Optimizing New Fulfillment Center Locations to Minimize Distribution Costs

Krishna Datta, Abraham Hill, Calvin Ying

May 2022

1 Executive Summary

Our team is entrusted with planning the opening of 3 fulfillment centers (FCs) that will be used over the next 20 years. We were provided 10 potential locations to open FCs and 20 demand points (DPs) that the company will serve. The first FC will be opened immediately and will be the only operational FC for 6 years. After 6 years, a second FC will be opened and the company will operate the two FCs concurrently for another 5 years. After a total of 11 years, a third FC will be opened and all three FCs will be operational for another 9 years. The business objective is to select the optimal facility locations so that the total cost (ie. the total distance between each operating FC and each demand point) over the next 20 years is minimized. To accomplish this task, we formulated an integer programming model and ran it in Python using the Gurobi optimization package.

Our proposed plan is to open **the first FC at location (57, 54), the second FC at location (4, 79), and the third FC at location (60, 15)**. Under this proposal, the total cost is 1994 (in units of distance) over 20 years, or 99.7 (in units of distance) per year on average.

To validate whether our model provides the optimal output, we formulated a few alternative models and compared them to our preferred model. We observed that our model indeed minimizes the total cost. In the following sections, we will give an overview of our model formulation, a summary of the model findings, and recommendations for future model refinements. We also attached the mathematical model formulation as well as raw Python code in the Appendix.

2 Model Formulation

The model we built is an integer program. Integer programming is an optimization technique with broad business applications, including vehicle routing, shift scheduling, production planning, and facility location. It leverages an underlying algorithm that evaluates a sequence of possible solutions until the optimal one is found. Every integer program consists of a set of decision variables, which encode possible business decisions we could make, an objective function, which defines the quality of a solution, and a set of constraints, which impose requirements on the solution.

In this problem, we defined the objective to be minimizing distribution distance. This is a good proxy for the total distribution cost, which includes the wages of delivery drivers, gasoline, and truck depreciation, which all scale proportionally with distance. Given the way we formulated the objective function of our model, we are very confident that the solution we found will minimize total distribution costs over the twenty year horizon.

The constraints added to the integer program ensure that the model is a valid test of our current decision point. We added constraints to require the appropriate sequencing of FC openings and to ensure that every DP will be served by some FC each year. We also added constraints so that our specific formulation of the model works as intended.

This model performs remarkably well, as measured both in solution quality and time to find the optimal solution. This problem is somewhat complex because the objective is to minimize the total cost over three phases. One approach to solving this problem is to break it into parts: first find the lowest cost combination of three FCs, then find the lowest cost opening sequence. This approach is not guaranteed to return the optimal solution because the model does not minimize the total cost over all time periods, so we are likely to have especially high costs in the first and second phases. We were able to create a model that is guaranteed to find the optimal solution by creating a set of variables for each phase. This allows us to minimize total cost and leverage the full power of integer programming to find the optimal solution.

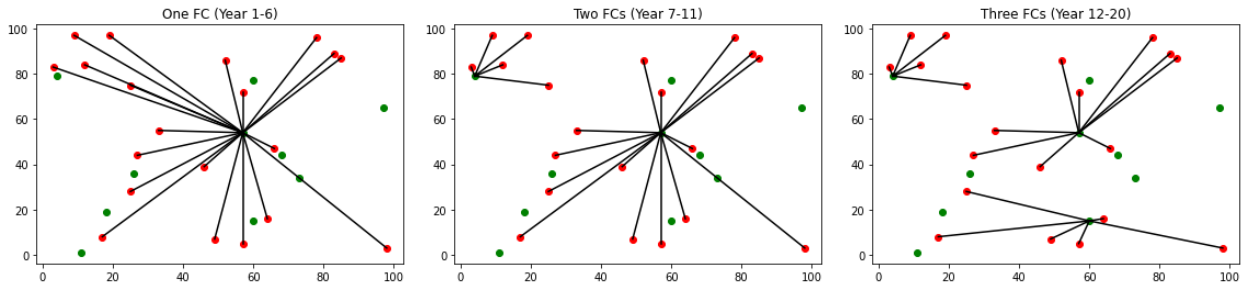
Furthermore, our model is able to find the optimal solution in about 0.02 seconds. This suggests that the model could be useful for problem instances of significantly larger size as the business scales.

3 Summary

From Year 1 to Year 6, we will open and operate the first FC at location (57, 54). We will incur a total cost of 848 (in units of distance) and an average cost of 141.3 per year over these 6 years. As seen in the left image below (where red dots represent DPs and green dots represent potential FC locations), the first FC is located at the center of the map, situating close to all DPs.

From Year 7 to Year 11, we will open the second FC at location (4, 79), and we will operate both the first and the second FCs concurrently during this time. We will incur a total cost of 650 (in units of distance) and an average cost of 130.0 per year over these 5 years. As seen in the center image below, the second FC is located at the upper left corner of the map, handling demand from DPs that are farther away from the rest of DPs.

From Year 12 to Year 20, we will open the third FC at location (60, 15), and we will operate all three FCs concurrently during this time. We will incur a total cost of 496 (in units of distance) and an average cost of 55.1 per year over these 9 years. As seen in the right image below, the third FC is located at the bottom of the map.



It is worth noting that as we open more FCs, the average cost per year drastically decreases, as customers can now be served by a FC that is located much more closely to them.

3.1 Future Opportunities

As listed below, we have identified a few opportunities to improve model performance and refine our model in order to align more closely with the real world practice.

- In addition to minimizing total cost, we can also consider setting the model objective function as maximizing profit. After all, businesses value profitability the most. If we decide to focus on maximizing profit, we may want to place FCs closer to the DPs that will generate the most revenue, even if this means increasing our overall total cost (as we expect the revenue to offset the cost). Furthermore, under this new model formulation, we may want to abandon DPs with lower spend potential, and thus our FCs will not need to serve all DPs.
- In our current model formulation, we assume all DPs have the same demand. This is overly simplistic, and we should refine our model by incorporating varying demand at different DPs. Furthermore, at each DP, we may see different demand behavior at different time of day or day of week. In order to simulate the real world practice, we should consider constructing and incorporating demand distributions in our model.
- In our current model formulation, we assume each potential FC has unlimited capacity. This is generally untrue in the real world, as FCs have limited space for inventory storage. Therefore, we should consider adding a constraint for facility capacity at each FC. Including these constraints in our model may alter the optimal locations to open FCs.
- In our current model formulation, we assume that the cost of opening and operating a FC is the same across all potential locations. In reality, these costs vary based on local real estate and labor markets. Therefore, we should consider minimizing operating costs as well as distribution costs.
- In our current model formulation, we do not discount the value of future savings. The value of money decreases over time, and as a result, future costs are slightly less important than current costs. Therefore, we should consider adding a discount factor to our objective function in order to compute the optimal present day value of the future costs we will incur.

4 Appendix: Technical Details

4.1 Data

The set $\mathcal{F} : \{i \in 1, \ldots, 10\}$ indexes the ten candidate locations for the three fulfillment centers (FCs) to be opened. The set $\mathcal{C} : \{j \in 1, \ldots, 20\}$ contains the twenty demand points (DPs) to be served by the FCs. Each FC and DP has an associated coordinate location, from which the distances d_{ij} between each FC and DP pair are calculated. We also define a set $\mathcal{T} : \{k \in 1, 2, 3\}$ of time periods, each with an associated length t_k. At the beginning of each time period, a new FC is opened, so in this case $t_k = (6, 5, 9)$ because the first FC serves alone for six years, then two serve together for five years, and finally all three are open for the remaining nine years.

4.2 Decision Variables

There are two sets of variables. The first set of variables are the decision variables, x_i^k, which specify which FCs will be open in each time period. The second set of variables, y_{ij}^k, specify which FCs will serve which DPs in each time period.

$$x_i^k = \begin{cases} 1 \text{ if FC } i \text{ is open in time period } k \\ 0 \text{ otherwise} \end{cases} \tag{1}$$

$$y_{ij}^k = \begin{cases} 1 \text{ if FC } i \text{ serves DP } j \text{ in time period } k \\ 0 \text{ otherwise} \end{cases} \tag{2}$$

4.3 Model

The following integer program was used to find the fulfillment center locations and opening sequence that minimize the total cost over the twenty year horizon.

$$\begin{aligned}
\text{minimize} \quad & \sum_{k \in \mathcal{T}} t_k \sum_{i \in \mathcal{F}} \sum_{j \in \mathcal{C}} d_{ij} y_{ij}^k \\
\text{subject to} \quad & y_{ij}^k \leq x_i^k, && i \in \mathcal{F}, j \in \mathcal{C}, k \in \mathcal{T}. \text{ Closed FCs cannot serve any DPs} \\
& \sum_{i \in \mathcal{F}} x_i^k = k, && k \in \mathcal{T}. \text{ Can only open one new FC each time period} \\
& \sum_{i \in \mathcal{F}} y_{ij}^k = 1, && j \in \mathcal{C},\ k \in \mathcal{T}. \text{ Each DP must be served by a FC} \\
& x_i^k \geq x_i^{k-1}, && k \in \mathcal{T}. \text{ Opened FCs must remain open in future time periods} \\
& x_i^k, y_{ij}^k \in \{0, 1\}, && i \in \mathcal{F}, j \in \mathcal{C}. \text{ Binary restrictions}
\end{aligned}$$

The nonzero values of x_i^k in the optimal solution correspond to facilities that should be opened in order to minimize costs.

4.4 Implementation

This integer program was implemented in Python and solved using Gurobi. The code is attached.

```
from gurobipy import *
import numpy as np
import matplotlib.pyplot
import pandas as pd

def initializeData():
    fcs = pd.read_excel('data.xlsx', sheet_name='FCs').values.tolist()
    dps = pd.read_excel('data.xlsx', sheet_name='DPs').values.tolist()
    noFacilities = len(fcs)
    noDemandPoints = len(dps)
    distances = []
    for idx, i in enumerate(fcs):
        distances.append([])
        for j in dps:
            distances[idx].append(np.linalg.norm(np.array(i) - np.array(j)))
    timePeriods = [6, 5, 9]

    return ( fcs, dps, noFacilities, noDemandPoints, distances, timePeriods )

( fcs, dps, noFacilities, noDemandPoints,
 distances, timePeriods ) = initializeData()
noPeriods = len(timePeriods)

xVars = [ [ 0 for i in range( noFacilities ) ] for j in range( noPeriods ) ]
yVars = [ [ [ 0 for i in range( noDemandPoints ) ]
          for j in range( noFacilities ) ] for k in range( noPeriods ) ]

ipModel = Model( "project" )

def constructVars():
    global noFacilities , noDemandPoints , ipModel , xVars , yVars , noPeriods
    for k in range( noPeriods ):
        for i in range( noFacilities ):
            var = ipModel.addVar( vtype = GRB.BINARY ,
                                  name = "x_" + str( k + 1 )
                                  + "_" + str( i + 1 ) )
            xVars[ k ][ i ] = var

    for k in range( noPeriods ):
        for i in range( noFacilities ):
            for j in range( noDemandPoints ):
                var = ipModel.addVar( vtype = GRB.BINARY ,
                                      name = "y_" + str( k + 1 ) +
                                      "_" + str( i + 1 ) + "_" + str( j + 1 ) )
                yVars[ k ][ i ][ j ] = var

    ipModel.update()
```

```
48
49  def constructObj():
50      global noFacilities , noDemandPoints , distances , ipModel , yVars
51      objExpr = LinExpr()
52      for k in range( noPeriods ):
53          objExprTemp = LinExpr()
54          for i in range( noFacilities ):
55              for j in range( noDemandPoints ):
56                  objExprTemp += distances[ i ][ j ] * yVars[ k ][ i ][ j ]
57          objExpr += objExprTemp
58      ipModel.setObjective( objExpr , GRB.MINIMIZE )
59      ipModel.update()
60
61  def contructConstrs():
62      global yVars , ipModel , xVars , noDemandPoints , noFacilities , noPeriods
63
64      for k in range( noPeriods ):
65          for i in range( noFacilities ):
66              constExprRhs = LinExpr()
67              constExprRhs = 1.0 * xVars[ k ][ i ]
68              for j in range( noDemandPoints ):
69                  constExprLhs = LinExpr()
70                  constExprLhs = 1.0 * yVars[ k ][ i ][ j ]
71                  ipModel.addConstr( lhs = constExprLhs ,
72                                     sense = GRB.LESS_EQUAL , rhs = constExprRhs )
73
74      for k in range( noPeriods ):
75          constExprLhs = LinExpr()
76          constExprRhs = LinExpr()
77          constExprRhs = 1.0 * ( k + 1 )
78          for i in range( noFacilities ):
79              constExprLhs += 1.0 * xVars[ k ][ i ]
80          ipModel.addConstr( lhs = constExprLhs ,
81                             sense = GRB.EQUAL , rhs = constExprRhs )
82
83      constExprRhs = LinExpr()
84      constExprRhs = 1.0
85      for k in range( noPeriods ):
86          for j in range( noDemandPoints ):
87              constExprLhs = LinExpr()
88              for i in range( noFacilities ):
89                  constExprLhs += yVars[ k ][ i ][ j ]
90              ipModel.addConstr( lhs = constExprLhs ,
91                                 sense = GRB.EQUAL , rhs = constExprRhs )
92
93      for k in range( 1, noPeriods ):
94          for i in range( noFacilities ):
95              constExprLhs = LinExpr()
96              constExprRhs = LinExpr()
97              constExprLhs = 1.0 * xVars[ k ][ i ]
98              constExprRhs = 1.0 * xVars[ k - 1 ][ i ]
99              ipModel.addConstr( lhs = constExprLhs ,
100                                 sense = GRB.GREATER_EQUAL , rhs = constExprRhs )
101
```

```
    ipModel.update()

constructVars()
constructObj()
contructConstrs()
ipModel.write( "project.lp" )
ipModel.optimize()
ipModel.printAttr( "X" )

data = ipModel.getAttr( "X" )
assgns = [ [ 0 for i in range( noDemandPoints ) ] for j in range( noPeriods ) ]
yValues = data[noPeriods * noFacilities:]
yValues = np.reshape(yValues, (noPeriods, noFacilities, noDemandPoints))
for k in range( noPeriods ):
    for i in range( noFacilities ):
        for j in range( noDemandPoints ):
            if yValues[ k ][ i ][ j ] == 1.0:
                assgns[ k ][ j ] = i

for k in range( noPeriods ):
    for fc in range( noFacilities ):
        matplotlib.pyplot.plot( fcs[ fc ][ 0 ]  , fcs[ fc ][ 1 ] , 'ro' ,
                                color = "green" , lw = 9 )

    for dp in range( noDemandPoints ):
        matplotlib.pyplot.plot( dps[ dp ][ 0 ]  , dps[ dp ][ 1 ] , 'ro' ,
                                color = "red" , lw = 9 )

    for dp in range( noDemandPoints ):
        dpx = dps[ dp ][ 0 ]
        dpy = dps[ dp ][ 1 ]
        fcx = fcs[ assgns[ k ][ dp ] ][ 0 ]
        fcy = fcs[ assgns[ k ][ dp ] ][ 1 ]
        matplotlib.pyplot.plot( [ dpx , fcx ], [ dpy , fcy ]  ,
                                color = "black"  )

    matplotlib.pyplot.show()
```