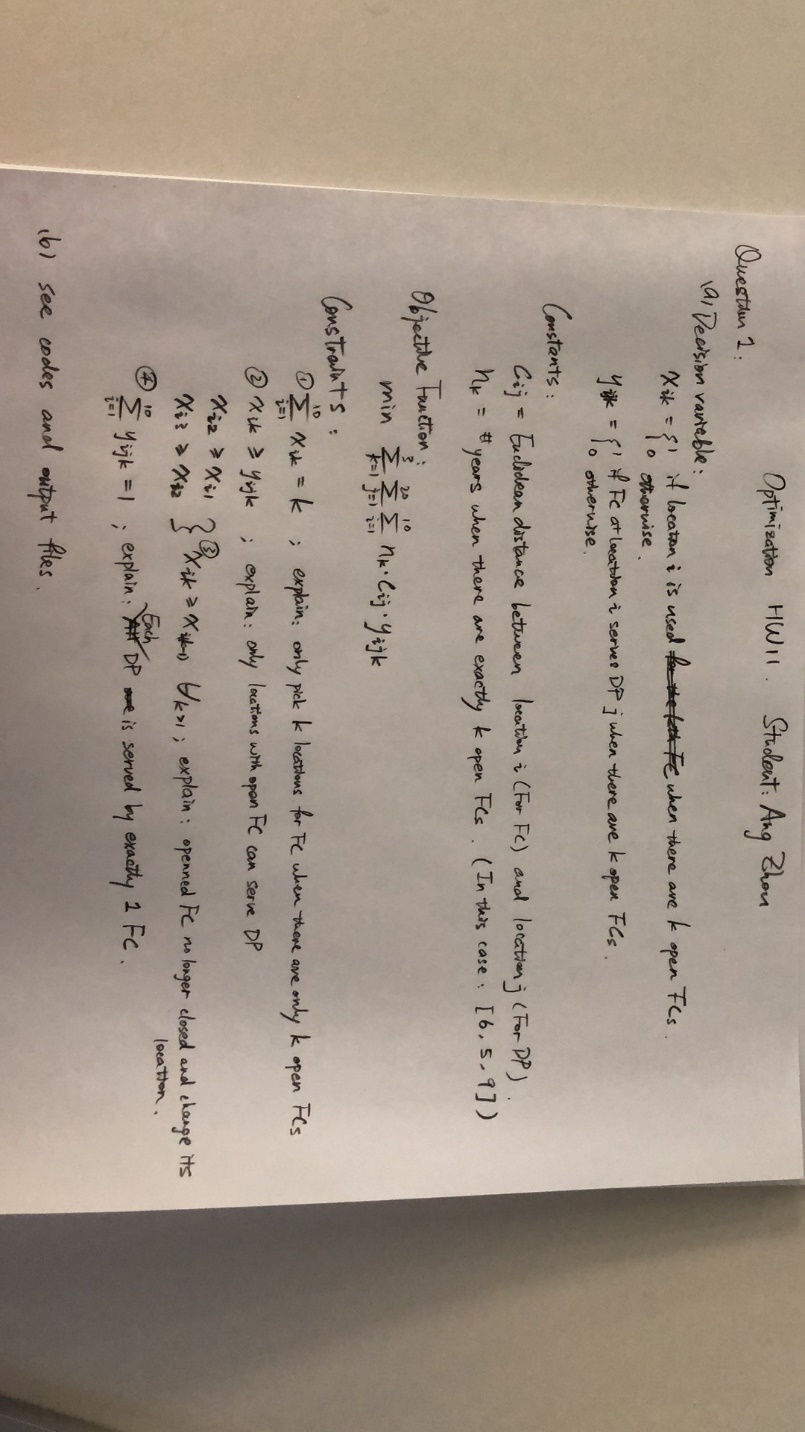
Optimization HW11 Student: Ang Zhou



(b) Code

from gurobipy import \*

import xlwt

import xlrd

import pandas as pd

import numpy as np

from scipy.spatial import distance

import matplotlib.pyplot as plt

# loading data

f = xlrd.open\_workbook('data.xlsx')

sheet = f.sheet\_by\_index(0)

df1 = pd.read\_excel('data.xlsx','FCs')

df2 = pd.read\_excel('data.xlsx','DPs')

A = list(df1.iloc[0])

B = list(df2.iloc[0])

distance.euclidean(A,B)

# create a new model

myModel = Model("HW11\_Q1")

# create decision vats and integrate them into the model

i\_s = df1.shape[0] ## number of FC

j\_s = df2.shape[0] ## number of DP

k\_s = 3

# vars storage

c\_s = [[0 for j in range(j\_s)] for i in range(i\_s)]

x\_s = [[0 for k in range(k\_s)] for i in range(i\_s)]

y\_s = [[[0 for k in range(k\_s)] for j in range(j\_s)] for i in range(i\_s)]

n\_s = [6,5,9]

# c\_s (cij) for distances

for i in range(i\_s):

for j in range(j\_s):

a = list(df1.iloc[i])

b = list(df2.iloc[j])

c\_s[i][j] = distance.euclidean(a,b)

# x\_s (xik)

for i in range(i\_s):

for k in range(k\_s):

xVar = myModel.addVar(vtype=GRB.INTEGER, name='x' + str(i+1) + ',' + str(k+1))

x\_s[i][k] = xVar

myModel.update()

# y\_s (yijk)

for i in range(i\_s):

for j in range(j\_s):

for k in range(k\_s):

yVar = myModel.addVar(vtype=GRB.INTEGER, name='y' + str(i+1) + ',' + str(j+1) + ',' + str(k+1))

y\_s[i][j][k] = yVar

myModel.update()

# create a linear expression for the objective

objExpr = LinExpr()

for i in range(i\_s):

for j in range(j\_s):

for k in range(k\_s):

yVar = y\_s[i][j][k]

c = c\_s[i][j]

n = n\_s[k]

objExpr += n\*c\*yVar

myModel.setObjective(objExpr, GRB.MINIMIZE)

myModel.update()

# Constraint for number of FC in each stage

for k in range(k\_s):

constExpr = LinExpr()

for i in range(i\_s):

xVar = x\_s[i][k]

constExpr += xVar

myModel.addConstr(lhs=constExpr, sense=GRB.EQUAL, rhs=k+1, name="stage" + str(k+1))

# Constraint for empty FC locations

for k in range(k\_s):

for j in range(j\_s):

for i in range(i\_s):

constExpr = LinExpr()

constExpr += x\_s[i][k] - y\_s[i][j][k]

myModel.addConstr(lhs=constExpr, sense=GRB.GREATER\_EQUAL, rhs=0, name="fc\_location" + str(j+1))

# Constraint for fixed FC locations

for i in range(i\_s):

constExpr1 = LinExpr()

constExpr1 += x\_s[i][1] - x\_s[i][0]

constExpr2 = LinExpr()

constExpr2 += x\_s[i][2] - x\_s[i][1]

myModel.addConstr(lhs=constExpr1, sense=GRB.GREATER\_EQUAL, rhs=0, name="fc\_location\_fix" + str(1) + str(i))

myModel.addConstr(lhs=constExpr2, sense=GRB.GREATER\_EQUAL, rhs=0, name="fc\_location\_fix" + str(2) + str(i))

# Constraint for each DP has exactly 1 FC

for k in range(k\_s):

for j in range(j\_s):

constExpr = LinExpr()

for i in range(i\_s):

yVar = y\_s[i][j][k]

constExpr += yVar

myModel.addConstr(lhs=constExpr, sense=GRB.EQUAL, rhs=1, name="full\_cover" + str(j+1) + ',' + str(k+1))

# boundaries

for k in range(k\_s):

for j in range(j\_s):

for i in range(i\_s):

constExpr = LinExpr()

constExpr = y\_s[i][j][k]

myModel.addConstr(lhs=constExpr, sense=GRB.LESS\_EQUAL, rhs=1, name="boundary\_y" + str(i+1) + ',' + str(j+1) + ',' + str(k+1))

for k in range(k\_s):

for i in range(i\_s):

constExpr = LinExpr()

constExpr = x\_s[i][k]

myModel.addConstr(lhs=constExpr, sense=GRB.LESS\_EQUAL, rhs=1, name="boundary\_x" + str(i+1) + ',' + str(k+1))

# 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="HW11\_Q1.lp")

# optimize the model

myModel.optimize()

allVars = myModel.getVars()

# this array includes the coordinates of fulfillment centers

# there are 10 fulfillment center locations

# for each fulfillment center, we keep x and y coordinates

nofcs = 10

fcs = [ 0 for j in range ( nofcs ) ]

fcs[0] = [60 , 15]

fcs[1] = [26 , 36]

fcs[2] = [73 , 34]

fcs[3] = [57 , 54]

fcs[4] = [18 , 19]

fcs[5] = [11 , 1]

fcs[6] = [60 , 77]

fcs[7] = [68 , 44]

fcs[8] = [97 , 65]

fcs[9] = [4 , 79]

# this array includes the coordinates of demand points

# there are 10 demand points

# for each demand point, we keep x and y coordinates

# for this example, there are 20 demand points

nodps = 20

dps = [ 0 for j in range ( nodps ) ]

dps[0] = [25 , 75]

dps[1] = [49 , 7]

dps[2] = [17 , 8]

dps[3] = [12 , 84]

dps[4] = [3 , 83]

dps[5] = [57 , 5]

dps[6] = [46 , 39]

dps[7] = [83 , 89]

dps[8] = [78 , 96]

dps[9] = [27 , 44]

dps[10] = [64 , 16]

dps[11] = [52 , 86]

dps[12] = [57 , 72]

dps[13] = [33 , 55]

dps[14] = [66 , 47]

dps[15] = [25 , 28]

dps[16] = [9 , 97]

dps[17] = [85 , 87]

dps[18] = [98 , 3]

dps[19] = [19 , 97]

# this array includes which fulfillment center each demand point is connected to

for k in range(k\_s):

assgns = [ 0 for j in range ( nodps ) ]

for j in range(j\_s):

for i in range(i\_s):

if int(y\_s[i][j][k].x) == 1:

assgns[j] = i

for fc in range( nofcs ):

plt.plot( fcs[ fc ][ 0 ] , fcs[ fc ][ 1 ] , 'ro' , color = "green" , lw = 9 )

for dp in range( nodps ):

plt.plot( dps[ dp ][ 0 ] , dps[ dp ][ 1 ] , 'ro' , color = "red" , lw = 9 )

for dp in range( nodps ):

dpx = dps[ dp ][ 0 ]

dpy = dps[ dp ][ 1 ]

fcx = fcs[ assgns[ dp ] ][ 0 ]

fcy = fcs[ assgns[ dp ] ][ 1 ]

plt.plot( [ dpx , fcx ], [ dpy , fcy ] , color = "black" )

plt.savefig(str(k+1)+'\_FC.png')

plt.show()

Results:

Objective: Optimal solution found (tolerance 1.00e-04)

Best objective 1.276015102350e+04

