a hub-and-spoke model, requiring all flights to flow through Las Vegas (the airlines main hub). Or the airline could choose to connect all the cities via a point-to-point model.

Thus, we can create two graphs to compare the cost of utilizing the two methods we've already discussed. Once we've created each graph, we can identify which graph yields the lowest travel cost when servicing the same city pairs (e.g. if we have X flights between city A and B, which would yield the lowest travel cost). Travel cost in our case is measured as flight minutes or hours.

For both of our graphs that we will be evaluating, the node numbering maps to the following city names:

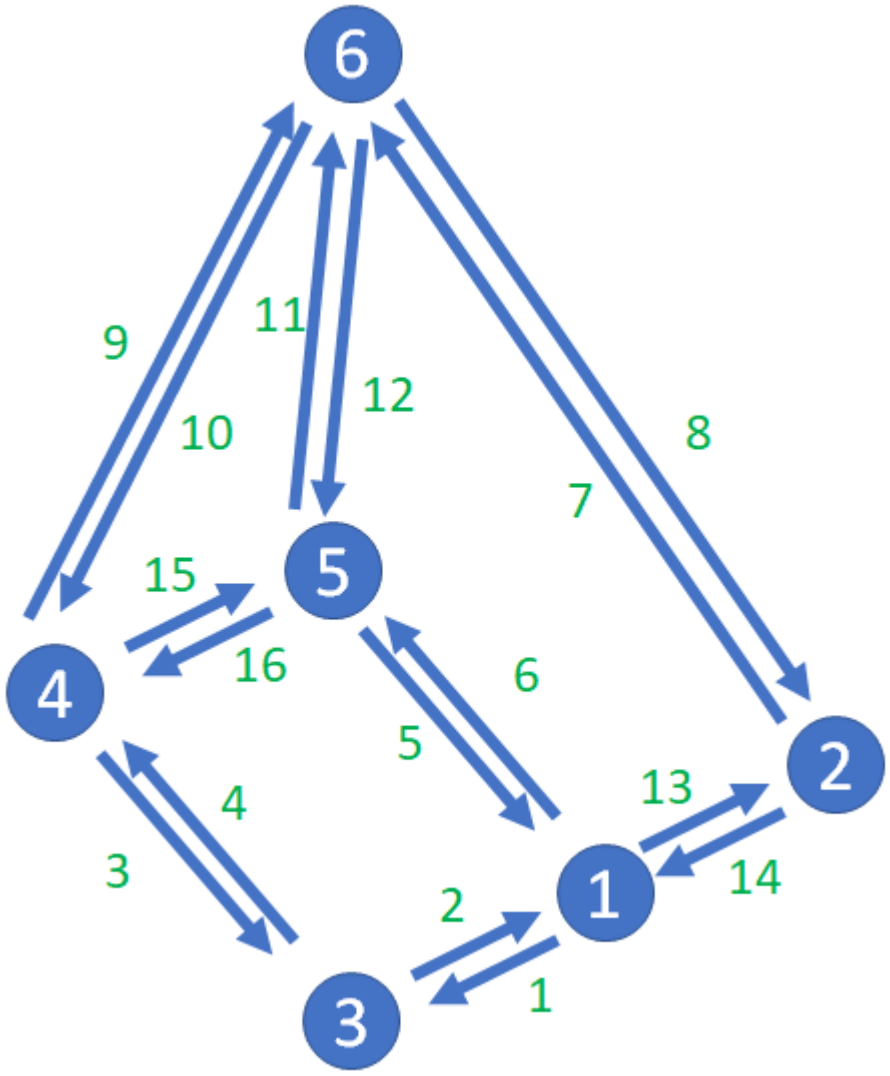
1. Las Vegas
2. Salt Lake City
3. Los Angeles
4. San Franscisco
5. Reno
6. Seattle

Our first graph will utilize the hub-and-spoke model as discussed above; the graph is illustrated below:



From this graph, we can see that we have 6 nodes (i.e. cities) and 12 total edges (i.e. flight routes). Thus, we can see that our $E \in \mathbb{R}^{6x12}$ and $b \in \mathbb{R}^{6}$.

Our second graph will utilize more of a point-to-point model as discussed above; the graph is illustrated below:



From this graph, we can see that we have the same 6 nodes but 16 total edges. Thus, we can see that our $E \in \mathbb{R}^{6x16}$ and $b \in \mathbb{R}^{6}$.

As discussed in the next section, our c matrix represents the cost of traveling along a given edge - we can estimate this using Google Maps estimates of travel time by flight between the city pairs.

Model Modifications

Once of the initial issues that can be seen with the initial linear program is that the initial program assumes that we are only moving mass (i.e. our customers) from an single initial node to a single final node. However, *Concave Airlines* will not just be servicing a single set of passengers between two cities, there will be numerous flights between these various cities on a given day. Thus, in order to account for numerous flights, we need our model to account for having multiple mass flows, initial, and final nodes.

We can update our model to account for this by modifying our equation such that,

$$\min_{x} \sum_{i}^{n} c^T x_i$$

$$\text{s.t. } Ex_i = b_i, x \geq 0 \forall i$$

where x_i represents a single flight between two cities as designated by our matrix b_i. We can maintain our minimization and inequality term as we want to minimize all of x and we still want x to be greater than or equal to zero.

For the purposes of this initial sample code, a random set of 100 routes will be generated between the cities in our graph. The randomization of routes generated will be a function of population of the departure city - as it's reasonable to assume more people are leaving a city with a higher population than a smaller one. We will then compare the resulting travel cost for these 100 routes against the two graphs discussed above.

The last modification that will be made to our model is the incorporation of costs associated with edges themselves. As normally experienced by passengers, there is a travel cost associated with the airport itself when leaving an airport - both when making a connecting flight or on the first flight of the journey. In order to account for this, we can incorporate a cost d at each edge in E_o such that our cost can be written as: $(c^T + d^T E_o)$. Thus, we re-write our problem as,

$$\min_x \sum_i^n (c^T + d^T E_o) x_i$$

$$\text{s.t. } Ex_i = b_i, x \geq 0 \forall i$$

Solving for this minimization problem should calculate the total flight hours required to support the 100 flights of interest.

Optimization Problems

Primal Problem

In the previous section, we defined the primal problem already; thus it can be written as,

$$\min_x \sum_{i=1}^n (c^T + d^T E_o) x_i$$

$$\text{s.t. } Ex_i = b_i, x \geq 0 \forall i$$

In our primal problem, our variables represent the following:

- c^T represents the travel cost along a given edge - i.e. the time to flight from one city to another, in minutes
- d^T represents the cost of traveling through a city, in minutes; this will be calculated as function of the population of each city
- E represents the graph flow which is solved as $E_i - E_o$
- x represents the flow of people / flights through our graph
- i represents a specific flight; thus, we attempting to calculate the cost of all flights through our graph
- b_i represents the departure and arrival cities for our given flight i

- n is the number of flights we'll be assessing

The objective is ultimately attempting to calculate the total flight hours of the fleet of aircraft for the 100 flights randomly generated. The first constraint - $Ex_i = b_i$ - ensures that a flight moves from its arrival to its destination along viable flight paths (as designated by our graph). The second constraint - $x >= 0$ - ensures that flights don't move backwards through our graph structure (which makes sense, once we depart from a city we don't want to return to it to reach out destination).

Dual problem

Solving for the dual optimization problem, we can solve for the Lagrangian of our problem as follows,

$$\mathcal{L}(x, v, \mu) = \sum_{i=1}^{n}(c^T + d^T E_o)x_i + \sum_{i=1}^{n} v_i^T(Ex_i - b_i) - \sum_{i=1}^{n} \mu_i^T x_i$$

We recall from our previous homeworks and lecture notes that we can solve for our dual constraint by taking the derivative of this function and setting it equal to zero; thus,

$$\frac{\partial}{\partial x}\mathcal{L}(x, v, \mu) = (c^T + d^T E_o) + v_i^T E - \mu_i^T = 0$$

Thus, our dual constraint is $(c^T + d^T E_o) + v_i^T E - \mu_i^T = 0$. Plugging in our dual constraint into our Lagrangian, we can see that our dual objective is $\sum_{i=1}^{n} -v_i^T b_i$. Thus, we can re-write our dual problem as,

$$\max_{v,\mu} \sum_{i=1}^{n} -v_i^T b_i$$

$$\text{s.t. } (c^T + d^T E_o) + v_i^T E - \mu_i^T = 0, \mu \geq 0$$

In our dual problem, our variables represent the following:

- v represents our value function, which the total cost to go from our initial state to our final state - i.e. the cost from destination to arrival
- μ represents ineffiences along each edge, which should become zero as we solve for our optimal value

Applications

Both the primal and dual optimization problem are solved in the code included at the end of this PDF. As mentioned previously, our nodes consist of the cities we wish to service which include:

- Las Vegas, NV
- Reno, NV

- Salt Lake City, UT
- Seattle, WA
- Los Angeles, CA
- San Francisco, CA

The graphs follow the graphs as illustrated in figures above for both the hub-and-spoke and point-to-point model. The costs along each edge were calculated using Google Maps flight estimates between city pairs. To generate our list of flights, I used the population of each city in my list and created a probability array out of the population. The city's with the higher populations were more likely to be included as either a destination or arrival airplane.

The results of the code are discussed in the next section.

Discussion

Results

After generating a randomized sample of 100 flights, the optimization problem as discussed was ran for both graph structures. The resulting primal and dual problem optimal values - i.e. total flight hours - for the graph structures are defined in the table below.

Hub-and-spoke Graph	Point-to-point Graph
3304 flight hours	3035 flight hours

While the results shown in the table above is the result for a random sample of 100 flights, the trend of the point-to-point having a lower cost (i.e. total flight hours) than the hub-and-spoke was consistent for numerous samples of flights. Across multiple test samples, the average difference in time ranged from $5 - 10\%$. While this difference may not seem substantial, this is considered a significant amount of savings in the context of the small margin aviation business.

Upon reviewing the results, it seems that the point-to-point graph is more likely to succeed because it provides greater versatility for travel *from* the largest cities (namely Los Angeles). As discussed in previous sections, the randomly sampled flights was weighted with respect to population size of departure cities. In the point-to-point graph, travelers from these larger cities are more likely to immediately travel to their destination city without requiring a connection, given that there is a greater chance to have a direct flight. With the hub-and-spoke model, all flights have to travel through Las Vegas (node 1), which requires a minimum of 1 connecting flight for any flight not including Las Vegas as destination or arrival city.

Future expansion

Some potential future expansions of this work could including,

- Model the cost of adding an edge to our graph; currently the model is able to account for the cost of having an "unoptimized" set of edges in our graph but it doesn't not account for the

added cost of having additional edges. Having an additional edge, in the context of the this example, results in additional cost in terms of additional aircraft to perform the route, additional staff to support the additional gates at the airport, etc.

- Model in conjunction with the cost the reward (i.e. financial incentive) of supporting the flights as inputted; likely would involve a dual optimization type of problem with minimizing cost but maximizing revenue
- Ability to evaluate the added cost of adding an additional city / node to the graph; would be valuable for an airline to understand what other cities may lead to additional revenue
- Updating the model to be a function of time in order to better account for the way in which flights can't occur all at the same time, ultimately requiring stagnation in between flights or additional delays due to air or ground traffic congestion while performing the flights
- Accounting for the cost of congestion through specific cities; it would be beneficial to understand how funneling flights through specific hub cities would result in additional cost to the system due to congestion of multiple flights through a single node in our graph

References

- https://aeronauticsonline.com/the-airline-economics-of-the-bicycle-wheel-point-to-point-vs-hub-and-spoke-flying/
- https://transportgeography.org/contents/chapter2/geography-of-transportation-networks/point-to-point-versus-hub-and-spoke-network/#:~:text=A%20point%2Dto%2Dpoint%20network,intermediary%20location%20called%20a
- https://simpleflying.com/how-do-airlines-plan-routes/

Code

In [1]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import matplotlib.lines as mlines
import cvxpy as cp
from mpl_toolkits.mplot3d.art3d import Poly3DCollection
import warnings
warnings.simplefilter('ignore')
np.random.seed(0)
```

In [2]:
```
# First randomly generate a set of 100 flights be used in our calculations
number_of_flights = 100
nodes = 6

# pop is the population of each of cities in our graph; prob will be used for generatin
pop = np.array([[6.34773e5, 1.97756e5, 3.967e6, 8.74961e5, 2.465e5, 7.24305e5]])
prob = pop / pop.sum()

# Our flights (designated by b)
b = np.zeros(shape=(number_of_flights, nodes))

for i in range(number_of_flights):
    # Generate our b_i by randomly selecting two city pairs with a weighted probability
    b_ones = np.random.choice(nodes, size=2, replace=False, p=np.squeeze(prob))
    b[i, b_ones[0]] = 1 # Pick a random starting city for flight
    b[i, b_ones[1]] = -1 # Pick a random end city for flight

# d is the cost of going through an airport, which is a function of population size as
d = 0.001*pop.T

print('---First Ten Flights Generated---')
print(b[0:10])
print('\nKey:\n-1 = Departure city\n 1 = Arrival City')
```

```
---First Ten Flights Generated---
[[ 0.  0.  1. -1.  0.  0.]
 [ 0.  0.  1.  0. -1.  0.]
 [ 0.  0.  1.  0.  0. -1.]
 [ 0.  0. -1.  0.  0.  1.]
 [ 0.  0. -1.  1.  0.  0.]
 [ 0.  0.  1.  0.  0. -1.]
 [ 1. -1.  0.  0.  0.  0.]
 [ 0.  0.  0.  1. -1.  0.]
 [ 0.  0.  0. -1.  0.  1.]
 [ 0.  0.  1. -1.  0.  0.]]

Key:
-1 = Departure city
 1 = Arrival City
```

Hub-and-spoke Model

Primal Problem

In [3]:

```
# Set the number of edges for the given model (Hub-and-spoke)
edges = 10

# Costs are equivalent to the number of minutes required to travel between two cities (
c = np.array([[65, 65, 85, 85, 70, 60, 150, 140, 80, 75]]).T

# Generate our E matrices based on the hub-and-spoke routes
E_i = np.array([[1, 0, 1, 0, 1, 0, 1, 0, 1, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
                [0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 1, 0, 0]])
E_o = np.array([[0, 1, 0, 1, 0, 1, 0, 1,  0, 1],
                [0,  0, 0,  0, 0,  0, 0,  0, 1,  0],
                [1, 0, 0,  0, 0,  0, 0,  0,  0,  0],
                [0,  0, 1, 0, 0,  0, 0,  0,  0,  0],
                [0,  0,  0, 0,1,  0, 0,  0,  0,  0],
                [0,  0,  0, 0, 0,  0,1,  0,  0,  0]])

E = E_i - E_o

# x is a variable with shape |e| x |i| where i is the the number of routes
x = cp.Variable([edges, number_of_flights])
obj = 0.
constraints = [x>=0]

for i in range(number_of_flights):
    obj = obj + (c.T + d.T@E_o)@x[:, i]

    constraints.append(E @ x[:, i] == b[i, :])

primal = cp.Problem(cp.Minimize(obj), constraints)
primal.solve()

print('Primal Problem')
print('Optimal Value:', np.round_((primal.value / 60), decimals=2), 'Flight Hours')

print('\nOptimal x (i.e. Route Taken for Given Flight):')
print(np.round_(np.absolute(x.value[:, 0:5]), decimals=1), 'Flight Hours')
print('\nKey:\n Cols = Flight(s)\n Rows = Edges Traveled (i.e. Legs of Flight)\n 1 = Ci
```

```
Primal Problem
Optimal Value: 3303.8 Flight Hours

Optimal x (i.e. Route Taken for Given Flight):
[[0. 0. 0. 1. 1.]
 [1. 1. 1. 0. 0.]
 [1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1.]
 [0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]] Flight Hours

Key:
 Cols = Flight(s)
```

```
Rows = Edges Traveled (i.e. Legs of Flight)
1 = City is in route
```

Dual Problem

In [4]:

```
v = cp.Variable([nodes, number_of_flights])
mu = cp.Variable([edges, number_of_flights])

obj = cp.sum(-v.T @ b.T)
constraints = [mu >= 0]
constraints.append((c.T + d.T@E_o) + v.T@E - mu.T == 0)

obj = 0.
constraints = [mu>=0]

for i in range(number_of_flights):
    obj = obj + (-v[:, i].T @ b[i, :])
    constraints.append(np.squeeze(c.T + d.T@E_o) + v[:,i].T@E - mu[:,i].T == 0)

dual = cp.Problem(cp.Maximize(obj), constraints)
dual.solve()

print('Dual Problem')
print('Optimal Value:', np.round_((dual.value / 60), decimals=2), 'Flight Hours')

print('\nOptimal v (1st 5 Columns):')
print(np.round_(v.value[:, 0:5], decimals=2))

print('\nOptimal mu (1st 5 Columns):')
print(np.round_(mu.value[:, 0:5], decimals=1))
```

```
Dual Problem
Optimal Value: 3303.8 Flight Hours

Optimal v (1st 5 Columns):
[[   15.88    71.59    18.44  -495.38  -492.82]
 [ -200.15  -144.45  -197.6   -711.36  -708.81]
 [ -683.89  -628.18  -681.34  3536.62  3539.18]
 [  975.84   191.63   138.48  -375.24 -1212.6 ]
 [ -173.28   388.09  -170.73  -684.49  -681.94]
 [   65.6    121.31   892.74 -1270.15  -443.01]]

Optimal mu (1st 5 Columns):
[[4731.8 4731.8 4731.8    0.     0. ]
 [   0.     0.     0.  4731.8 4731.8]
 [   0.   839.9  839.9  839.8 1679.7]
 [1679.7  839.8  839.8  839.9    0. ]
 [ 505.7    0.   505.7  505.6  505.6]
 [ 505.6 1011.3  505.6  505.7  505.7]
 [ 824.6  824.6    0.  1649.1  824.5]
 [ 824.5  824.5 1649.1    0.   824.6]
 [ 493.8  493.8  493.8  493.7  493.7]
 [ 493.7  493.7  493.7  493.8  493.8]]
```

Point-to-point Model

Primal Problem

In [5]:

```
# Costs are equivalent to the number of minutes required to travel between two cities (
```

```
c = np.array([[65, 65, 80, 80, 70, 80, 130, 115, 125, 125, 110, 100, 75, 80, 60, 75]]).

# Generate our E matrices based on the hub-and-spoke routes
E_i = np.array([[0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0],
                [1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1],
                [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0],
                [0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0]])

E_o = np.array([[1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0],
                [0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0],
                [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1],
                [0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0]])

E = E_i - E_o

# x is a variable with shape |e| x |i| where i is the the number of routes
edges=16
x = cp.Variable([edges, number_of_flights])
obj = 0.
constraints = [x>=0]

for i in range(number_of_flights):
    obj = obj + (c.T + d.T@E_o)@x[:, i]

    constraints.append(E @ x[:, i] == b[i,:])

primal = cp.Problem(cp.Minimize(obj), constraints)
primal.solve()

print('Primal Solution:', np.round_((primal.value / 60), decimals=2), 'Flight Hours')

print('\n---Route Taken for Given Flight---')
print(np.round_(np.absolute(x.value[:, 0:5]), decimals=1))
print('\nKey:\n Cols = Flight(s)\n Rows = Edges Traveled (i.e. Legs of Flight)\n 1 = Ci
```

```
Primal Solution: 3035.53 Flight Hours

---Route Taken for Given Flight---
[[0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [1. 0. 1. 0. 0.]
 [0. 0. 0. 1. 1.]
 [0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]]

Key:
 Cols = Flight(s)
```

Rows = Edges Traveled (i.e. Legs of Flight)
1 = City is in route

In [6]:

```
v = cp.Variable([nodes, number_of_flights])
mu = cp.Variable([edges, number_of_flights])

obj = cp.sum(-v.T @ b.T)
constraints = [mu >= 0]
constraints.append((c.T + d.T@E_o) + v.T@E - mu.T == 0)

obj = 0.
constraints = [mu>=0]

for i in range(number_of_flights):
    obj = obj + (-v[:, i].T @ b[i, :])
    constraints.append(np.squeeze(c.T + d.T@E_o) + v[:,i].T@E - mu[:,i].T == 0)

dual = cp.Problem(cp.Maximize(obj), constraints)
dual.solve()

print('Dual Problem')
print('Optimal Value:', np.round_((dual.value / 60), decimals=2), 'Flight Hours')

print('\nOptimal v (1st 5 Columns):')
print(np.round_(v.value[:, 0:5], decimals=2))

print('\nOptimal mu (1st 5 Columns):')
print(np.round_(mu.value[:, 0:5], decimals=1))
```

```
Dual Problem
Optimal Value: 3035.53 Flight Hours

Optimal v (1st 5 Columns):
[[ -222.31    28.04  -211.24  -258.71  -244.05]
 [ -266.47   -79.49    64.68  -965.04  -633.12]
 [ -295.7   -671.74  -902.3   3757.97  3148.85]
 [  659.26    69.92    52.66  -289.03  -898.15]
 [  -72.63   344.54    94.23  -956.2   -786.3 ]
 [  197.84   308.73   901.97 -1288.99  -587.23]]

Optimal mu (1st 5 Columns):
[[6.2640e+02 0.0000e+00 8.7000e+00 4.7165e+03 4.0927e+03]
 [4.1054e+03 4.7318e+03 4.7231e+03 1.5300e+01 6.3910e+02]
 [0.0000e+00 2.1330e+02 0.0000e+00 5.0020e+03 5.0020e+03]
 [5.0020e+03 4.7887e+03 5.0020e+03 0.0000e+00 0.0000e+00]
 [1.6680e+02 0.0000e+00 1.1000e+01 1.0140e+03 8.5880e+02]
 [8.6440e+02 1.0313e+03 1.0202e+03 1.7300e+01 1.7250e+02]
 [7.9210e+02 7.1600e+02 1.1650e+03 3.8000e+00 3.7370e+02]
 [3.7500e+02 4.5110e+02 2.0000e+00 1.1633e+03 7.9340e+02]
 [5.3850e+02 1.2388e+03 1.8493e+03 0.0000e+00 1.3109e+03]
 [1.3107e+03 6.1050e+02 0.0000e+00 1.8493e+03 5.3840e+02]
 [6.2700e+02 3.2070e+02 1.1642e+03 2.3700e+01 5.5560e+02]
 [5.5380e+02 8.6010e+02 1.6600e+01 1.1571e+03 6.2520e+02]
 [6.6560e+02 6.0220e+02 9.8570e+02 3.4000e+00 3.2070e+02]
 [3.2190e+02 3.8530e+02 1.8000e+00 9.8410e+02 6.6680e+02]
 [2.0310e+02 1.2096e+03 9.7650e+02 2.6780e+02 1.0468e+03]
 [1.0534e+03 4.6900e+01 2.7990e+02 9.8870e+02 2.0960e+02]]
```

In [ ]: