Homework 3 DBA3701

March 22, 2021

Chua Shao Hwee James, A0167073A, DBA3701

1 Question 1

Consider the following optimization problem.

max $x_1 + x_2 + x_3$

s.t.

$x_1 + 2x_2 + 2x_3 \leq 15$

$2x_1 + 2x_2 + x_3 \leq 15$

$2x_1 + x_2 + 2x_3 \leq 15$

$x_1, x_2, x_3 \geq 0$

a) Dual problem

min $15p_1 + 15p_2 + 15p_3$

$p_1 + 2p_2 + 2p_3 \geq 1$

$2p_1 + p_2 + 2p_3 \geq 1$

$2p_1 + 2p_2 + p_3 \geq 1$ s.t. $p_1, p_2, p_3 \geq 0$

b) Show that optimal solution is $x_1 = x_2 = x_3 = 3$

Solving the system of linear equations from the primal

$x_1 + 2x_2 + 2x_3 = 15$

$2x_1 + 2x_2 + x_3 = 15$

$2x_1 + x_2 + 2x_3 = 15$

$x_1, x_2, x_3 = 3 \geq 0$

Hence x is primal feasible.

Solving the system of linear equations from the dual

$p_1 + 2p_2 + 2p_3 = 1$

$2p_1 + p_2 + 2p_3 = 1$

$2p_1 + 2p_2 + p_3 = 1$

$p_1, p_2, p_3 = 1/5 \geq 0$

Hence p is dual feasible.

By weak duality, if x is primal feasible, and if p is dual feasible, then if $b^T p = c^T x$, then x and p are optimal.

$b^T p = 15p_1 + 15p_2 + 15p_3 = 9$ $c^T x = x_1 + x_2 + x3 = 9$

Therefore, the optimal solution is $x_1 + x_2 + x3 = 9$

(c) What are the shadow prices associated with the first three constraints.

The shadow prices are $p_1, p_2, p_3 = 1/5$

2 Question 2

a) Formulate a linear program to maximum the daily profit of the company

Decision variables:

N_i: The number of product i to production, where i = product one and two.

Constants:

L_i: The labour cost for one unit of product i

T_i: The testing hours for one unit of product i

Constraints: $\sum_{i \in products} L_i N_i \leq 90$

$\sum_{i \in products} T_i N_i \leq 90$

$L_1 \geq 200$

$L_2 \geq 100$

Objective function: $\max\ (7-3)N_1 + (9-1)N_2 = \max\ 4N_1 + 8N_2$

b) Provide the python code for the shadow prices of assembly labor and testing

[1]:
```
from gurobipy import *
import numpy as np

def qn2():


    # Preparing an optimization model
    model = Model("qn2")

    # Declaring variables
    N_1 = model.addVar(name='N_1')
    N_2 = model.addVar(name='N_2')

    # Declare constants
    L_1 = 1/5
    L_2 = 3/10
    T_1 = 1/7
    T_2 = 3/7


    # Adding constraints
    model.addConstr(L_1 * N_1 + L_2 * N_2 <= 90, name="Labour constraints")
    model.addConstr(T_1 * N_1 + T_2 * N_2 <= 90, name="Testing constraints")

    model.addConstr(N_1 >= 200, name="Item number 1 constraints")
    model.addConstr(N_2 >= 100, name="Item number 2 constraints")

    model.setObjective(4* N_1 + 8*N_2, GRB.MAXIMIZE)


    model.optimize()
    # print optimal solutions
    print("Primal Solutions:")
    for v in model.getVars():
        print('%s %g' % (v.varName, v.x))


    print("Dual Solutions:")
    for d in model.getConstrs():
        print('%s %g' % (d.ConstrName, d.Pi))

qn2()
```

```
Academic license - for non-commercial use only - expires 2021-03-22
Using license file /Users/jameschua/gurobi.lic
Gurobi Optimizer version 9.1.1 build v9.1.1rc0 (mac64)
Thread count: 6 physical cores, 12 logical processors, using up to 12 threads
Optimize a model with 4 rows, 2 columns and 6 nonzeros
Model fingerprint: 0xb31a1108
Coefficient statistics:
  Matrix range     [1e-01, 1e+00]
  Objective range  [4e+00, 8e+00]
  Bounds range     [0e+00, 0e+00]
  RHS range        [9e+01, 2e+02]
Presolve removed 2 rows and 0 columns
Presolve time: 0.00s
Presolved: 2 rows, 2 columns, 4 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    2.1333333e+03   2.159450e+01   0.000000e+00      0s
       2    2.0400000e+03   0.000000e+00   0.000000e+00      0s
```

```
Solved in 2 iterations and 0.01 seconds
Optimal objective  2.040000000e+03
Primal Solutions:
N_1 270
N_2 120
Dual Solutions:
Labour constraints 13.3333
Testing constraints 9.33333
Item number 1 constraints 0
Item number 2 constraints 0
```

Qn 2b answer: Shadow price for labour is 13.33, testing is 9.33

2.1 Qn 3

Formulate the LOP and solve for an assignment that minimizes total race time

Decision variables: S_{ij} Whether or not swimmer i should swim for style j

Constants: T_{ij} Time for swimmer i to swim for style j

Objective function: min $\sum_i \sum_j T_{ij} S_{ij}$

Constraints:

$\sum_{i \in swimmers} S_{ij} = 1$ for all j $\in$ styles. This indicates there is only one swimmer for each style.

$\sum_{j \in styles} S_{ij} = 1$ for all i $\in$ swimmers. This indicates there is only each swimmer must swim at least one syle.

Sij are all binary

[2]:
```python
from gurobipy import *
import numpy as np
def qn3():
    # Preparing an optimization model
    model = Model("qn3")

    # Declaring variables
    S = model.addMVar((4,4), name='S', vtype=GRB.BINARY)

    # Adding style matrix
    # Use float('inf') to represent N.A
    style_mat = np.array([[48.3, 9999999, 66.0, 55.4],
                          [50.2, 53.2, 58.9, 56.2],
                          [47.5, 49.8, 9999999, 53.0],
                          [48.1, 51.8, 64.4, 54.7]])

    for style in [0, 1,2,3]:
        model.addConstr(S[:,style].sum() == 1, name="style constraints")
    for swimmer in [0, 1,2,3]:
        model.addConstr(S[swimmer,:].sum() == 1, name="swimmer constraints")
    # Gurobi can't do 4x4 mul 4x4 matrix with MVars, so we need to do this␣
↪instead
    obj = sum(S[swimmer,:] @ style_mat[swimmer,:] for swimmer in [0,1,2,3])
    model.setObjective(obj, GRB.MINIMIZE)


    model.optimize()
    # print optimal solutions
    print("Primal Solutions:")

    results = []
    for v in model.getVars():
        results.append(v.x)
        # print('%s %g' % (v.varName, v.x))

    print("Results")
    print(np.array(results).reshape((4,4)))
qn3()
```

```
Gurobi Optimizer version 9.1.1 build v9.1.1rc0 (mac64)
Thread count: 6 physical cores, 12 logical processors, using up to 12 threads
Optimize a model with 8 rows, 16 columns and 32 nonzeros
Model fingerprint: 0xccf4dcdb
Variable types: 0 continuous, 16 integer (16 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [5e+01, 1e+07]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+00]
Found heuristic solution: objective 1.000016e+07
Presolve time: 0.00s
Presolved: 8 rows, 16 columns, 32 nonzeros
Variable types: 0 continuous, 16 integer (16 binary)

Root relaxation: objective 2.117000e+02, 7 iterations, 0.00 seconds

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

*    0     0               0     211.7000000  211.70000  0.00%     -    0s

Explored 0 nodes (7 simplex iterations) in 0.01 seconds
Thread count was 12 (of 12 available processors)

Solution count 2: 211.7 1.00002e+07
```

```
Optimal solution found (tolerance 1.00e-04)
Best objective 2.117000000000e+02, best bound 2.117000000000e+02, gap 0.0000%
Primal Solutions:
Results
[[ 1. -0. -0. -0.]
 [-0. -0.  1. -0.]
 [-0.  1. -0.  0.]
 [ 0. -0. -0.  1.]]
```

Qn 3 Answer: Swimmer 1 should swim for FreeStyle. Swimmer 2 should swim for BreastStroke. Swimmer 3 should swim for Butterfly. Swimmer 4 should swim for Backstroke

This gives a total time of 211.7 seconds

2.2 Qn 4

a) Maximize the total flow from Node 1 to Node 9

Constants:

C_{ij}: Capacity of arc from Node i to j

Decision variables: X_{ij}: Amount to ship from one node i to node j. X_{12}, X_{13}, X_{15}, X_{23}, X_{26}, X_{34}, X_{47}, X_{49}, X_{56}, X_{57}, X_{69}, X_{79},

Objective function: max $flow$

Subject to:

Flow constraints:

Node 1 (source): $flow = X_{12} + X_{13} + X_{15}$

Node 9 (sink) :$flow = X_{69} + X_{79} + X_{49}$

Other nodes: $\sum_{a \in nodes} X_{aj} = \sum_{b \in nodes} X_{jb}$ for all j = 2 … 8

Capacity Constraints $0 \leq X_{ij} \leq C_{ij}$

```
[3]: from gurobipy import *
     import numpy as np
     def qn4():
         # Preparing an optimization model
         model = Model("qn3")
         nodes = [1,2,3,4,5,6,7,8,9]
         intermediate_nodes = [2,3,4,5,6,7,8]

         arcs, arcs_caps_dict = multidict({(1,2): 9, (1,3): 5, (1,5): 20, (2,3): 18,␣
      ↪(2,6): 20, (2,7): 15,
                                 (3,4): 15, (4,7): 5, (4,9): 16, (5,6): 14, (5,7): 5,␣
      ↪(6,9): 6,
                                 (7,9): 8
                                 })
```

```
    variables = model.addVars(arcs, ub = arcs_caps_dict)

    for a in arcs.select('*', 2):
        print(a)
    for a in (arcs.select(2, '*')):
        print(a)


    # Add flow constraints
    for node in intermediate_nodes: # Don't include first and last nodes!
        lhs = [variables[a] for a in arcs.select('*', node)]
        rhs = [variables[b] for b in arcs.select(node, '*')]
        model.addConstr(sum(lhs) == sum(rhs))

    flow = sum(variables[a] for a in arcs.select(1,'*'))
    model.addConstr(flow == sum(variables[a] for a in arcs.select('*', 9))) #␣
↪All nodes that end with 9.

    # Add capacity constraints
    model.setObjective(flow, GRB.MAXIMIZE)


    model.optimize()
    # print optimal solutions
    print("Solutions for each arc:")

    #printing the optimal solutions obtained
    p = model.getAttr('x', variables)
    print("Max Flow:")
    for arc in arcs:
        print("Flow for arc %s: %g" % (arc , p[arc]))

qn4()
```

```
(1, 2)
(2, 3)
(2, 6)
(2, 7)
Gurobi Optimizer version 9.1.1 build v9.1.1rc0 (mac64)
Thread count: 6 physical cores, 12 logical processors, using up to 12 threads
Optimize a model with 8 rows, 13 columns and 26 nonzeros
Model fingerprint: 0x31ebc274
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [5e+00, 2e+01]
  RHS range        [0e+00, 0e+00]
Presolve removed 2 rows and 2 columns
Presolve time: 0.00s
Presolved: 6 rows, 11 columns, 22 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    2.5000000e+01   2.400000e+01   0.000000e+00      0s
       4    2.5000000e+01   0.000000e+00   0.000000e+00      0s

Solved in 4 iterations and 0.01 seconds
Optimal objective  2.500000000e+01
Solutions for each arc:
Max Flow:
Flow for arc (1, 2): 9
Flow for arc (1, 3): 5
Flow for arc (1, 5): 11
Flow for arc (2, 3): 6
Flow for arc (2, 6): 0
Flow for arc (2, 7): 3
Flow for arc (3, 4): 11
Flow for arc (4, 7): 0
Flow for arc (4, 9): 11
Flow for arc (5, 6): 6
Flow for arc (5, 7): 5
Flow for arc (6, 9): 6
Flow for arc (7, 9): 8
```

b) Find the shortest path from Node 1 to Node 9

[4]:
```
def qn4b():
    model = Model('qn3b')
    nodes = [1,2,3,4,5,6,7,8,9]
    intermediate_nodes = [2,3,4,5,6,7,8]

    arcs, arcs_cost_dict = multidict({(1,2): 7, (1,3): 7, (1,5): 8, (2,3): 5,␣
↪(2,6): 10, (2,7): 8,
                        (3,4): 12, (4,7): 4, (4,9): 7, (5,6): 11, (5,7): 12,␣
↪(6,9): 13,
                        (7,9): 2
                        })

    variables = model.addVars(arcs)

    model.setObjective(quicksum(arcs_cost_dict[arc]*variables[arc] for arc in␣
↪arcs), GRB.MINIMIZE)

    # Add flow constraints
    model.addConstrs(quicksum(variables[arc] for arc in arcs.select('*',i)) == ␣
↪quicksum(variables[arc] for arc in arcs.select(i,'*'))
                      for i in intermediate_nodes)
    model.addConstr(quicksum(variables[arc] for arc in arcs.select(1,'*')) -␣
↪quicksum(variables[arc] for arc in arcs.select('*',1))
                     == 1)
    model.addConstr(quicksum(variables[arc] for arc in arcs.select(9,'*')) -␣
↪quicksum(variables[arc] for arc in arcs.select('*',9))
                     == -1)

    model.optimize()

    print("\nOptimal Value: \t%g\n" % model.objVal)

    #printing the optimal solutions obtained
    p = model.getAttr('x', variables)
    print("Shortest Path:")
    for arc in arcs:
        print("Flow on arc %s: %g" % (arc , p[arc]))
    for arc in arcs:
        if p[arc] == 1:
            print(fr"Travel from Node{arc[0]} to Node{arc[1]}")
qn4b()
```

```
Gurobi Optimizer version 9.1.1 build v9.1.1rc0 (mac64)
Thread count: 6 physical cores, 12 logical processors, using up to 12 threads
Optimize a model with 9 rows, 13 columns and 26 nonzeros
Model fingerprint: 0xc2deac67
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [2e+00, 1e+01]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+00, 1e+00]
Presolve removed 6 rows and 5 columns
Presolve time: 0.00s
Presolved: 3 rows, 8 columns, 16 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    7.0000000e+00   2.000000e+00   0.000000e+00      0s
       1    1.7000000e+01   0.000000e+00   0.000000e+00      0s

Solved in 1 iterations and 0.01 seconds
Optimal objective  1.700000000e+01

Optimal Value:  17

Shortest Path:
```

```
Flow on arc (1, 2): 1
Flow on arc (1, 3): 0
Flow on arc (1, 5): 0
Flow on arc (2, 3): 0
Flow on arc (2, 6): 0
Flow on arc (2, 7): 1
Flow on arc (3, 4): 0
Flow on arc (4, 7): 0
Flow on arc (4, 9): 0
Flow on arc (5, 6): 0
Flow on arc (5, 7): 0
Flow on arc (6, 9): 0
Flow on arc (7, 9): 1
Travel from Node1 to Node2
Travel from Node2 to Node7
Travel from Node7 to Node9
```

Therefore the solution is to

1. Travel from Node1 to Node2
2. Travel from Node2 to Node7
3. Travel from Node7 to Node9

c) Find the minimum cost flows to satisfy the demands at sink nodes 5,7,9, where the source node is 1 and demands at sink nodes 5,7,9 are respectivity 10,15,5

Decision variables:

X_{ij}: Flow from node i to j

Constants:

C_{ij}: Max capacity of the arc from node i to j

P_{ij}: Price of transporting 1 unit of flow from node i to j

D_i: Supply or Demand of the node. Supply is +ve, demand is -ve per convention.

Objective function:

min $\sum_{i \in nodes} \sum_{j \in nodes} P_{ij} X_{ij}$

Subject to constraints:

$\sum_{k \in nodes} X_{ik} - \sum_{j \in nodes} X_{ji} = D_i$ for all i $\in$ nodes

$0 \leq X_{ij} \leq C_{ij}$ for all i, j $\in$ nodes

[5]:
```python
def qn4c():
    model = Model('mincost')

    nodes = [1,2,3,4,5,6,7,8,9]
    nodes, demand = multidict(zip(nodes, [30, 0, 0, 0, -10, 0, -15, 0, -5]))
    arcs, arcs_cost_dict = multidict({(1,2): 7, (1,3): 7, (1,5): 8, (2,3): 5,␣
↪(2,6): 10, (2,7): 8,
                         (3,4): 12, (4,7): 4, (4,9): 7, (5,6): 11, (5,7): 12,␣
↪(6,9): 13,
                         (7,9): 2
                         })
    _, arcs_caps_dict = multidict({(1,2): 9, (1,3): 5, (1,5): 20, (2,3): 18,␣
↪(2,6): 20, (2,7): 15,
                         (3,4): 15, (4,7): 5, (4,9): 16, (5,6): 14, (5,7): 5,␣
↪(6,9): 6,
                         (7,9): 8
                         })

    #defining decision variables
    x = model.addVars(arcs, ub = arcs_caps_dict)

    model.setObjective(quicksum(arcs_cost_dict[arc] * x[arc] for arc in arcs),␣
↪GRB.MINIMIZE)

    model.addConstrs(quicksum(x[arc] for arc in arcs.select(i,'*')) -␣
↪quicksum(x[arc] for arc in arcs.select('*',i))
                     == demand[i] for i in nodes)

    model.optimize()

    print(f"Minimum Cost: {model.objVal}" )

    #printing the optimal solutions obtained
    p = model.getAttr('x', x)
    for arc in arcs:
        print(f"Flow on arc {arc}: {p[arc]}")
qn4c()
```

```
Gurobi Optimizer version 9.1.1 build v9.1.1rc0 (mac64)
Thread count: 6 physical cores, 12 logical processors, using up to 12 threads
Optimize a model with 9 rows, 13 columns and 26 nonzeros
Model fingerprint: 0x85d4f4b5
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [2e+00, 1e+01]
  Bounds range     [5e+00, 2e+01]
  RHS range        [5e+00, 3e+01]
Presolve removed 3 rows and 2 columns
Presolve time: 0.01s
Presolved: 6 rows, 11 columns, 22 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
```

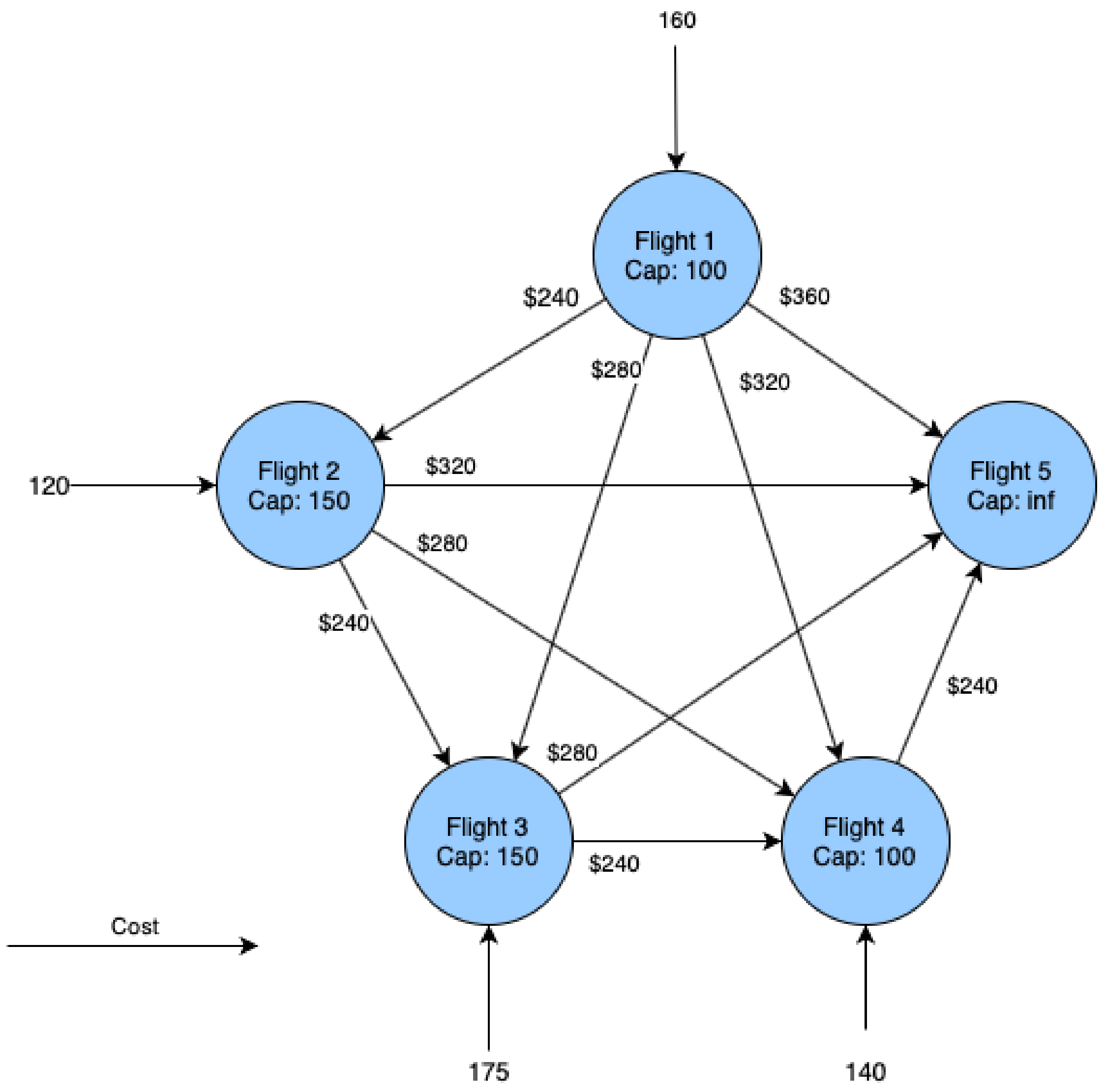
```
       0    3.0581600e+02   3.605700e+01   0.000000e+00      0s
       7    4.7000000e+02   0.000000e+00   0.000000e+00      0s

Solved in 7 iterations and 0.01 seconds
Optimal objective  4.700000000e+02
Minimum Cost: 470.0
Flow on arc (1, 2): 9.0
Flow on arc (1, 3): 5.0
Flow on arc (1, 5): 16.0
Flow on arc (2, 3): 0.0
Flow on arc (2, 6): 0.0
Flow on arc (2, 7): 9.0
Flow on arc (3, 4): 5.0
Flow on arc (4, 7): 5.0
Flow on arc (4, 9): 0.0
Flow on arc (5, 6): 1.0
Flow on arc (5, 7): 5.0
Flow on arc (6, 9): 1.0
Flow on arc (7, 9): 4.0
```

2.3 Question 5

United Airlines has five daily flights from Boston (BOS) to San Fransisco (SFO), every two hours starting at 7am. The capacities of these flights are respectively 100, 150, 150, 100, and ∞. Passengers suffering from over booking are diverted to later flights. Delayed passengers get \$200 plus \$20 for every hour of delay. Suppose that today the first four flight have 160, 120, 175 and 140 confirmed reservation, the airline need to determine the most economical passenger routing strategy

a) Draw the network flows for this problem.



b) Formulate the mathematical optimization problem to determine the most economical passenger routing strategy.

Decision variables:

X_{ij} : Number of passengers to route from flight i to flight j$

Constants:

P_{ij}: Price of routing one passenger from flight i to flight j

C_i: Capacity of flight i

R_i: Passengers originally reserved on flight i

Objective function:

$min 240(X_{12} + X_{23} + X_{34} + X_{45}) + 280(X_{13} + X_{24} + X_{3_5}) + 320(X_{14} + X_{25}) + 360X_{15} $

Subject to constraints:

$R_i + \sum_{j \in flights} X_{ji} - \sum_{j \in flights} X_{ij} \leq C_i$ for all i in flights 1,2,3,4,5

$X_{ij} \geq 0$

X_{ij} are integers

[9]:
```
def qn5b():
    model = Model('routing')

    nodes = [1,2,3,4,5]
    arcs, arcs_cost_dict = multidict({(1,2): 240, (1,3): 280, (1,4): 320, (1,5):
 ↪ 360,
                                       (2,3): 240, (2,4): 280, (2,5): 320, (3,4):
 ↪ 240, (3,5): 280, (4,5): 240
                                       })
    _, nodes_cap_dict = multidict({1: 100, 2: 150, 3: 150, 4: 100, 5: np.Inf})
    reserved = {1: 160, 2: 120, 3: 175, 4: 140, 5: 0}
    # reserved = {1: 0, 2: 0, 3: 151, 4: 0, 5: 0}

    #defining decision variables
    x = model.addVars(arcs)

    model.setObjective(quicksum(x[arc]*arcs_cost_dict[arc] for arc in arcs),␣
 ↪GRB.MINIMIZE)

    model.addConstrs(reserved[i] + quicksum(x.select("*",i)) - quicksum(x.
 ↪select(i, "*")) <= nodes_cap_dict[i] for i in nodes )


    model.optimize()

    print(f"Minimum Cost: {model.objVal}" )

    #printing the optimal solutions obtained
    p = model.getAttr('x', x)
    for arc in arcs:
        if p[arc] > 0:
            print(f"Route {p[arc]} passengers from Flight {arc[0]} to {arc[1]}")
qn5b()
```

```
Gurobi Optimizer version 9.1.1 build v9.1.1rc0 (mac64)
Thread count: 6 physical cores, 12 logical processors, using up to 12 threads
Optimize a model with 5 rows, 10 columns and 20 nonzeros
Model fingerprint: 0x44b883f1
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [2e+02, 4e+02]
  Bounds range     [0e+00, 0e+00]
  RHS range        [2e+01, 6e+01]
Presolve removed 3 rows and 7 columns
Presolve time: 0.00s
Presolved: 2 rows, 3 columns, 4 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    1.6600000e+04   6.000000e+01   0.000000e+00      0s
       2    3.4600000e+04   0.000000e+00   0.000000e+00      0s

Solved in 2 iterations and 0.01 seconds
Optimal objective  3.460000000e+04
Minimum Cost: 34600.0
Route 30.0 passengers from Flight 1 to 2
Route 30.0 passengers from Flight 1 to 5
Route 25.0 passengers from Flight 3 to 5
Route 40.0 passengers from Flight 4 to 5
```