Airline Fleet Optimization

Airlines X is operating its fleet of both (i) Airbus A320 and (ii) Boeing 737 from two main hubs A and B.

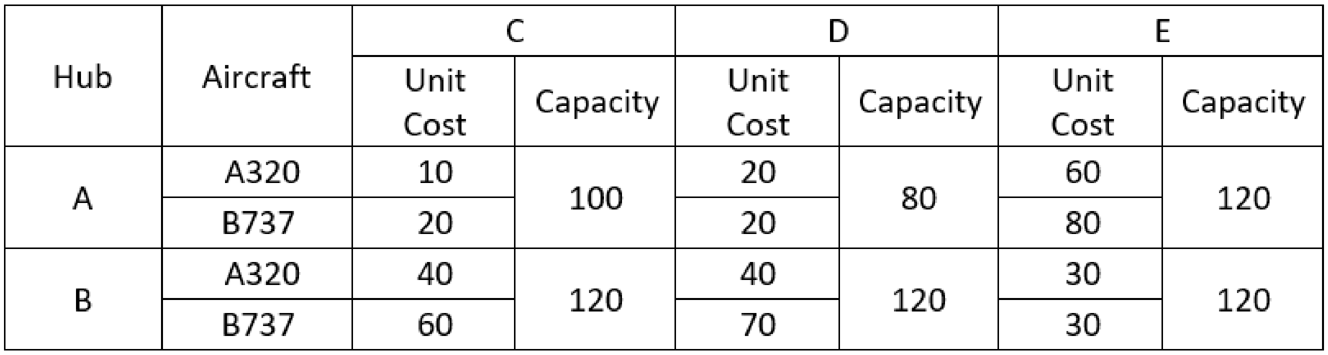
It has flight routes from these hubs to three cities in C, D, and E.

In hub A, X operates 50 units of A320 and 60 units of Boeing 737. In hub B, X has 60 units of A320 and 40 units of Boeing 737.

During one week, the number of flights for aircraft A320 that should be ready for operation is 50, 50, and 10 units in C, D, and E, respectively.

Meanwhile, the number of flights for Boeing 737 that are planned to operate is 40, 30, and 30 units in C, D, and E, respectively.

The projected operating capacities and transportation unit costs between the cities are as follows:



Hub	Aircraft	C		D		E	
		Unit Cost	Capacity	Unit Cost	Capacity	Unit Cost	Capacity
A	A320	10	100	20	80	60	120
	B737	20		20		80	
B	A320	40	120	40	120	30	120
	B737	60		70		30	

Now we should find out how to calculate the number of aircraft operated between X's hubs and the cities that X Airlines has the flight routes connected to.

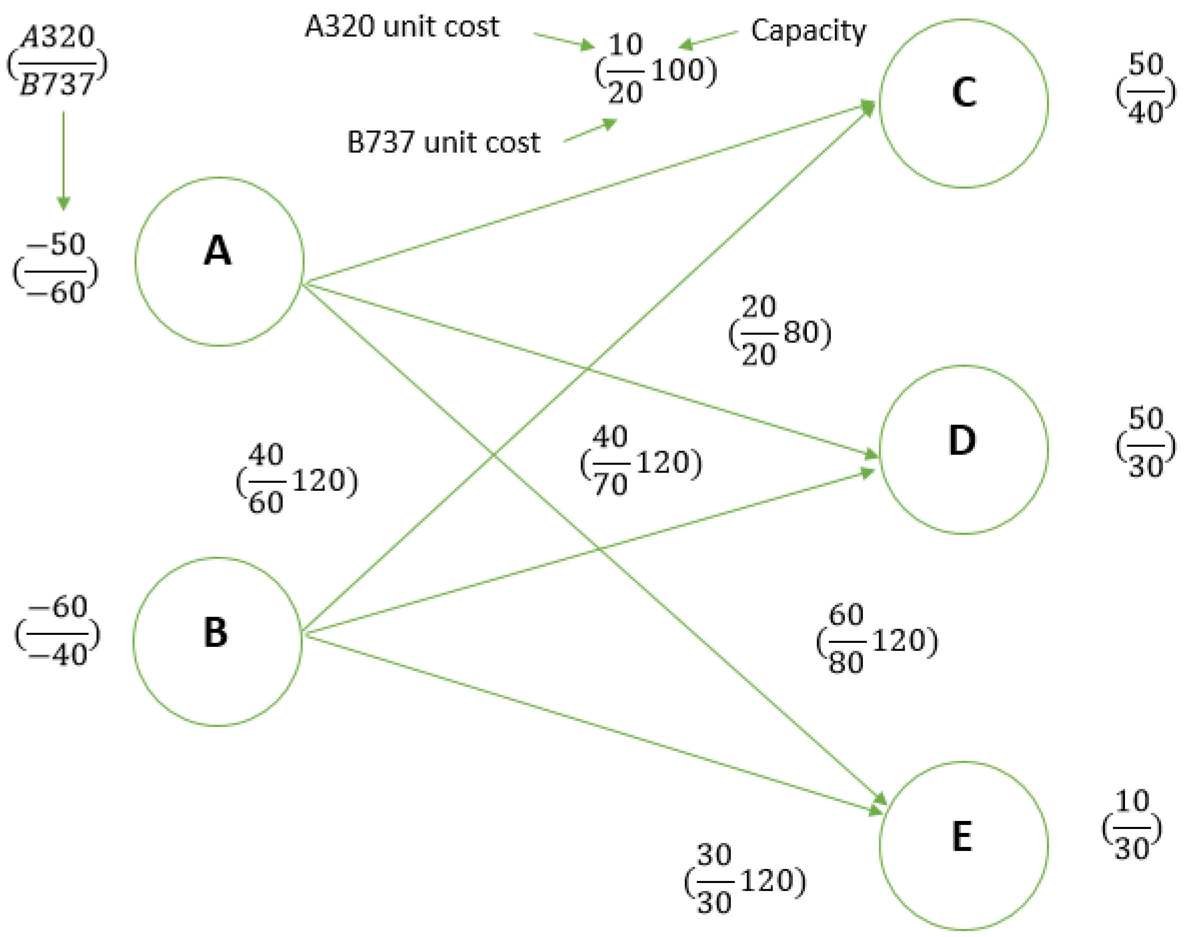
The optimal solution should minimize the transportation unit costs.

We could take network flow method to solve this problem. This is the graph describing the model.

The number of aircraft available in each hub is indicated next to each node.

For destination cities, the number of flights weekly operated is also mentioned.

The operating capacities and transportation unit costs are also provided next to each arc in the graph.



Here I use Gurobi and Python to find the solution to this optimization problem.

In [1]:
```python
import gurobipy as gp
from gurobipy import GRB

# Base data
# Fleet of X Airlines
fleet = ['A320', 'B737']
# The nodes represent the hub and connected cities.
nodes = ['A', 'B', 'C', 'D', 'E']

# Operation capacities between each hub and its connected cities for one week schedule
arcs, capacity = gp.multidict({
    ('A', 'C'): 100,
    ('A', 'D'): 80,
    ('A', 'E'): 120,
    ('B', 'C'): 120,
    ('B', 'D'): 120,
    ('B', 'E'): 120})

# Transporation unit cost for each type of aircraft between the hubs and the destination
cost = {
    ('A320', 'A', 'C'):   10,
    ('A320', 'A', 'D'): 20,
    ('A320', 'A', 'E'):  60,
    ('A320', 'B', 'C'):   40,
    ('A320', 'B', 'D'): 40,
    ('A320', 'B', 'E'):  30,
    ('B737', 'A', 'C'):   20,
    ('B737', 'A', 'D'): 20,
    ('B737', 'A', 'E'):  80,
    ('B737', 'B', 'C'):   60,
    ('B737', 'B', 'D'): 70,
    ('B737', 'B', 'E'):  30}

# Number of flights that are estimated to operate, listing as pairs of fleet aircraft :
inflow = {
    ('A320', 'A'): -50,
    ('A320', 'B'): -60,
    ('A320', 'C'): 50,
    ('A320', 'D'): 50,
    ('A320', 'E'): 10,
    ('B737', 'A'): -60,
    ('B737', 'B'): -40,
    ('B737', 'C'): 40,
    ('B737', 'D'): 30,
    ('B737', 'E'): 30}

# Create optimization model
m = gp.Model('netflow')

# Create variables
flow = m.addVars(fleet, arcs, name="flow")

# Arc-capacity constraints
# "flow.sum('*', i, j)" adds up all elements replaced by '*', i.e., in this case both A3
m.addConstrs( (flow.sum('*', i, j) <= capacity[i, j] for i, j in arcs), "cap")
m.addConstrs( (flow.sum('*', i, j) >= 0 for i, j in arcs), "cap")

# Flow balance constraints (for both aircraft types)
m.addConstrs( flow.sum(h, '*', k) - flow.sum(h, k, '*')  == inflow[h, k] for k in nodes

# Compute optimal solution
obj = sum( cost[h, i, j]*flow[h, i, j] for i, j in arcs for h in fleet )
m.setObjective(obj, GRB.MINIMIZE)
m.optimize()

# Print solution
if m.status == GRB.OPTIMAL:
    solution = m.getAttr('x', flow)
    for h in fleet:
        print('\nOptimal flows for %s:' % h)
        for i, j in arcs:
            if solution[h, i, j] > 0:
                print('\n Number of flights between %s and %s: %g' % (i, j, solution[h,

# Print optimal value of the objective function
print('\nValue of the transportation unit costs: %g' % m.objVal)
```

```
Set parameter Username
Academic license - for non-commercial use only - expires 2024-02-11
Gurobi Optimizer version 10.0.1 build v10.0.1rc0 (win64)

CPU model: Intel(R) Core(TM) i5-3210M CPU @ 2.50GHz, instruction set [SSE2|AVX]
Thread count: 2 physical cores, 4 logical processors, using up to 4 threads

Optimize a model with 22 rows, 12 columns and 48 nonzeros
Model fingerprint: 0x910fb905
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [1e+01, 8e+01]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+01, 1e+02]
Presolve removed 22 rows and 12 columns
Presolve time: 0.01s
Presolve: All rows and columns removed
Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    5.5000000e+03   0.000000e+00   2.000000e+01      0s
Extra simplex iterations after uncrush: 1
```

```
       1    5.5000000e+03   0.000000e+00   0.000000e+00      0s

Solved in 1 iterations and 0.02 seconds (0.00 work units)
Optimal objective  5.500000000e+03

Optimal flows for A320:

 Number of flights between A and C: 50

 Number of flights between B and D: 50

 Number of flights between B and E: 10

Optimal flows for B737:

 Number of flights between A and C: 30

 Number of flights between A and D: 30

 Number of flights between B and C: 10

 Number of flights between B and E: 30

Value of the transportation unit costs: 5500
```

From this model, we could adjust the values, such as the number of aircraft in the fleet and the number of flights for particular hubs or destination cities. Therefore, we would get the optimal solution subject to availability and demand.

For instance, we now change the number of fleet aircraft and demand for flights among routes.

In [2]:
```python
import gurobipy as gp
from gurobipy import GRB

# Base data
# Fleet of X Airlines
fleet = ['A320', 'B737']
# The nodes represent the hub and connected cities.
nodes = ['A', 'B', 'C', 'D', 'E']

# Operation capacities between each hub and its connected cities for one week schedule
arcs, capacity = gp.multidict({
    ('A', 'C'): 100,
    ('A', 'D'): 80,
    ('A', 'E'): 120,
    ('B', 'C'): 120,
    ('B', 'D'): 120,
    ('B', 'E'): 120})

# Transporation unit cost for each type of aircraft between the hubs and the destination
cost = {
    ('A320', 'A', 'C'):   10,
    ('A320', 'A', 'D'): 20,
    ('A320', 'A', 'E'):  60,
    ('A320', 'B', 'C'):   40,
    ('A320', 'B', 'D'): 40,
    ('A320', 'B', 'E'):  30,
    ('B737', 'A', 'C'):   20,
    ('B737', 'A', 'D'): 20,
    ('B737', 'A', 'E'):  80,
    ('B737', 'B', 'C'):   60,
    ('B737', 'B', 'D'): 70,
    ('B737', 'B', 'E'):  30}
```

```
# Number of flights that are estimated to operate, listing as pairs of fleet aircraft :
inflow = {
    ('A320', 'A'): -60,
    ('A320', 'B'): -40,
    ('A320', 'C'): 40,
    ('A320', 'D'): 50,
    ('A320', 'E'): 10,
    ('B737', 'A'): -60,
    ('B737', 'B'): -40,
    ('B737', 'C'): 40,
    ('B737', 'D'): 35,
    ('B737', 'E'): 25}

# Create optimization model
m = gp.Model('netflow')

# Create variables
flow = m.addVars(fleet, arcs, name="flow")

# Arc-capacity constraints
# "flow.sum('*', i, j)" adds up all elements replaced by '*', i.e., in this case both A3
m.addConstrs( (flow.sum('*', i, j) <= capacity[i, j] for i, j in arcs), "cap")
m.addConstrs( (flow.sum('*', i, j) >= 0 for i, j in arcs), "cap")

# Flow balance constraints (for both aircraft types)
m.addConstrs( flow.sum(h, '*', k) - flow.sum(h, k, '*')  == inflow[h, k] for k in nodes

# Compute optimal solution
obj = sum( cost[h, i, j]*flow[h, i, j] for i, j in arcs for h in fleet )
m.setObjective(obj, GRB.MINIMIZE)
m.optimize()

# Print solution
if m.status == GRB.OPTIMAL:
    solution = m.getAttr('x', flow)
    for h in fleet:
        print('\nOptimal flows for %s:' % h)
        for i, j in arcs:
            if solution[h, i, j] > 0:
                print('\n Number of flights between %s and %s: %g' % (i, j, solution[h,

# Print optimal value of the objective function
print('\nValue of the transportation unit costs: %g' % m.objVal)
```

```
Gurobi Optimizer version 10.0.1 build v10.0.1rc0 (win64)

CPU model: Intel(R) Core(TM) i5-3210M CPU @ 2.50GHz, instruction set [SSE2|AVX]
Thread count: 2 physical cores, 4 logical processors, using up to 4 threads

Optimize a model with 22 rows, 12 columns and 48 nonzeros
Model fingerprint: 0x8a63cfbd
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [1e+01, 8e+01]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+01, 1e+02]
Presolve removed 20 rows and 7 columns
Presolve time: 0.01s
Presolved: 2 rows, 5 columns, 5 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    3.7984500e+03   3.757750e+00   0.000000e+00      0s
       2    5.1500000e+03   0.000000e+00   0.000000e+00      0s

Solved in 2 iterations and 0.03 seconds (0.00 work units)
Optimal objective  5.150000000e+03
Optimal flows for A320:

 Number of flights between A and C: 40

 Number of flights between A and D: 20

 Number of flights between B and D: 30

 Number of flights between B and E: 10

Optimal flows for B737:

 Number of flights between A and C: 25

 Number of flights between A and D: 35

 Number of flights between B and C: 15

 Number of flights between B and E: 25

Value of the transportation unit costs: 5150
```

The model could be taken for more complex problems, based on the different scenarios and requirements. This is an example that applies Python and Gurobi to flight planning and fleet assignment.