ORIE 5380 Optimization Methods Project

Abraham Hill, Sal Galarza

November 2021

1 Model

Our decision variables $Y \in \mathbb{R}^{nxn}$ are clients' assignments to depots. $Y_{ij} = 1$ if client i is assigned to depot j, and otherwise it is 0. We are also defining a parameter called "depots" in the code, which I will denote here as $y \in \mathbb{R}^{1xn}$, a vector that keeps track of which columns of Y are actually depots. Since there will likely be fewer than n depots, Y will have some columns that have all zeros. $y_j = 1$ if the j-th column of Y has nonzero entries, and it is 0 otherwise. We are also defining a parameter $D \in \mathbb{R}^{nxn}$ which is a matrix of distances between the clients. D_{ij} is the distance between client i and j, which is calculated from the matrix X. With these definitions in place, our model is as follows:

$$minimize \sum_{j=1}^{n} y_j$$

Subject to

$$y_j \geq \frac{\sum_{i=1}^{n} Y_{ij}}{n+1}, \forall j$$

lower bound on y

$$y_j \leq \sum_{i=1}^{n} Y_{ij} + 0.5, \forall j$$

upper bound on y

$$D_{ij} = \sqrt{(X_{0i} - X_{0j})^2 + (X_{1i} - X_{1j})^2}, \forall i, j$$

definition of D

$$\sum_{j=1}^{n} Y_{ij} = 1, \forall i$$

Uniqueness of client assignment

$$y_i n_{min} \leq \sum_{i=1}^{n} Y_{ij} \leq y_i n_{max}, \forall i$$

Depot number of clients

$$D_{ik} \leq d_{max} + 10000(2 - Y_{ij} - Y_{kj}), \forall j, i, k$$

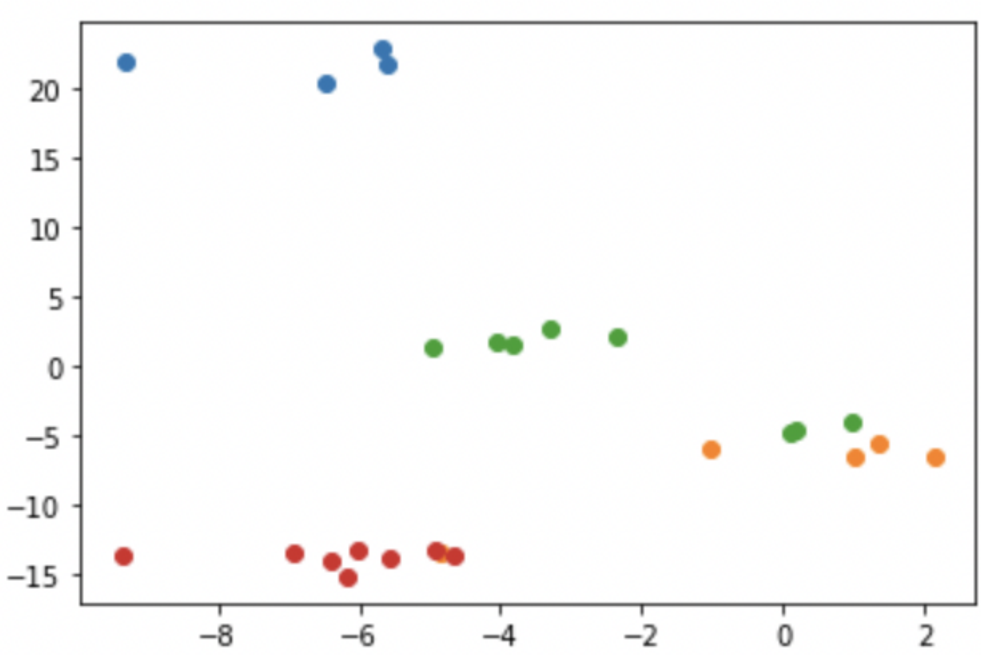
Distance between clients for a depot

$$Y, y \in 0, 1$$

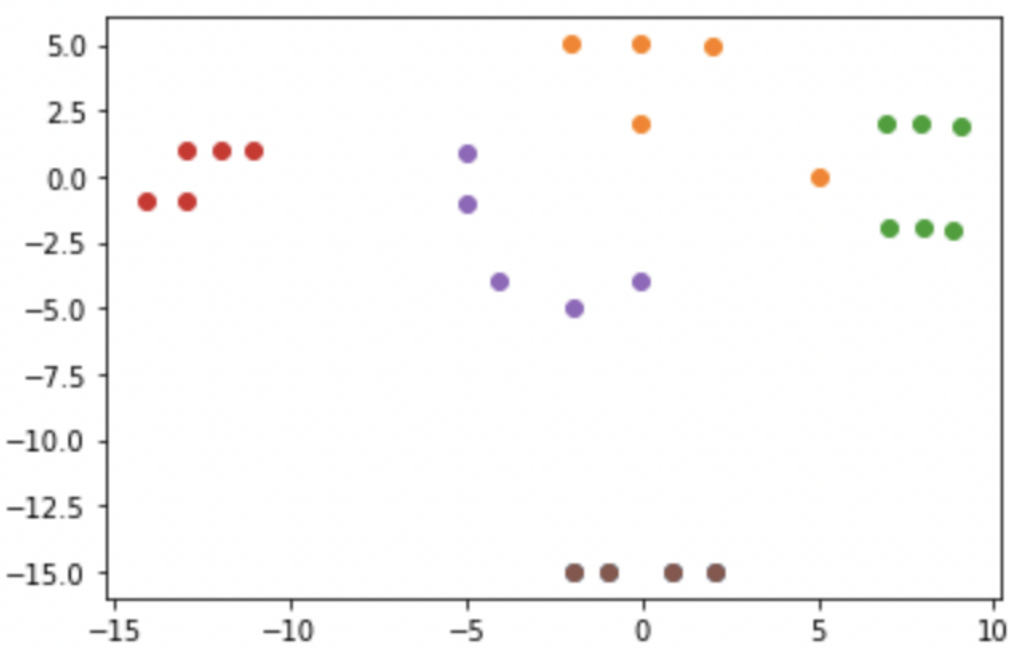
Binary constraint

Although it does take two constraints to define, the vector y is very useful in the objective function of this problem as well as in the number of clients in a depot constraint. In the objective, we can simply sum the elements of y to get the desired quantity. In the number of clients in a depot constraint, y acts indicates whether a column of Y represents a depot. There are many columns of Y with all zeros, which would violate the minimum number of clients constraint if not for the use of y_i being multiplied by n_{min} and n_{max}. I think the maximum distance constraint is very strong. If either Y_{ij} or Y_{kj} are equal to 0 (i.e. clients i and k are not in the same depot), the constraint is automatically satisfied because we are multiplying by a large number on the right hand side of the inequality. Otherwise, the constraint reduces to $D_{ik} \leq d_{max}$, which is what we want. Here are the solutions given by this model for each instance:

Instance 1 has an optimal value of 4, with the following clusters:



Instance 2 has an optimal value of 5, with the following clusters:



Instance 3 is infeasible.

2 Results with Different Parameters

In instance 1, changing n_{min} in either direction does not lead to an increase in or decrease in objective value. Increasing it from 2 to 5 makes the model infeasible. Decreasing n_{max} from 8 to 6 increases the optimal value to 5, and further decreasing it to 5 increases the optimal value again to 6. The pattern continues: $n_{max} = 5$ yields optimal value of 7. It is the same at 4, but 3 yields an optimal value of 9 and 2 is infeasible. Increasing n_{max} does not change the objective. Decreasing d_{max} to 9 increases the objective to 5, and the model is infeasible at $d_{max} = 3$

In instance 2, decreasing n_{min} does not lead to an increase in objective value. Increasing it from 4 to 5 makes the model infeasible.

In instance 3, decreasing n_{min} from 4 to 3 makes the problem feasible, and the optimal solution has objective value 6. Further decreasing n_{min} does not increase objective value.

3 Minimizing Average Pairwise Distance Within Clusters

The constraints for this problem are the same. To compute the objective function, we computed all the pairwise distances, then multiplied all the distances of pairs of clients that are not in the same cluster by zero. To do this, we defined a new parameter $Z_{ik} \in \mathbb{R}^{nxn}$ that indicates whether clients i and k are in the same cluster. $Z_{ik} = 1$ if i and k are in the same cluster and is 0 otherwise. Using this, the new objective equation is

$$minimize \sum_{i=1}^{n} \sum_{k=1}^{n} D_{ik} Z_{ik}/2$$

We divide by 2 to adjust for double-counting pairs. Z is constrained to be binary, and these are the upper and lower bounds:

$$Z_{ik} \geq Y_{ij} + Y_{kj} - 1.9, \forall j$$

lower bound on Z

$$Z_{ik} \leq (Y_{ij} + Y_{kj})/2 + 0.1, \forall j$$

upper bound on Z

This model took a very long time to run, so we will not include the solution in this report. However, the implementation is included in the code.

In [344]:
```
import sys
print(sys.executable)
!{sys.executable} -m pip install gurobipy
from gurobipy import *
import math
from matplotlib import pyplot as plt
```

```
/Users/Abe/opt/anaconda3/bin/python
Requirement already satisfied: gurobipy in /Users/Abe/opt/anaconda3/lib/p
ython3.8/site-packages (9.5.0)
```

In [345]:
```
#open jld
#https://docs.h5py.org/en/stable/high/group.html
#https://docs.h5py.org/en/stable/high/attr.html
import h5py
x1_data = h5py.File("X1.jld", "r")
x2_data = h5py.File("X2.jld", "r")
x3_data = h5py.File("X3.jld", "r")

#print(x1_data.keys(),x2_data.keys(),x3_data.keys())

# f["input"].value, f["output"].value
# f["X"].value
# f["_creator"]

print(x1_data["X"])
print(x1_data["X"][0])
print(x1_data["X"][1])

#x1_data["_creator"].keys()
```

```
<HDF5 dataset "X": shape (2, 25), type "<f8">
[-9.31431352 -5.66901261 -5.58994131 -6.48529461 -4.64803558 -6.92318026
 -4.9219469  -6.18895012 -6.383588   -4.83052083 -5.54884829 -6.03902456
 -9.36696392 -1.01326107  0.08758034  1.34522284  1.02763538  0.18065355
  0.96876053  2.14521255 -3.81989054 -4.94544952 -3.29144358 -2.36855237
 -4.05178677]
[ 21.85248196  22.82571229  21.70189592  20.44812788 -13.57047711
 -13.45455931 -13.16898973 -15.16087729 -14.11121926 -13.3698081
 -13.74643136 -13.17614957 -13.63194927  -5.99053048  -4.86670161
  -5.53625377  -6.47446421  -4.69903272  -4.01117971  -6.54907345
   1.50498652   1.28581208   2.7638501    2.11314414   1.67859478]
```

In [346]:
```
instance = 2
# 1, 2, or 3

if instance == 1:
    #2, 8, 10
    n_min = 2
    n_max = 8
    d_max = 10
    X = [x1_data["X"][0], x1_data["X"][1]]

elif instance == 2:
    #4, 6, 9
    n_min = 4
    n_max = 6
    d_max = 9
    X = [x2_data["X"][0], x2_data["X"][1]]

elif instance == 3:
    #4, 6, 9
    n_min = 4
    n_max = 6
    d_max = 9
    X = [x3_data["X"][0], x3_data["X"][1]]

else:
    print("Error: instance needs to be 1, 2, or 3")

n = len(x1_data["X"][0]) #should be 25
```

In [347]:
```
# create model
myModel = Model( "ClientDepot_MinClusters" )
```

In [348]:
```
# decision variables and parameters

# Y is an assignment matrix:
# Y[i][j] = 1: client i is assigned to depot j, otherwise 0
# each entry of Y is its own decision variable
Y = []
for i in range(n):
    Y.append([])
    for j in range(n):
        Y[i].append(0)
        Y[i][j] = myModel.addVar(lb = 0, ub = 1.0, vtype = GRB.BINARY )


#depots is a vector that indicates whether a depot has clients assigned to
#depots[j] = 0: no clients assigned to depot j.
#          = 1: one or more clients assigned to depot j.
depots = [myModel.addVar(lb = 0, ub = 1.0, vtype = GRB.BINARY ) for j in ra

#parallel (perpendicular?) list to depots, used just for summing
clients = [None for j in range(n)]

# Define a distance matrix to hopefully reduce computational intensity
# D[i][ii] is the distance between clients i and ii
D = []
for i in range(len(clients)):
    D.append([])
    for ii in range(len(clients)):
        D[i].append( math.sqrt((X[0][i] - X[0][ii])**2 + (X[1][i] - X[1][ii
#max(D[1])

#Y = myModel.addVar( vtype = GRB.BINARY, name = "Y" )
myModel.update()
#print(Y)
```

In [349]:
```
# objective: MINIMIZE NUMBER OF CLUSTERS (part a)

objExpr = LinExpr()

for j in range(len(depots)):
    objExpr += depots[j]

myModel.setObjective( objExpr , GRB.MINIMIZE )

print(objExpr)
```

```
<gurobi.LinExpr: C625 + C626 + C627 + C628 + C629 + C630 + C631 + C632 +
C633 + C634 + C635 + C636 + C637 + C638 + C639 + C640 + C641 + C642 + C64
3 + C644 + C645 + C646 + C647 + C648 + C649>
```

In [350]:
```
# create expressions for constraints and add to the model

# FIRST CONSTRAINT: definition of "depots"
#(vector that indicates if a column of Y represents a depot)

firstConst = [LinExpr() for j in depots]
for j in range(len(depots)):

    for i in range(len(clients)):
        firstConst[j] += Y[i][j]/(n+1)

    # 0 < firstConst[j] < 1 if there is at least 1 client assigned to depot

    myModel.addConstr( lhs = depots[j] , sense = GRB.GREATER_EQUAL , rhs =
    #if there are some clients assigned to depot j (nonzero entries in the
    #then depots[j] must be non-zero. since depots is binary, it will take

    myModel.addConstr( lhs = depots[j], sense = GRB.LESS_EQUAL, rhs = first
    # if there is no client assigned to depots[j], then the rhs will be 0.5
    # depots[j] will be forced to be less than 0.5, and the only option is
    # NOTE that this contraint is automatically satisfied if firstConst[j]



#SECOND CONSTRAINT: client-depot uniqueness
#ensures that the sum of each row of Y is exactly 1 (each client assigned t

secondConst = [LinExpr() for i in clients]
for i in range(len(clients)):

    for j in range(len(depots)):
        secondConst[i] += Y[i][j]

    myModel.addConstr( lhs = secondConst[i] , sense = GRB.EQUAL , rhs = 1 )


#THIRD CONSTRAINT: min and max clients constraint

thirdConst = [LinExpr() for j in depots]
for j in range(len(depots)):

    for i in range(len(clients)):
        thirdConst[j] += Y[i][j]

    myModel.addConstr( lhs = thirdConst[j] , sense = GRB.LESS_EQUAL , rhs =
    myModel.addConstr( lhs = thirdConst[j] , sense = GRB.GREATER_EQUAL , rh

# FOURTH CONSTRAINT: max distance constraint

for j in range(len(depots)):

    #compare the distances between all clients in the column

    for i in range(len(depots)):
        for ii in range(len(depots)): #TRIPLE LOOP :)
```

```
            act = 10000*(1 - Y[i][j]) + 10000*(1 - Y[ii][j])
            myModel.addConstr( lhs = D[i][ii], sense = GRB.LESS_EQUAL , rhs

            #if Y[i][j] or Y[ii][j] = 0, the constraint will be satisfied a
            #if they are both 1, then the constraint is "act"ivated,
            #so the distance between that pair must be less than d_max
```

In [351]:
```
# 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 = "testOutput_A.lp" )
# optimize the model
myModel.optimize()
```

```
Gurobi Optimizer version 9.5.0 build v9.5.0rc5 (mac64[rosetta2])
Thread count: 8 physical cores, 8 logical processors, using up to 8 threa
ds
Optimize a model with 15750 rows, 650 columns and 33850 nonzeros
Model fingerprint: 0xba5c7fd6
Variable types: 0 continuous, 650 integer (650 binary)
Coefficient statistics:
  Matrix range     [4e-02, 2e+04]
  Objective range  [1e+00, 1e+00]
  Bounds range     [1e+00, 1e+00]
  RHS range        [5e-01, 2e+04]
Presolve removed 13459 rows and 0 columns
Presolve time: 0.07s
Presolved: 2291 rows, 650 columns, 14555 nonzeros
Variable types: 0 continuous, 650 integer (650 binary)

Root relaxation: objective 5.000000e+00, 2198 iterations, 0.14 seconds
(0.23 work units)

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

H    0     0                       5.0000000    0.00000   100%     -    0
s
     0     0          -    0         5.00000    5.00000  0.00%     -    0
s

Explored 1 nodes (2242 simplex iterations) in 0.23 seconds (0.35 work uni
ts)
Thread count was 8 (of 8 available processors)

Solution count 1: 5

Optimal solution found (tolerance 1.00e-04)
Best objective 5.000000000000e+00, best bound 5.000000000000e+00, gap 0.0
000%
```

In [352]:
```
# print optimal objective and optimal solution
print( "\nOptimal Objective: " + str( myModel.ObjVal ) )

print( "\nOptimal Solution:" )
Y_vals = []
for i in range(len(clients)):
    Y_vals.append([])
    for j in range(len(depots)):
        Y_vals[i].append(int(Y[i][j].x))

for i in range(len(clients)):
    print(Y_vals[i])
```

```
Optimal Objective: 5.0

Optimal Solution:
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
0]
[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
0]
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
0]
[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0]
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
```

```
0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
0]
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0]
[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0]
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0]
[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0]
```

In [357]:

```
# Grab the indices of the clusters:
depots_vals = [int(val.x) for val in depots]

def clusters(assign_matrix):
    clusts = []

    for j in range(len(depots_vals)):

        if depots_vals[j] == 1: #check if the col of Y is a depot

            clusts.append([])
            for i in range(len(clients)):

                if assign_matrix[i][j] == 1:
                     clusts[-1].append(i)

    return clusts

clusters(Y_vals)
```

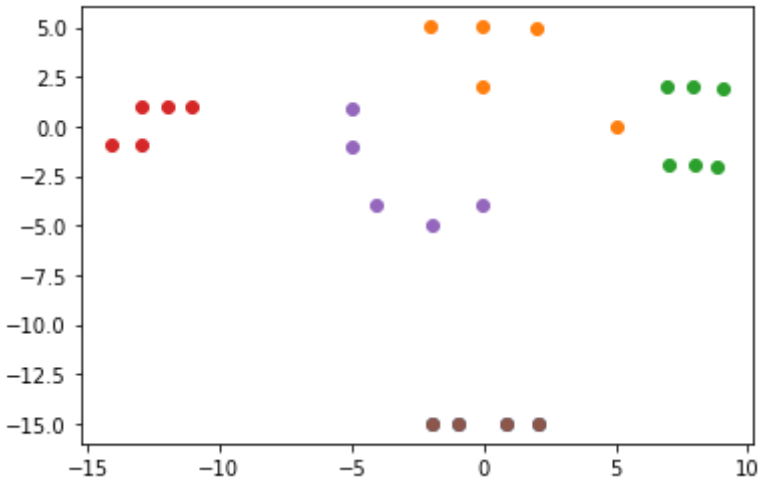
Out[357]:

```
[[2, 14, 22, 24],
 [0, 4, 18, 21, 23],
 [6, 10, 11, 12, 15, 17],
 [1, 3, 5, 7, 13],
 [8, 9, 16, 19, 20]]
```

In [359]:
```
# Create scatterplot


c = clusters(Y_vals)

X_colors_0 = []
for i in range(len(c)):
    X_colors_0.append([])

    for index in c[i]:
        X_colors_0[i].append(X[0][index])


X_colors_1 = []
for i in range(len(c)):
    X_colors_1.append([])

    for index in c[i]:
        X_colors_1[i].append(X[1][index])


for i in range(len(X_colors_1)):

    plt.scatter(X_colors_0[i], X_colors_1[i])

print(c)
plt.scatter(X_colors_0[0], X_colors_1[0])
```

```
[[2, 14, 22, 24], [0, 4, 18, 21, 23], [6, 10, 11, 12, 15, 17], [1, 3, 5,
7, 13], [8, 9, 16, 19, 20]]
```

Out[359]:
```
<matplotlib.collections.PathCollection at 0x7fdc800604c0>
```



In [ ]:

In [267]:
```
# MINIMIZING AVG PAIRWISE DIST
```

In [280]:
```
import sys
print(sys.executable)
!{sys.executable} -m pip install gurobipy
from gurobipy import *
import math
from matplotlib import pyplot as plt
```

```
/Users/Abe/opt/anaconda3/bin/python
Requirement already satisfied: gurobipy in /Users/Abe/opt/anaconda3/lib/p
ython3.8/site-packages (9.5.0)
```

In [269]:
```
#open jld
#https://docs.h5py.org/en/stable/high/group.html
#https://docs.h5py.org/en/stable/high/attr.html
import h5py
x1_data = h5py.File("X1.jld", "r")
x2_data = h5py.File("X2.jld", "r")
x3_data = h5py.File("X3.jld", "r")

#print(x1_data.keys(),x2_data.keys(),x3_data.keys())

# f["input"].value, f["output"].value
# f["X"].value
# f["_creator"]

print(x1_data["X"])
print(x1_data["X"][0])
print(x1_data["X"][1])

#x1_data["_creator"].keys()
```

```
<HDF5 dataset "X": shape (2, 25), type "<f8">
[-9.31431352 -5.66901261 -5.58994131 -6.48529461 -4.64803558 -6.92318026
 -4.9219469  -6.18895012 -6.383588   -4.83052083 -5.54884829 -6.03902456
 -9.36696392 -1.01326107  0.08758034  1.34522284  1.02763538  0.18065355
  0.96876053  2.14521255 -3.81989054 -4.94544952 -3.29144358 -2.36855237
 -4.05178677]
[ 21.85248196  22.82571229  21.70189592  20.44812788 -13.57047711
 -13.45455931 -13.16898973 -15.16087729 -14.11121926 -13.3698081
 -13.74643136 -13.17614957 -13.63194927  -5.99053048  -4.86670161
  -5.53625377  -6.47446421  -4.69903272  -4.01117971  -6.54907345
   1.50498652   1.28581208   2.7638501    2.11314414   1.67859478]
```

In [270]:
```
#Set parameters

instance = 2
# 1 or 2 (3 not feasible)

if instance == 1:
    n_min = 2
    n_max = 8
    d_max = 10
    X = [x1_data["X"][0], x1_data["X"][1]]

elif instance == 2:
    n_min = 4
    n_max = 6
    d_max = 9
    X = [x2_data["X"][0], x2_data["X"][1]]

else:
    print("Error: instance needs to be 1 or 2")

n = len(x1_data["X"][0]) #should be 25
```

In [271]:
```
# Define a distance matrix to hopefully reduce computational intensity
# D[i][ii] is the distance between clients i and ii

D = []
for i in range(len(clients)):
    D.append([])
    for ii in range(len(clients)):
        D[i].append( math.sqrt((X[0][i] - X[0][ii])**2 + (X[1][i] - X[1][ii
max(D[1])
```

Out[271]:
```
22.18458855699755
```

In [272]:
```
# create model
myModel = Model( "ClientDepot_MinDist" )
```

In [273]:
```
# decision variables and parameters

# Y is an assignment matrix:
# Y[i][j] = 1: client i is assigned to depot j, otherwise 0
# each entry of Y is its own decision variable
Y = []
for i in range(n):
    Y.append([])
    for j in range(n):
        Y[i].append(0)
        Y[i][j] = myModel.addVar(lb = 0, ub = 1.0, vtype = GRB.BINARY )


#depots is a vector that indicates whether a depot has clients assigned to
#depots[j] = 0: no clients assigned to depot j.
#          = 1: one or more clients assigned to depot j.
depots = [myModel.addVar(lb = 0, ub = 1.0, vtype = GRB.BINARY ) for j in ra

#parallel (perpendicular?) list to depots, used just for summing
clients = [None for j in range(n)]

# Z is a matrix that indicates whether clients i and ii are in the same dep
Z = []
for i in range(len(clients)):
    Z.append([])
    for ii in range(len(clients)):
        Z[i].append(None)
        Z[i][ii] = myModel.addVar(lb = 0, ub = 1.0, vtype = GRB.CONTINUOUS


#Y = myModel.addVar( vtype = GRB.BINARY, name = "Y" )
myModel.update()
#print(Y)
```

In [274]:
```
# objective: MINIMIZE PARIWISE DISTANCE

# find an expression for total pairwise distance,
# then select only the pairs that are in the same cluster.

objExpr = LinExpr()

for i in range(len(clients)):
    for ii in range(len(clients)):

        objExpr += D[i][ii]*Z[i][ii]/2
        #divide by 2 since this will double count

myModel.setObjective( objExpr , GRB.MINIMIZE )
```

In [275]:
```
# create expressions for constraints and add to the model

#ZEROTH CONSTRAINT: definition of Z
#matrix that indicates whether two clients are in the same depot

for j in range(len(depots)):

    for i in range(len(clients)):
        for ii in range(len(clients)):

            myModel.addConstr( lhs = Z[i][ii] , sense = GRB.LESS_EQUAL , rh
            #Y[i][j] + Y[ii][j] ) / 2
            # if Y[i][j] and Y[ii][j] are not both 1, then Z[i][ii] is cons
            # Otherwise this constraint is satisfied by any value of Z[i][i

            myModel.addConstr( lhs = Z[i][ii] , sense = GRB.GREATER_EQUAL ,
            #Y[i][j] + Y[ii][j] - 1
            # if Y[i][j] and Y[ii][j] are both 1, then Z[i][ii] >= 0.5, so
            # Otherwise this constraint is satisfied by any value of Z[i][i



# FIRST CONSTRAINT: definition of "depots"
#(vector that indicates if a column of Y represents a depot)

firstConst = [LinExpr() for j in depots]
for j in range(len(depots)):

    for i in range(len(clients)):
        firstConst[j] += Y[i][j]/(n+1)

    # 0 < firstConst[j] < 1 if there is at least 1 client assigned to depot

    myModel.addConstr( lhs = depots[j] , sense = GRB.GREATER_EQUAL , rhs =
    #if there are some clients assigned to depot j (nonzero entries in the
    #then depots[j] must be non-zero. since depots is binary, it will take

    myModel.addConstr( lhs = depots[j], sense = GRB.LESS_EQUAL, rhs = first
    # if there is no client assigned to depots[j], then the rhs will be 0.5
    # depots[j] will be forced to be less than 0.5, and the only option is
    # NOTE that this contraint is automatically satisfied if firstConst[j]



#SECOND CONSTRAINT: client-depot uniqueness
#ensures that the sum of each row of Y is exactly 1 (each client assigned t

secondConst = [LinExpr() for i in clients]
for i in range(len(clients)):

    for j in range(len(depots)):
        secondConst[i] += Y[i][j]

    myModel.addConstr( lhs = secondConst[i] , sense = GRB.EQUAL , rhs = 1 )
```

```
#THIRD CONSTRAINT: min and max clients constraint

thirdConst = [LinExpr() for j in depots]
for j in range(len(depots)):

    for i in range(len(clients)):
        thirdConst[j] += Y[i][j]

    myModel.addConstr( lhs = thirdConst[j] , sense = GRB.LESS_EQUAL , rhs =
    myModel.addConstr( lhs = thirdConst[j] , sense = GRB.GREATER_EQUAL , rh

# FOURTH CONSTRAINT: max distance constraint

for j in range(len(depots)):

    #compare the distances between all clients in the column

    for i in range(len(depots)):
        for ii in range(len(depots)): #TRIPLE LOOP :)

            act = 100*(1 - Y[i][j]) + 100*(1 - Y[ii][j]) #100 is about 5x t
            myModel.addConstr( lhs = D[i][ii] , sense = GRB.LESS_EQUAL , rh
            #if Y[i][j] or Y[ii][j] = 0, the constraint will be satisfied a
            #if they are both 1, then the constraint is "act"ivated,
            #so the distance between that pair must be less than d_max
```

In [276]:
```
# 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 = "testOutput_C.lp" )
# optimize the model
myModel.optimize()
```

```
Gurobi Optimizer version 9.5.0 build v9.5.0rc5 (mac64[rosetta2])
Thread count: 8 physical cores, 8 logical processors, using up to 8 threa
ds
Optimize a model with 47000 rows, 1275 columns and 126350 nonzeros
Model fingerprint: 0xd4143991
Variable types: 625 continuous, 650 integer (650 binary)
Coefficient statistics:
  Matrix range     [4e-02, 2e+02]
  Objective range  [4e-01, 1e+01]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e-01, 2e+02]
Presolve removed 40177 rows and 445 columns
Presolve time: 0.14s
Presolved: 6823 rows, 830 columns, 28230 nonzeros
Variable types: 0 continuous, 830 integer (830 binary)

Root relaxation: objective 0.000000e+00, 1094 iterations, 0.06 seconds
(0.13 work units)

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

     0     0    0.00000    0   67          -    0.00000      -     -    0
s
H    0     0                      14.5550193    0.00000   100%     -    0
s
     0     0    0.00000    0  102   14.55502    0.00000   100%     -    0
s
     0     0    0.00000    0  103   14.55502    0.00000   100%     -    0
s
     0     0    0.00000    0   54   14.55502    0.00000   100%     -    0
s
     0     0    0.00000    0   67   14.55502    0.00000   100%     -    1
s
     0     0    0.00000    0   95   14.55502    0.00000   100%     -    1
s
     0     0    0.00000    0   85   14.55502    0.00000   100%     -    1
s
     0     0    0.00000    0   60   14.55502    0.00000   100%     -    1
s
     0     0    0.18602    0   85   14.55502    0.18602  98.7%     -    1
s
     0     0    1.38725    0   64   14.55502    1.38725  90.5%     -    2
s
     0     0    1.38725    0   62   14.55502    1.38725  90.5%     -    2
s
     0     2    1.38725    0   62   14.55502    1.38725  90.5%     -    2
s
H   74    81                      14.5550190    1.38725  90.5%   186    3
```

```
s
H  114   116                      13.4193127    1.38725  89.7%   176    4
s
H  148   151                      13.4193108    1.38725  89.7%   174    4
s
   180   191    1.38725   23  104   13.41931    1.38725  89.7%   183    5
s
H  975   846                      13.4193107    1.38725  89.7%   194    8
s
H 1220  1007                      13.4193102    1.38725  89.7%   202    9
s
  1291  1089    2.41836   15  102   13.41931    1.38725  89.7%   205   10
s
  1478  1194    1.38725   12  103   13.41931    1.38725  89.7%   221   15
s
  1538  1238    1.38725   17   85   13.41931    1.38725  89.7%   260   20
s
H 1554  1186                      13.4193101    1.38725  89.7%   263   20
s
H 1602  1154                      12.7522969    1.38725  89.1%   269   20
s
H 1611  1113                      12.7522948    1.38725  89.1%   271   21
s
H 1656  1090                      12.7522948    1.38725  89.1%   279   22
s
H 1693  1061                      12.3784323    1.38725  88.8%   286   24
s
H 1697  1014                      12.0508771    1.38725  88.5%   290   24
s
  1738  1042    1.38725   28   91   12.05088    1.38725  88.5%   295   25
s
H 1741   997                      12.0508749    1.38725  88.5%   295   25
s
H 1992  1073                      12.0508736    1.38725  88.5%   336   29
s
  2044  1076    1.38725   41   98   12.05087    1.38725  88.5%   348   32
s
H 2085  1059                      12.0508727    1.38725  88.5%   354   33
s
  2142  1112    1.38725   47  102   12.05087    1.38725  88.5%   367   35
s
H 2357  1118                      12.0508726    1.38725  88.5%   396   38
s
  2476  1187    3.12024   64  111   12.05087    1.38725  88.5%   395   40
s
  2768  1327 infeasible   86        12.05087    1.38725  88.5%   428   45
s
  3026  1419    1.38725   97  108   12.05087    1.38725  88.5%   443   50
s
  3307  1541    1.38725  116  102   12.05087    1.38725  88.5%   479   55
s
  3494  1668    1.38725  125  123   12.05087    1.38725  88.5%   505   60
s
  4000  2022    1.69815  136  128   12.05087    1.69645  85.9%   502   65
s
  4602  2290    1.91253  145  123   12.05087    1.69645  85.9%   481   70
s
```

```
  4881  2582    3.83842  155  262   12.05087    1.69645  85.9%   491   76
s
  5418  2957     cutoff  162        12.05087    1.96553  83.7%   488   81
s

Explored 5872 nodes (2863955 simplex iterations) in 83.62 seconds (203.64
work units)
Thread count was 8 (of 8 available processors)

Solution count 10: 12.0509 12.0509 12.0509 ... 13.4193

Solve interrupted
Best objective 1.205081919215e+01, best bound 2.013258635713e+00, gap 83.
2936%
```

In [277]:
```
# print optimal objective and optimal solution
print( "\nOptimal Objective: " + str( myModel.ObjVal ) )

print( "\nOptimal Solution:" )
Y_vals = []
for i in range(len(clients)):
    Y_vals.append([])
    for j in range(len(depots)):
        Y_vals[i].append(int(Y[i][j].x))

for i in range(len(clients)):
    print(Y_vals[i])
```

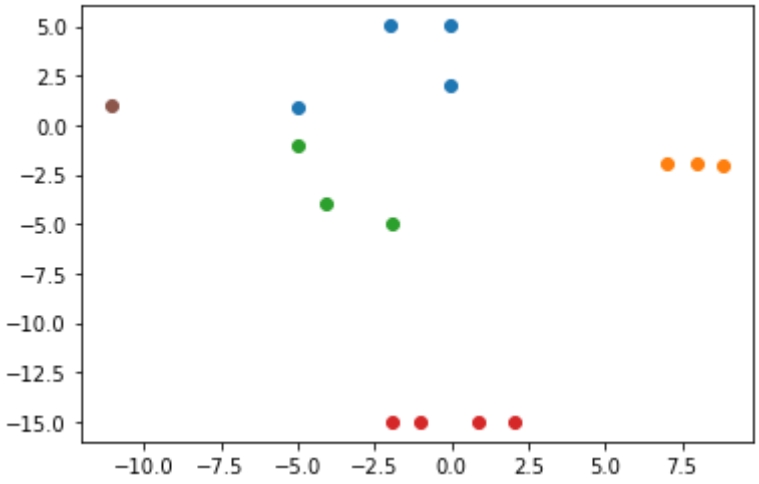
```
Optimal Objective: 12.050819192152407

Optimal Solution:
[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0]
[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0]
[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0]
[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0]
[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0]
[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0]
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
0]
[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
0]
```

In [278]:
```
# Grab the indices of the clusters:
depots_vals = [int(val.x) for val in depots]

def clusters(assign_matrix):
    clusts = []

    for j in range(len(depots_vals)):

        if depots_vals[j] == 1: #check if the col of Y is a depot

            clusts.append([])
            for i in range(len(clients)):

                if assign_matrix[i][j] == 1:
                    clusts[-1].append(i)

    return clusts

clusters(Y_vals)
```

Out[278]:
```
[[0, 9, 18, 23], [10, 11, 17], [16, 19, 20], [2, 14, 22, 24], [], [5]]
```

In [281]:
```
# Create scatterplot


c = clusters(Y_vals)

X_colors_0 = []
for i in range(len(c)):
    X_colors_0.append([])

    for index in c[i]:
        X_colors_0[i].append(X[0][index])


X_colors_1 = []
for i in range(len(c)):
    X_colors_1.append([])

    for index in c[i]:
        X_colors_1[i].append(X[1][index])


for i in range(len(X_colors_1)):

    plt.scatter(X_colors_0[i], X_colors_1[i])
```



In [ ]: