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]:

```python
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]:

```python
# 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, 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]:

```python
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 [ ]: