

Airline Traffic Optimization

Problem:

An airline has 4 planes all starting at Los Angeles International Airport (LAX). We want to optimize the minimum total distance for the daily flight routing for the 4 planes. The planes must schedule flights to 13 other major airports: Hartsfield-Jackson Atlanta International Airport (ATL), San Francisco (SFO), New York (JFK), Denver (DEN), Dallas/Fort Worth (DFW), Chicago O'Hare (ORD), Detroit Metropolitan (DTW), Orlando (MCO), Las Vegas (LAS), Seattle (SEA), Phoenix (PHX), Minneapolis/St. Paul (MSP), and Boston (BOS). Each airport must be visited exactly once per day by one of the four planes. Furthermore, at the end of the day, the planes must return to the starting airport. Lastly, each plane must fly either 4 or 5 flights each day.

Objective:

The objective will be to minimize the total distance flown across the four planes.

Variables:

Let n be the number of airports, so in this case, let $n = 14$.

Airports = {LAX, ATL, SFO, JFK, DEN, DFW, ORD, DTW, MCO, LAS, SEA, PHX, MSP, BOS}

We will assign each airport a number, so airports = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14}

The starting airport, LAX, is 1.

Let w, x, y, z represent the four planes

Let $w_{i,j} = 1$ if plane 1 travels from airport i to airport j , 0 otherwise.

Let $x_{i,j} = 1$ if plane 1 travels from airport i to airport j , 0 otherwise.

Let $y_{i,j} = 1$ if plane 1 travels from airport i to airport j , 0 otherwise.

Let $z_{i,j} = 1$ if plane 1 travels from airport i to airport j , 0 otherwise.

Let $d_{i,j}$ be the distance from airport i to airport j .

Let t be variable to avoid subtours.
Here is the code for the variables:

```
distances = np.array([
    [0, 1946, 337, 2475, 862, 1231, 1745, 1979, 2214, 236, 954, 370,
    [1946, 0, 2139, 760, 1199, 732, 606, 594, 403, 1747, 2182, 1587,
    [337, 2139, 0, 2569, 967, 1462, 1840, 2079, 2442, 414, 679, 651,
    [2475, 760, 2569, 0, 1620, 1390, 740, 502, 944, 2247, 2421, 215,
    [862, 1199, 967, 1620, 0, 641, 888, 1120, 1547, 628, 1021, 602,
    [1231, 732, 1462, 1390, 641, 0, 802, 987, 980, 1221, 1667, 868,
    [1745, 606, 1840, 740, 888, 802, 0, 237, 989, 1520, 1733, 1413,
    [1979, 594, 2079, 502, 1120, 987, 237, 0, 957, 1778, 1931, 1670,
    [2214, 403, 2442, 944, 1547, 980, 989, 957, 0, 2043, 2553, 1842,
    [236, 1747, 414, 2247, 628, 1221, 1520, 1778, 2043, 0, 869, 255,
    [954, 2182, 679, 2421, 1021, 1667, 1733, 1931, 2553, 869, 0, 11,
    [370, 1587, 651, 2153, 602, 868, 1413, 1670, 1842, 255, 1107, 0,
    [1533, 906, 1589, 1028, 680, 853, 333, 529, 1310, 1284, 1398, 1,
    [2611, 946, 2704, 187, 1778, 1564, 847, 632, 1183, 2368, 2484,
])

n = 14

w = cp.Variable((n, n), boolean=True)
x = cp.Variable((n, n), boolean=True)
y = cp.Variable((n, n), boolean=True)
z = cp.Variable((n, n), boolean=True)
t = cp.Variable(n, nonneg = True)
```

Objective Function:

The objective function will be the distance matrix multiplied by the sum of the plane matrices.

Written Mathematically as:

$$\sum_{i=1}^n \sum_{j=1}^n d_{i,j} (w_{i,j} + x_{i,j} + y_{i,j} + z_{i,j})$$

With the code:

```
obj_func = (
    cp.sum(cp.multiply(distances, w)) +
    cp.sum(cp.multiply(distances, x)) +
    cp.sum(cp.multiply(distances, y)) +
    cp.sum(cp.multiply(distances, z))
)
```

Constraints:

The first constraint are that each airport (aside from the starting airport) must be departed from and arrived at exactly once between all 4 planes.

For all $i > 1$ (without starting airport):

$$\sum_{j=2}^n w_{i,j} + x_{i,j} + y_{i,j} + z_{i,j} = 1$$

For all $j > 1$ (without starting airport):

$$\sum_{i=2}^n w_{i,j} + x_{i,j} + y_{i,j} + z_{i,j} = 1$$

The corresponding code:

```

for i in range(1, n):
    constraints.append(cp.sum(w[:, i]) + cp.sum(x[:, i]) + cp.sum(y[:, i]) + cp.sum(z[:, i]) == 0)
    constraints.append(cp.sum(w[i, :]) + cp.sum(x[i, :]) + cp.sum(y[i, :]) + cp.sum(z[i, :]) == 0)

```

The next constraint is to ensure that no flight departs and arrives at the same airport.

$$\sum_{i=1}^n w_{i,i} = 0$$

$$\sum_{i=1}^n x_{i,i} = 0$$

$$\sum_{i=1}^n y_{i,i} = 0$$

$$\sum_{i=1}^n z_{i,i} = 0$$

The corresponding code:

```

for i in range(n):
    constraints.append(w[i, i] == 0)
    constraints.append(x[i, i] == 0)
    constraints.append(y[i, i] == 0)
    constraints.append(z[i, i] == 0)

```

The next constraint is to ensure that the number of flights arriving to each

airport is the same as the number of flights departing from each airport.

For all i :

$$\begin{aligned}\sum_{j=1}^n w_{i,j} &= \sum_{j=1}^n w_{j,i} \\ \sum_{j=1}^n x_{i,j} &= \sum_{j=1}^n x_{j,i} \\ \sum_{j=1}^n y_{i,j} &= \sum_{j=1}^n y_{j,i} \\ \sum_{j=1}^n z_{i,j} &= \sum_{j=1}^n z_{j,i}\end{aligned}$$

The corresponding code:

```
for i in range(n):
    constraints.append(cp.sum(w[:, i]) == cp.sum(w[i, :]))
    constraints.append(cp.sum(x[:, i]) == cp.sum(x[i, :]))
    constraints.append(cp.sum(y[:, i]) == cp.sum(y[i, :]))
    constraints.append(cp.sum(z[:, i]) == cp.sum(z[i, :]))
```

The next constraint is the number of flights constraint. Each plane must

fly wither 4 or 5 flights

$$4 \leq \sum_{i=1}^n \sum_{j=1}^n w_{i,j} \leq 5$$

$$4 \leq \sum_{i=1}^n \sum_{j=1}^n x_{i,j} \leq 5$$

$$4 \leq \sum_{i=1}^n \sum_{j=1}^n y_{i,j} \leq 5$$

$$4 \leq \sum_{i=1}^n \sum_{j=1}^n z_{i,j} \leq 5$$

Here is the code for those constraints:

```
constraints.append(cp.sum(w)<=5)
constraints.append(cp.sum(w)>=4)
constraints.append(cp.sum(x)<=5)
constraints.append(cp.sum(x)>=4)
constraints.append(cp.sum(y)<=5)
constraints.append(cp.sum(y)>=4)
constraints.append(cp.sum(z)<=5)
constraints.append(cp.sum(z)>=4)
```

The next constraint is the starting airport constraint.

$$\sum_{j=2}^n w_{i,j} = 1$$

$$\sum_{j=2}^n x_{i,j} = 1$$

$$\sum_{j=2}^n y_{i,j} = 1$$

$$\sum_{j=2}^n z_{i,j} = 1$$

Since the number of incoming flights must be equal to the number of departing flights (specified earlier), this ensures at some point, each plane will return to the starting airport.

Here is the corresponding code:

```
constraints.append(cp.sum(w[0,:]) == 1)
constraints.append(cp.sum(x[0,:]) == 1)
constraints.append(cp.sum(y[0,:]) == 1)
constraints.append(cp.sum(z[0,:]) == 1)
```

The final constraint is too eliminate sub tours. I used the same idea with the variable t as used in class.

For all $2 \leq i, j \leq n$, and i does not equal j:

$$t_i - t_j + (n - 1)w_{i,j} \leq n - 2$$

$$t_i - t_j + (n - 1)x_{i,j} \leq n - 2$$

$$t_i - t_j + (n - 1)y_{i,j} \leq n - 2$$

$$t_i - t_j + (n - 1)z_{i,j} \leq n - 2$$

The corresponding code:

```
for i in range(1, n):
    for j in range(1, n):
        if i != j:
            constraints.append(t[i] - t[j] + (n - 1) * w[i, j] <= n - 1)
            constraints.append(t[i] - t[j] + (n - 1) * x[i, j] <= n - 1)
            constraints.append(t[i] - t[j] + (n - 1) * y[i, j] <= n - 1)
            constraints.append(t[i] - t[j] + (n - 1) * z[i, j] <= n - 1)
```

Running the code: