appendix4

December 5, 2022

1 Appendix 4 - Updated Formulation with Cargo Arrivals as Decision Variables

[ ]:
```
# import gurobi
from gurobipy import *
import numpy as np

# number of airplanes
planes = 1200

# number of days
days = 5

# number of airports
noairports = 3

# list of airports
airports = ['A', 'B', 'C']

# origin-destination pairs
odpairs = ['AB', 'AC', 'BA', 'BC', 'CA', 'CB']

# number of origin-destination pairs
pairs = 6

# total cargo amounts across schedule for the week
total_cargo = np.array([1100, 250, 125, 125, 200, 1500])

# holding costs
holdingcost = 10

# repositioning costs
ABcost = 7
BCcost = 6
ACcost = 3

# create new model
```

```
myModel = Model("Cargo_Operations")

# create decision variables for cargo supply
xVars = [[0 for i in range(days)] for j in range(pairs)]

for i in range(pairs):
    for j in range(days):
        curVar = myModel.addVar(vtype = GRB.INTEGER, name = "x" + odpairs[i] +␣
 ↪str(j+1))
        xVars[i][j] = curVar

# create decision variables for airplane shipments
yVars = [[0 for i in range(days)] for j in range(pairs)]

for i in range(pairs):
    for j in range(days):
        curVar = myModel.addVar(vtype = GRB.INTEGER, name = "y" + odpairs[i] +␣
 ↪str(j+1))
        yVars[i][j] = curVar

# create decision variables for airplane repositioning
zVars = [[0 for i in range(days)] for j in range(pairs)]

for i in range(pairs):
    for j in range(days):
        curVar = myModel.addVar(vtype = GRB.INTEGER, name = "z" + odpairs[i] +␣
 ↪str(j+1))
        zVars[i][j] = curVar

# create decision variables for airplanes that are grounded at the airport
sVars = [[0 for i in range(days)] for j in range(noairports)]

for i in range(noairports):
    for j in range(days):
        curVar = myModel.addVar(vtype = GRB.INTEGER, name = "s" + airports[i] +␣
 ↪str(j+1))
        sVars[i][j] = curVar

# create decision variables for cargo amounts
cargoVars = [[0 for i in range(days)] for j in range(pairs)]

for i in range(pairs):
    for j in range(days):
        curVar = myModel.addVar(vtype = GRB.INTEGER, name = "c" + odpairs[i] +␣
 ↪str(j+1))
        cargoVars[i][j] = curVar
```

```
# integrate decision variables into the model
myModel.update()

# create a linear expression for the objective
objExpr = LinExpr()

# holding costs (number of cargo available minus number actually shipped)
for i in range(pairs):
    for j in range(days):
        curVar1 = xVars[i][j]
        curVar2 = yVars[i][j]
        objExpr += holdingcost * (curVar1 - curVar2)

# repositioning costs for A to B
for j in range(days):
    curVar = zVars[0][j]
    objExpr += ABcost * curVar

# repositioning costs for A to C
for j in range(days):
    curVar = zVars[1][j]
    objExpr += ACcost * curVar

# repositioning costs for B to A
for j in range(days):
    curVar = zVars[2][j]
    objExpr += ABcost * curVar

# repositioning costs for B to C
for j in range(days):
    curVar = zVars[3][j]
    objExpr += BCcost * curVar

# repositioning costs for C to A
for j in range(days):
    curVar = zVars[4][j]
    objExpr += ACcost * curVar

# repositioning costs for C to B
for j in range(days):
    curVar = zVars[5][j]
    objExpr += BCcost * curVar

myModel.setObjective(objExpr, GRB.MINIMIZE)

# create constraints for flow of cargo
for i in range(pairs):
```

```
    for j in range(days):
        constExpr = LinExpr()
        xVar1 = xVars[i][j]
        yVar1 = yVars[i][j]

        # last day should have zero left-over in cargo
        if j != 4:
            xVar2 = xVars[i][j+1]
            constExpr += xVar1 + cargoVars[i][j+1] - yVar1
            myModel.addConstr(lhs = constExpr, sense=GRB.EQUAL, rhs = xVar2,␣
 ↪name = "CargoSupplyConstraints")
        else:
            constExpr += xVar1 - yVar1
            myModel.addConstr(lhs = constExpr, sense=GRB.EQUAL, rhs = 0, name =␣
 ↪"CargoSupplyConstraints")

# create constraints to initialize initial demand of cargo
for i in range(pairs):
    constExpr = LinExpr()
    curVar = xVars[i][0]
    constExpr += curVar
    myModel.addConstr(lhs = constExpr, sense = GRB.EQUAL, rhs =␣
 ↪cargoVars[i][0], name = "InitialDemand")

# create constraints for the total amount of cargo shipped across␣
 ↪origin-destination pairs for all days
for i in range(pairs):
    constExpr = LinExpr()
    for j in range(days):
        curVar = cargoVars[i][j]
        constExpr += curVar
    myModel.addConstr(lhs = constExpr, sense=GRB.EQUAL, rhs = total_cargo[i],␣
 ↪name = "TotalCargo")

# create constraints to ensure shipment doesn't exceed available supply
for i in range(pairs):
    for j in range(days):
        constExpr = LinExpr()
        xVar1 = xVars[i][j]
        yVar1 = yVars[i][j]
        constExpr += yVar1
        myModel.addConstr(lhs = constExpr, sense=GRB.LESS_EQUAL, rhs =xVar1,␣
 ↪name = "ShipmentConstraints")

# create constraints for flow of airplanes for airport A
for i in range(days):
```

```
    constExpr = LinExpr()
    # last day should reset the number of airplanes for following week
    if i != 4:
        constExpr = sVars[0][i] + yVars[2][i] + yVars[4][i] + zVars[2][i] +␣
 ↪zVars[4][i] - sVars[0][i+1] - yVars[0][i+1] - yVars[1][i+1] - zVars[0][i+1]␣
 ↪- zVars[1][i+1]
    else:
        constExpr = sVars[0][i] + yVars[2][i] + yVars[4][i] + zVars[2][i] +␣
 ↪zVars[4][i] - sVars[0][0] - yVars[0][0] - yVars[1][0] - zVars[0][0] -␣
 ↪zVars[1][0]
    myModel.addConstr(lhs = constExpr, sense=GRB.EQUAL, rhs = 0, name =␣
 ↪"AirportAConstraints")

# create constraints for flow of airplanes for airport B
for i in range(days):
    constExpr = LinExpr()
    # last day should reset the number of airplanes for following week
    if i != 4:
        constExpr = sVars[1][i] + yVars[0][i] + yVars[5][i] + zVars[0][i] +␣
 ↪zVars[5][i] - sVars[1][i+1] - yVars[2][i+1] - yVars[3][i+1] - zVars[2][i+1]␣
 ↪- zVars[3][i+1]
    else:
        constExpr = sVars[1][i] + yVars[0][i] + yVars[5][i] + zVars[0][i] +␣
 ↪zVars[5][i] - sVars[1][0] - yVars[2][0] - yVars[3][0] - zVars[2][0] -␣
 ↪zVars[3][0]
    myModel.addConstr(lhs = constExpr, sense=GRB.EQUAL, rhs = 0, name =␣
 ↪"AirportBConstraints")

# create constraints for flow of airplanes for airport C
for i in range(days):
    constExpr = LinExpr()
    # last day should reset the number of airplanes for following week
    if i != 4:
        constExpr = sVars[2][i] + yVars[1][i] + yVars[3][i] + zVars[1][i] +␣
 ↪zVars[3][i] - sVars[2][i+1] - yVars[4][i+1] - yVars[5][i+1] - zVars[4][i+1]␣
 ↪- zVars[5][i+1]
    else:
        constExpr = sVars[2][i] + yVars[1][i] + yVars[3][i] + zVars[1][i] +␣
 ↪zVars[3][i] - sVars[2][0] - yVars[4][0] - yVars[5][0] - zVars[4][0] -␣
 ↪zVars[5][0]
    myModel.addConstr(lhs = constExpr, sense=GRB.EQUAL, rhs = 0, name =␣
 ↪"AirportCConstraints")

# create constraint to bound total number of planes
constExpr = LinExpr()
for i in range(pairs):
```

```
    constExpr += yVars[i][0]
    constExpr += zVars[i][0]
for i in range(noairports):
    constExpr += sVars[i][0]
myModel.addConstr(lhs = constExpr, sense=GRB.EQUAL, rhs = planes, name =␣
 ↪"TotalPlaneConstraints")

# integrate objective and constraints into the model
myModel.update()

# write the model in a file to make sure it is constructed correctly
myModel.write(filename = "CargoProject.lp")

# optimize the model
myModel.optimize()
```

```
Warning: linear constraint 0 and linear constraint 1 have the same name
"CargoSupplyConstraints"
Gurobi Optimizer version 9.5.2 build v9.5.2rc0 (win64)
Thread count: 4 physical cores, 8 logical processors, using up to 8 threads
Optimize a model with 88 rows, 135 columns and 375 nonzeros
Model fingerprint: 0x0ecbf665
Variable types: 0 continuous, 135 integer (0 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [3e+00, 1e+01]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+02, 2e+03]
Presolve removed 36 rows and 30 columns
Presolve time: 0.00s
Presolved: 52 rows, 105 columns, 303 nonzeros
Variable types: 0 continuous, 105 integer (0 binary)

Root relaxation: objective 1.512500e+04, 45 iterations, 0.00 seconds (0.00 work
units)

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

*    0     0               0    15125.000000 15125.0000  0.00%     -    0s

Explored 1 nodes (45 simplex iterations) in 0.02 seconds (0.00 work units)
Thread count was 8 (of 8 available processors)

Solution count 1: 15125

Optimal solution found (tolerance 1.00e-04)
Best objective 1.512500000000e+04, best bound 1.512500000000e+04, gap 0.0000%
```

[ ]:
```
# print optimal objective and optimal solution
print("\nOptimal Objective: " + str(myModel.ObjVal))
print("\nOptimal Solution: " )
allVars = myModel.getVars()
for curVar in allVars:
    print(curVar.varName + " " + str(curVar.x))
```

```
Optimal Objective: 15125.0

Optimal Solution:
xAB1 0.0
xAB2 0.0
xAB3 700.0
xAB4 400.0
xAB5 0.0
xAC1 0.0
xAC2 0.0
xAC3 0.0
xAC4 0.0
xAC5 250.0
xBA1 0.0
xBA2 0.0
xBA3 0.0
xBA4 125.0
xBA5 0.0
xBC1 0.0
xBC2 0.0
xBC3 0.0
xBC4 0.0
xBC5 125.0
xCA1 0.0
xCA2 0.0
xCA3 200.0
xCA4 0.0
xCA5 0.0
xCB1 650.0
xCB2 200.0
xCB3 100.0
xCB4 0.0
xCB5 550.0
yAB1 -0.0
yAB2 -0.0
yAB3 700.0
yAB4 400.0
yAB5 -0.0
yAC1 -0.0
yAC2 -0.0
```

```
yAC3 0.0
yAC4 -0.0
yAC5 250.0
yBA1 -0.0
yBA2 -0.0
yBA3 0.0
yBA4 125.0
yBA5 -0.0
yBC1 -0.0
yBC2 -0.0
yBC3 -0.0
yBC4 -0.0
yBC5 125.0
yCA1 -0.0
yCA2 -0.0
yCA3 200.0
yCA4 -0.0
yCA5 0.0
yCB1 650.0
yCB2 200.0
yCB3 100.0
yCB4 0.0
yCB5 550.0
zAB1 -0.0
zAB2 -0.0
zAB3 -0.0
zAB4 -0.0
zAB5 -0.0
zAC1 -0.0
zAC2 -0.0
zAC3 -0.0
zAC4 -0.0
zAC5 -0.0
zBA1 -0.0
zBA2 700.0
zBA3 200.0
zBA4 125.0
zBA5 -0.0
zBC1 200.0
zBC2 300.0
zBC3 -0.0
zBC4 550.0
zBC5 275.0
zCA1 -0.0
zCA2 -0.0
zCA3 -0.0
zCA4 -0.0
zCA5 -0.0
```

```
zCB1 -0.0
zCB2 -0.0
zCB3 -0.0
zCB4 -0.0
zCB5 -0.0
sA1 -0.0
sA2 0.0
sA3 -0.0
sA4 -0.0
sA5 0.0
sB1 350.0
sB2 -0.0
sB3 -0.0
sB4 -0.0
sB5 -0.0
sC1 -0.0
sC2 -0.0
sC3 -0.0
sC4 -0.0
sC5 -0.0
cAB1 0.0
cAB2 0.0
cAB3 700.0
cAB4 400.0
cAB5 0.0
cAC1 0.0
cAC2 0.0
cAC3 -0.0
cAC4 0.0
cAC5 250.0
cBA1 0.0
cBA2 0.0
cBA3 0.0
cBA4 125.0
cBA5 0.0
cBC1 0.0
cBC2 0.0
cBC3 0.0
cBC4 0.0
cBC5 125.0
cCA1 0.0
cCA2 0.0
cCA3 200.0
cCA4 0.0
cCA5 0.0
cCB1 650.0
cCB2 200.0
cCB3 100.0
```

```
cCB4 0.0
cCB5 550.0
```