Cargo Operations of Express Air Project
ORIE 5380 - Optimization Methods
James Chu (qjc2)

1. Executive Summary

Express Air is a cargo operator that is seeking to determine a weekly aircraft movement cycle that delivers all its cargo while minimizing total cost. As part of our analysis, we are unable to create an aircraft movement cycle with Express Air's current number of aircraft and the schedule in which cargo arrives into the system. However, feasible solutions can be found by either increasing the number of aircraft or redistributing the cargo arrivals. We ultimately recommend a solution that does the latter, with an associated cost of 15,125.

2. Problem Overview

Express Air is a cargo operator that delivers cargo between three airports (A, B, and C) using its aircraft fleet. There are two components that Express Air needs to consider when delivering cargo:

1) Cargo loads

   - Each day, a set amount of cargo loads arrives within the system that needs to be delivered between each origin-destination pair
   - Each load of cargo requires a single aircraft to carry
   - The total amount of cargo loads that needs to be delivered between each origin-destination pair on any given day is equal to the newly arrived cargo loads that day, plus any remaining cargo loads that was not delivered in the previous days. Express Air determines the portion of cargo loads that will be delivered on any given day, which is constrained by the number of aircraft available at an airport
   - There is a cost per day associated to holding and not delivering a cargo load

2) Aircraft Positioning

   - For each airport, Express Air has three options on how to handle the aircraft on any given day:
     1) The airport can use its aircraft to deliver cargo to its destinations
     2) The airport can ground the aircraft, and have it remain at the airport to be available for the next day
     3) The airport can reposition empty aircraft to other airports, so to help balance the cargo load delivery requirements
   - It takes one full day to travel between each airport
   - Any aircraft that's repositioned or used to deliver cargo will be available for use by the destination airport the following day
   - There is a cost associated to repositioning an aircraft for each origin-destination pair

Since there's a consistent weekly pattern for cargo loads that needs to be delivered, Express Air wants to find a repeatable weekly movement cycle for the aircraft. This implies that the aircraft that moves into an airport on Friday evening should be equal to the aircraft that's available for the same airport for the following Monday morning.

Furthermore, since the amount of cargo that needs to be delivered each week is fixed, we cannot optimize the revenue and costs of delivering cargo. Instead, we are tasked with minimizing the cost to reposition empty aircraft and the cost to hold cargo, while ensuring all cargo is delivered by the end of the week and that we have a repeatable weekly aircraft movement cycle.

3. Data Description

The number of aircraft that Express Air has available to distribute across its three airports is 1,200. The amount of cargo loads arriving into the system that needs to be delivered between origin destination pairs for each day is given by Table 1 below:

Origin-Destination	Monday	Tuesday	Wednesday	Thursday	Friday
A-B	100	200	100	400	300
A-C	50	50	50	50	50
B-A	25	25	25	25	25
B-C	25	25	25	25	25
C-A	40	40	40	40	40
C-B	400	200	300	200	400

Table 1: Amount of cargo loads arriving into the system on each day that needs to be delivered between each origin-destination pair

The cost to hold one load of cargo is 10 per day, while the cost to reposition empty aircraft between airports is given by Figure 1 below:

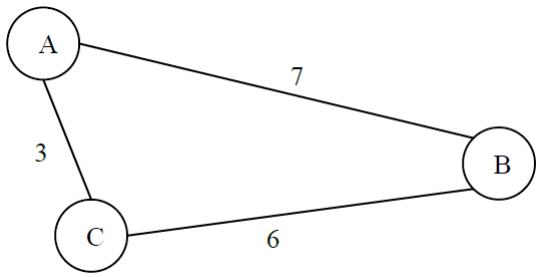

Figure 1: Repositioning costs among different airports

4. Overview of Optimization Model and Mathematical Details

To solve for Express Air's weekly movement pattern, we establish several decision variables for our optimization problem:

- Let $x_{i,j,t}$ represent the amount of cargo loads that needs to be delivered between airport i and airport j at time t
- Let $y_{i,j,t}$ represent the amount of cargo loads shipped out of airport i to airport j on day t. Since each aircraft can carry only one cargo load, this also represents the number of aircraft that is delivering cargo between airport i and airport j
- Let $z_{i,j,t}$ represent the number of aircraft repositioned from airport i to airport j on day t

- Let $s_{i,t}$ represent the number of aircraft that's grounded at airport i on day t (i.e. the aircraft does not leave the airport)

We create these decision variables $\forall i, j \in \{A, B, C\}, i \neq j, t = 1, 2, 3, 4, 5$. All decision variables are non-negative integer values.

The objective function is given by the sum of the holding costs and repositioning costs across all the days. The amount of cargo that we hold on a given day t per origin-destination pair can which can be calculated as the amount of cargo loads that needs to be delivered minus the amount that's actually delivered: $x_{i,j,t} - y_{i,j,t}$. The number of aircraft that's repositioned on a given day is just given by $z_{i,j,t}$. Therefore, the objective of our optimization problem is:

$$min \sum_{t=1}^{5} \sum_{i=\{A,B,C\}} \sum_{j=\{A,B,C\}} 10(x_{i,j,t} - y_{i,j,t}) + r_{i,j} z_{i,j,t}$$

where $r_{i,j}$ is the cost to reposition aircraft between airport i and airport j (given by Figure 1) and $i \neq j$.

We then establish the constraints of the problem:

C1) The number of cargo loads shipped out of each origin-destination pair cannot exceed the available cargo load to ship:

$$y_{i,j,t} \leq x_{i,j,t} \ \forall i, j \in \{A, B, C\}, i \neq j, t = 1, 2, 3, 4, 5$$

C2) The amount of cargo loads available to be shipped between each origin-destination pair on any given day is equal to the cargo that was held from the previous day (given by $x_{i,j,t} - y_{i,j,t}$) plus the arrival of new cargo loads into the system. This can be represented as:

$$x_{i,j,t+1} = x_{i,j,t} - y_{i,j,t} + D_{i,j,t+1} \ \forall i, j \in \{A, B, C\}, i \neq j, t = 1, 2, 3, 4$$

where $D_{i,j,t+1}$ is the cargo loads arriving into the system given by Table 1.

C3) Since we require that all cargo loads be delivered by the end of the week, we create a constraint that enforces that all cargo loads are delivered between each origin-destination pair by Friday:

$$x_{i,j,5} - y_{i,j,5} = 0 \ \forall i, j \in \{A, B, C\}, i \neq j$$

C4) At the beginning of the week, there should be no cargo that's held from the previous day. Therefore, we can directly assign $x_{i,j,1}$ equal to the cargo loads arriving into the system on Monday:

$$x_{i,j,1} = c_{i,j,1} \ \forall i, j \in \{A, B, C\}, i \neq j$$

C5) We also establish constraints for the flow of aircraft at each airport per day. The number of aircraft the flows into an airport at time t must equal the flow out of the airport at time $t+1$:

- The flow into airport i at time t is equal to number of aircraft delivering to airport i, the number of aircraft that's repositioned to airport i, and the number of aircraft that's grounded at airport i at time t:

$$s_{i,t} + \sum_{j \neq i, j \in \{A,B,C\}} y_{j,i,t} + \sum_{j \neq i, j \in \{A,B,C\}} z_{j,i,t}$$

- The flow out of airport i at time $t+1$ is equal to the number of aircraft delivering out of airport i, the number of aircraft that's positioned out of airport i, and the number of aircraft that's grounded at airport i at time $t+1$:

$$s_{i,t+1} + \sum_{j \neq i, j \in \{A,B,C\}} y_{i,j,t+1} + \sum_{j \neq i, j \in \{A,B,C\}} z_{i,j,t+1}$$

Combining the two flows above, we get the constraint:

$$s_{i,t} + \sum_{j \neq i, j \in \{A,B,C\}} y_{j,i,t} + \sum_{j \neq i, j \in \{A,B,C\}} z_{j,i,t} = s_{i,t+1} + \sum_{j \neq i, j \in \{A,B,C\}} y_{i,j,t+1} + \sum_{j \neq i, j \in \{A,B,C\}} z_{i,j,t+1}$$

$$\forall i \in \{A, B, C\}, t = 1, 2, 3, 4$$

C6) In order to get a repeatable weekly movement cycle, we also need to create a constraint where the flow into airport i at the end of Friday is the same as the flow out of airport i on Monday. Following a similar logic to 5), we obtain the following constraints:

$$s_{i,5} + \sum_{j \neq i, j \in \{A,B,C\}} y_{j,i,5} + \sum_{j \neq i, j \in \{A,B,C\}} z_{j,i,5} = s_{i,1} + \sum_{j \neq i, j \in \{A,B,C\}} y_{i,j,1} + \sum_{j \neq i, j \in \{A,B,C\}} z_{i,j,1}$$

$$\forall i \in \{A, B, C\}$$

C7) Finally, we need to ensure that the number of aircraft in the system is equal to 1,200. We do this by enforcing that the number of aircraft flowing out of day 1 is equal to 1,200. The flow constraints established by C5) will ensure that all the other days will also have 1,200 aircraft in total:

$$\sum_{i \in \{A,B,C\}} s_{i,1} + \sum_{j \neq i, i, j \in \{A,B,C\}} y_{i,j,1} + \sum_{j \neq i, i, j \in \{A,B,C\}} z_{i,j,1} = 1,200$$

5. Analysis of Results and Findings

We initially attempt use Gurobi to solve the optimization problem that we've formulated above. However, the output indicates that a solution is not feasible. We can also come to the same conclusion by considering the total number of available aircraft, and the amount of cargo loads arriving into the system for Thursday and Friday for the A-B and C-B origin-destination pairs only (see Table 2 below):

Origin-Destination	Thursday	Friday
A-B	400	300
C-B	200	400

Table 2: Amount of cargo loads arriving into the system for Thursday and Friday that needs to be delivered between A-B and C-B

Assuming that we have zero cargo that we hold from previous days, the number of aircraft needed to deliver the cargo loads for A-B and C-B over Thursday and Friday is 1,300. This is because of two reasons:

1) In order for us to have a repeatable schedule, all the cargo loads must be shipped out by Friday to their destinations. No cargo loads can roll-over to the following week

2) Any aircraft that we use to deliver cargo for A-B and C-B on Thursday will only be usable by airport B on Friday. Therefore, airport A and airport C will need additional aircraft to deliver their cargo loads arriving on Friday

To illustrate this, we can take the case where airport A has 400 aircraft and airport C has 200 aircraft on Thursday, which they use to deliver all the cargo loads they received on Thursday to airport B. As a result, on Friday, airport B will have 600 aircraft. However, on Friday, airport A and airport C still needs 300 and 400 aircraft, respectively, to deliver Friday's cargo to airport B. Since 600 aircraft is "locked" at airport B, airport A and airport C will need a total of 700 aircraft to ensure that all cargo loads are shipped out by Friday. Therefore, a minimum of 1,300 aircraft is necessary to fulfill this requirement. Since, Express Air only have 1,200 aircraft available, no feasible solution exists.

Updated Formulation with Number of Aircraft as Decision Variable

One option to find a feasible solution is to simply increase the number of aircraft to handle the existing cargo arriving into the system. We do this by modifying our original formulation by:

1) Making the number of aircraft a decision variable in our optimization problem: $aircraft$

2) Updating our seventh constraint (C7) so that the aircraft flow out of day 1 is equal to $aircraft$:

$$\sum_{i\in\{A,B,C\}} s_{i,0} + \sum_{j\neq i, i,j\in\{A,B,C\}} y_{i,j,0} + \sum_{j\neq i, i,j\in\{A,B,C\}} z_{i,j,0} = aircraft$$

3) We also update the objective function to include $aircraft$. This is because obtaining aircraft generally incurs large fixed costs, so we want to minimize the total number of aircraft in our system:

$$min\ (aircraft + \sum_{t=1}^{5} \sum_{i=\{A,B,C\}} \sum_{j=\{A,B,C\}} 10(x_{i,j,t} - y_{i,j,t}) + r_{i,j} z_{i,j,t})$$

Running this new formulation through Gurobi, we determine that a minimum of 1,390 aircraft is required to have a feasible solution. The optimal objective value is 16,515. However, since the objective function contains the $aircraft$, we subtract 1,390 from 16,515 to obtain the true cost, which is 15,125. Increasing the number of aircraft beyond 1,390 does not yield any additional benefits, as the excess aircraft would just stay grounded at each airport. As mentioned before, obtaining additional aircraft is expensive, and should only be considered if the cargo arrival schedule (i.e. Table 1) is fixed.

Updated Formulation with Weekly Rollover

Another option is to allow cargo from one week to roll-over to the next week by modifying the following constraints:

1) We remove the constraint (C4) that initializes the amount of cargo at the beginning of the week

2) Instead, we introduce a new constraint that allows cargo from Friday to rollover to following week's Monday:

$$x_{i,j,0} = x_{i,j,5} - y_{i,j,5} + D_{i,j,0}\ \forall i, j \in \{A, B, C\}, i \neq j$$

This formulation yields an objective value of 17,925.

Updated Formulation with Cargo Arrivals as Decision Variables

Alternatively, we can also find a feasible solution by redistributing the cargo arrivals between the days, while keeping the number of aircraft fixed at 1,200 and not allowing weekly rollovers. We do this by again modifying our original formulation by:

1) Converting the number of cargo loads arriving into the system into a decision variable: $c_{i,j,t}$

2) Since the total amount of cargo that needs to be delivered per week between each origin-destination pair is fixed, we add a new constraint that bounds the $c_{i,j,t}$:

$$\sum_{t=1}^{5} c_{i,j,t} = total_{i,j}$$

$$\forall j \neq i, i, j \in \{A, B, C\}$$

where $total_{i,j}$ is given by the Table 3.

Origin-Destination	Total
A-B	1100
A-C	250
B-A	125
B-C	125
C-A	200
C-B	1500

Table 3: The total amount of cargo that enters the system in a week that needs to be delivered between each origin-destination pair

Running this through Gurobi, we obtain an objective value of 15,125 and the following cargo arrival schedule:

Origin-Destination	Monday	Tuesday	Wednesday	Thursday	Friday
A-B	0	0	700	400	0
A-C	0	0	0	0	250
B-A	0	0	0	125	0
B-C	0	0	0	0	125
C-A	0	0	200	0	0
C-B	650	200	100	0	550

Table 4: Feasible schedule for cargo loads arriving into system on each day for each origin-destination pair

This is just one feasible schedule. An alternative feasible schedule that yields the same objective value is shown in Table 5, where all the cargo arrivals are spread out evenly across the days:

Origin-Destination	Monday	Tuesday	Wednesday	Thursday	Friday
A-B	220	220	220	220	220
A-C	50	50	50	50	50
B-A	25	25	25	25	25
B-C	25	25	25	25	25
C-A	40	40	40	40	40
C-B	300	300	300	300	300

Table 5: Alternate feasible schedule for cargo load arrivals, where the arrivals are spread out through the days

The choice of schedule will largely depend on factors outside the scope of this analysis, which includes the cost to have cargo arrive within the system and any limitations on the amount of cargo that can arrive at one time. If there is a cost associated with having cargo arrive into the system each day, we'd recommend the adjusting the cargo load arrival schedule to what's shown in Table 4 as there are several days where no cargo arrives into the system at all. **We also recommend this solution over all other solutions from alternative formulations (i.e. allowing rollover and increasing number of planes) as it has the lowest cost: 15,125.** The corresponding aircraft schedule to the new cargo arrival schedule shown in Table 4, and the solution to the original task is shown in Table 6, 7, and 8.

Origin-Destination	Monday	Tuesday	Wednesday	Thursday	Friday
A-B	0	0	700	400	0
A-C	0	0	0	0	250
B-A	0	0	0	125	0
B-C	0	0	0	0	125
C-A	0	0	200	0	0
C-B	650	200	100	0	550

Table 6: Schedule for aircraft delivering cargo for each day for each origin-destination pair

Origin-Destination	Monday	Tuesday	Wednesday	Thursday	Friday
A-B	0	0	0	0	0
A-C	0	0	0	0	0
B-A	0	700	200	125	0
B-C	200	300	0	550	275
C-A	0	0	0	0	0
C-B	0	0	0	0	0

Table 7: Schedule for repositioning aircraft for each day for each origin-destination pair

Airport	Monday	Tuesday	Wednesday	Thursday	Friday
A	0	0	0	0	0
B	350	0	0	0	0
C	0	0	0	0	0

Table 8: Schedule for holding aircraft at each airport

Intuitively, the aircraft schedules above are sensible, given that the majority of cargo is being delivered to airport B. As a result of this, airport B must constantly reposition aircraft back to airports A and C, so that A and C can continue to deliver cargo to airport B in the following days. By the same logic, it also makes sense that airport B would hold aircraft at the beginning of the week that it can reposition later.

As noted above, we notice that airport B is performing all the repositioning of aircraft. If we want to further optimize the cost, we can seek to reduce the cost to reposition aircraft from B to A and from B to C. If we reduce the repositioning cost from B to A by one and B to C by one, our total cost decreases by 1,025 and 1,325 respectively.

We also explore the marginal impact of increasing the total amount of cargo that needs to be delivered between each origin-destination pair. This is summarized in Table 9.

Origin-Destination	Marginal Cost
A-B	7
A-C	1
B-A	-7
B-C	-6
C-A	-1
C-B	6

Table 9: The marginal impact to total cost when the amount of cargo that needs to be delivered is incremented by one for each origin-destination pair

Surprisingly increasing the total cargo for B-A, B-C, and C-A reduces the total cost. This is because for each incremental cargo added for these origin-destination pairs, an aircraft is now being used to deliver the cargo for "free", instead of being repositioned, which incurs a cost. Express Air may consider adding additional cargo to these origin-destination pairs.

appendix1

December 5, 2022

1 Appendix 1 - Original Formulation of Optimization Problem

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

# number of aircraft
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

# initialize cargo amounts
cargo_amounts = np.array([
    [100, 200, 100, 400, 300],
    [50, 50, 50, 50, 50],
    [25, 25, 25, 25, 25],
    [25, 25, 25, 25, 25],
    [40, 40, 40, 40, 40],
    [400, 200, 300, 200, 400]
])

# 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

# 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 + cargo_amounts[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 =␣
 ↪cargo_amounts[i][0])

# 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 the 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()
```

```
Set parameter Username
Academic license - for non-commercial use only - expires 2023-10-08
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 82 rows, 105 columns and 315 nonzeros
Model fingerprint: 0xbfb441be
Variable types: 0 continuous, 105 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        [3e+01, 1e+03]
Presolve removed 48 rows and 30 columns
Presolve time: 0.00s
Presolved: 34 rows, 75 columns, 225 nonzeros
Variable types: 0 continuous, 75 integer (0 binary)

Root relaxation: infeasible, 25 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 infeasible    0               - infeasible      -     -    0s

Explored 1 nodes (25 simplex iterations) in 0.01 seconds (0.00 work units)
Thread count was 8 (of 8 available processors)

Solution count 0

Model is infeasible or unbounded
Best objective -, best bound -, gap -
```

[ ]:
```
# 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))
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
c:\Users\jchu1\Desktop\FileLibrary\ORIE5380\project\project.ipynb Cell 3 in␣
 ↪<cell line: 2>()
```

```
      <a href='vscode-notebook-cell:/c%3A/Users/jchu1/Desktop/FileLibrary/
 ↪ORIE5380/project/project.ipynb#W1sZmlsZQ%3D%3D?line=0'>1</a> # print optimal
 ↪objective and optimal solution
----> <a href='vscode-notebook-cell:/c%3A/Users/jchu1/Desktop/FileLibrary/
 ↪ORIE5380/project/project.ipynb#W1sZmlsZQ%3D%3D?line=1'>2</a> print("\nOptimal
 ↪Objective: " + str(myModel.ObjVal))
      <a href='vscode-notebook-cell:/c%3A/Users/jchu1/Desktop/FileLibrary/
 ↪ORIE5380/project/project.ipynb#W1sZmlsZQ%3D%3D?line=2'>3</a> print("\nOptimal
 ↪Solution: " )
      <a href='vscode-notebook-cell:/c%3A/Users/jchu1/Desktop/FileLibrary/
 ↪ORIE5380/project/project.ipynb#W1sZmlsZQ%3D%3D?line=3'>4</a> allVars = myMode
 ↪getVars()

File src\gurobipy\model.pxi:353, in gurobipy.Model.__getattr__()

File src\gurobipy\model.pxi:1884, in gurobipy.Model.getAttr()

File src\gurobipy\attrutil.pxi:100, in gurobipy.__getattr()

AttributeError: Unable to retrieve attribute 'ObjVal'
```

appendix2

December 5, 2022

1 Appendix 2 - Updated Formulation with Number of Aircraft as Decision Variable

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

# 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

# initialize cargo amounts
cargo_amounts = np.array([
    [100, 200, 100, 400, 300],
    [50, 50, 50, 50, 50],
    [25, 25, 25, 25, 25],
    [25, 25, 25, 25, 25],
    [40, 40, 40, 40, 40],
    [400, 200, 300, 200, 400]
])

# 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 variable for total number of airplanes
aircraftVar = myModel.addVar(vtype = GRB.INTEGER, name = "aircraft")

# 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

# add a new cost for each aircraft (so to minimize total number required)
objExpr += aircraftVar

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 + cargo_amounts[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 =␣
 ↪cargo_amounts[i][0])

# 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 = aircraftVar, 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()
```

```
Set parameter Username
Academic license - for non-commercial use only - expires 2023-10-08
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 82 rows, 106 columns and 316 nonzeros
Model fingerprint: 0x4ad0dab4
Variable types: 0 continuous, 106 integer (0 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [1e+00, 1e+01]
  Bounds range     [0e+00, 0e+00]
  RHS range        [3e+01, 4e+02]
Presolve removed 49 rows and 31 columns
Presolve time: 0.00s
Presolved: 33 rows, 75 columns, 210 nonzeros
Variable types: 0 continuous, 75 integer (0 binary)

Root relaxation: objective 1.651500e+04, 36 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    16515.000000 16515.0000  0.00%     -    0s

Explored 1 nodes (36 simplex iterations) in 0.02 seconds (0.00 work units)
Thread count was 8 (of 8 available processors)

Solution count 1: 16515

Optimal solution found (tolerance 1.00e-04)
Best objective 1.651500000000e+04, best bound 1.651500000000e+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: 16515.0

Optimal Solution:
xAB1 100.0
xAB2 200.0
xAB3 100.0
xAB4 400.0
xAB5 300.0
xAC1 50.0
xAC2 50.0
xAC3 50.0
xAC4 50.0
xAC5 50.0
xBA1 25.0
xBA2 25.0
xBA3 25.0
xBA4 25.0
xBA5 25.0
xBC1 25.0
xBC2 25.0
xBC3 25.0
xBC4 25.0
xBC5 25.0
xCA1 40.0
xCA2 40.0
xCA3 40.0
xCA4 40.0
xCA5 40.0
xCB1 400.0
xCB2 200.0
xCB3 300.0
xCB4 200.0
xCB5 400.0
yAB1 100.0
yAB2 200.0
yAB3 100.0
yAB4 400.0
yAB5 300.0
yAC1 50.0
yAC2 50.0
yAC3 50.0
yAC4 50.0
yAC5 50.0
```

```
yBA1 25.0
yBA2 25.0
yBA3 25.0
yBA4 25.0
yBA5 25.0
yBC1 25.0
yBC2 25.0
yBC3 25.0
yBC4 25.0
yBC5 25.0
yCA1 40.0
yCA2 40.0
yCA3 40.0
yCA4 40.0
yCA5 40.0
yCB1 400.0
yCB2 200.0
yCB3 300.0
yCB4 200.0
yCB5 400.0
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 320.0
zBA2 450.0
zBA3 170.0
zBA4 -0.0
zBA5 85.0
zBC1 430.0
zBC2 -0.0
zBC3 180.0
zBC4 350.0
zBC5 365.0
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 135.0
sA3 500.0
sA4 285.0
sA5 -0.0
sB1 -0.0
sB2 -0.0
sB3 -0.0
sB4 -0.0
sB5 100.0
sC1 -0.0
sC2 265.0
sC3 -0.0
sC4 15.0
sC5 -0.0
aircraft 1390.0
```

appendix3

December 5, 2022

1 Appendix 3 - Updated Formulation with Weekly Rollover

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

# number of aircraft
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

# initialize cargo amounts
cargo_amounts = np.array([
    [100, 200, 100, 400, 300],
    [50, 50, 50, 50, 50],
    [25, 25, 25, 25, 25],
    [25, 25, 25, 25, 25],
    [40, 40, 40, 40, 40],
    [400, 200, 300, 200, 400]
])

# 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

# 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's cargo should roll-over
        if j != 4:
            xVar2 = xVars[i][j+1]
            constExpr += xVar1 + cargo_amounts[i][j+1] - yVar1
            myModel.addConstr(lhs = constExpr, sense=GRB.EQUAL, rhs = xVar2,␣
 ↪name = "CargoSupplyConstraints")
        else:
            xVar2 = xVars[i][0]
            constExpr += xVar1 + cargo_amounts[i][0] - yVar1
            myModel.addConstr(lhs = constExpr, sense=GRB.EQUAL, rhs = xVar2,␣
 ↪name = "CargoSupplyConstraints")

# 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 the 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 76 rows, 105 columns and 315 nonzeros
Model fingerprint: 0xab13a924
Variable types: 0 continuous, 105 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        [3e+01, 1e+03]
Presolve removed 30 rows and 24 columns
Presolve time: 0.00s
Presolved: 46 rows, 81 columns, 255 nonzeros
Variable types: 0 continuous, 81 integer (0 binary)

Root relaxation: objective 1.792500e+04, 42 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    17925.000000 17925.0000  0.00%     -    0s

Explored 1 nodes (42 simplex iterations) in 0.01 seconds (0.00 work units)
Thread count was 8 (of 8 available processors)

Solution count 1: 17925

Optimal solution found (tolerance 1.00e-04)
Best objective 1.792500000000e+04, best bound 1.792500000000e+04, gap 0.0000%
```

[ ]:
```python
# 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: 17925.0

Optimal Solution:
xAB1 290.0
xAB2 290.0
xAB3 100.0
xAB4 400.0
xAB5 300.0
xAC1 50.0
xAC2 50.0
```

```
xAC3 50.0
xAC4 50.0
xAC5 50.0
xBA1 25.0
xBA2 25.0
xBA3 25.0
xBA4 25.0
xBA5 25.0
xBC1 25.0
xBC2 25.0
xBC3 25.0
xBC4 25.0
xBC5 25.0
xCA1 40.0
xCA2 40.0
xCA3 40.0
xCA4 40.0
xCA5 40.0
xCB1 400.0
xCB2 200.0
xCB3 300.0
xCB4 200.0
xCB5 400.0
yAB1 200.0
yAB2 290.0
yAB3 100.0
yAB4 400.0
yAB5 110.0
yAC1 50.0
yAC2 50.0
yAC3 50.0
yAC4 50.0
yAC5 50.0
yBA1 25.0
yBA2 25.0
yBA3 25.0
yBA4 25.0
yBA5 25.0
yBC1 25.0
yBC2 25.0
yBC3 25.0
yBC4 25.0
yBC5 25.0
yCA1 40.0
yCA2 40.0
yCA3 40.0
yCA4 40.0
yCA5 40.0
```

```
yCB1 400.0
yCB2 200.0
yCB3 300.0
yCB4 200.0
yCB5 400.0
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 275.0
zBA2 305.0
zBA3 165.0
zBA4 95.0
zBA5 185.0
zBC1 165.0
zBC2 265.0
zBC3 275.0
zBC4 255.0
zBC5 365.0
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 220.0
sA4 -0.0
sA5 -0.0
sB1 20.0
sB2 -0.0
sB3 -0.0
sB4 -0.0
sB5 -0.0
sC1 -0.0
sC2 -0.0
sC3 -0.0
```

```
sC4 110.0
sC5 -0.0
```

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
```