THE UNCAPACITATED FIXEDCHARGE LOCATION PROBLEM (UFLP)

8.2 (10-Node UFLP Instance: Exact) The file `10node.xlsx` contains data for a 10-node instance of the UFLP, with nodes located on the unit square and $I = J$, pictured in Figure 8.22. The file lists the x- and y-coordinates, demands h_i, and fixed costs f_j for each node, as well as the transportation cost c_{ij} between each pair of nodes i and j. Transportation costs equal 10 times the Euclidean distance between the nodes. All fixed costs equal 200.

Solve this instance of the UFLP exactly by implementing the UFLP in the modeling language of your choice and solving it with a MIP solver. Report the optimal locations, optimal assignments, and optimal cost.

Define the following notation:

Sets

I = set of customers

J = set of potential facility locations

Parameters

h_i = annual demand of customer $i \in I$

c_{ij} = cost to transport one unit of demand from facility $j \in J$ to customer $i \in I$

f_j = fixed annual cost to open a facility at site $j \in J$

Decision Variables

x_j = 1 if facility j is opened, 0 otherwise

y_{ij} = the fraction of customer i's demand that is served by facility j

The UFLP is formulated as follows:

$$\text{(UFLP)} \quad \text{minimize} \quad \sum_{j \in J} f_j x_j + \sum_{i \in I} \sum_{j \in J} h_i c_{ij} y_{ij} \tag{8.3}$$

$$\text{subject to} \quad \sum_{j \in J} y_{ij} = 1 \qquad \forall i \in I \tag{8.4}$$

$$y_{ij} \leq x_j \qquad \forall i \in I, \forall j \in J \tag{8.5}$$

$$x_j \in \{0, 1\} \qquad \forall j \in J \tag{8.6}$$

$$y_{ij} \geq 0 \qquad \forall i \in I, \forall j \in J \tag{8.7}$$

```
import pandas as pd
from pulp import *

#SETS
CUSTOMERS = list(range(1,11))
FACILITY = list(range(1,11))

#PARAMETERS
demand = {1: 60, 2: 27, 3: 29, 4: 26, 5: 33,
          6: 15, 7: 17, 8: 97, 9: 97, 10: 19}

fixedcost = {1: 200, 2: 200, 3: 200, 4: 200, 5: 200,
             6: 200, 7: 200, 8: 200, 9: 200, 10: 200}

transp = {1 : {1: 0.0, 2: 92.195, 3: 30.0, 4: 70.711, 5: 85.44,
               6: 44.721, 7: 67.082, 8: 36.056, 9: 50.99, 10: 50},
          2: {1: 92.195, 2: 0.0, 3: 76.158, 4: 50, 5: 44.721,
              6: 50, 7: 31.623, 8: 56.569, 9: 60.828, 10: 70.711},
          3: {1: 30, 2: 76.158, 3: 0.0, 4: 72.801, 5: 58.31,
              6: 41.231, 7: 60, 8: 31.623, 9: 22.361, 10: 20},
          4: {1: 70.711, 2: 50, 3: 72.801, 4: 0.0, 5: 80.623,
              6: 31.623, 7: 22.361, 8: 41.231, 9: 72.111, 10: 80.623},
          5: {1: 85.44, 2: 44.721, 3: 58.31, 4: 80.623, 5: 0.0,
              6: 60.828, 7: 58.31, 8: 60, 9: 36.056, 10: 42.426},
          6: {1: 44.721, 2: 50, 3: 41.231, 4: 31.623, 5: 60.828,
              6: 0.0, 7: 22.361, 8: 10, 9: 42.426, 10: 50},
          7: {1: 67.082, 2: 31.623, 3: 60, 4: 22.361, 5: 58.31,
              6: 22.361, 7: 0.0, 8: 31.623, 9: 53.852, 10: 63.246},
          8: {1: 36.056, 2: 56.569, 3: 31.623, 4: 41.231, 5: 60,
              6: 10, 7: 31.623, 8: 0.0, 9: 36.056, 10: 42.426},
          9: {1: 50.99, 2: 60.828, 3: 22.361, 4: 72.111, 5: 36.056,
              6: 42.426, 7: 53.852, 8: 36.056, 9: 0.0, 10: 10},
          10: {1: 50, 2: 70.711, 3: 20, 4: 80.623, 5: 42.426,
               6: 50, 7: 63.246, 8: 42.426, 9: 10, 10: 0.0}
         }


#SET PROBLEM VARIABLE
prob = LpProblem("FacilityLocation", LpMinimize)

#DECISION VARIABLE
serv_vars = LpVariable.dicts("Service",
                             [(i,j) for i in CUSTOMERS for j in FACILITY],
                             0)

use_vars = LpVariable.dicts("UseLocation", FACILITY, 0, 1, LpBinary)

#OBJECTIVE FUNCTION
prob += lpSum(fixedcost[j]*use_vars[j] for j in FACILITY) + lpSum(demand[i]*transp[j][i]*serv_vars[(i,j)]
                                                                  for j in FACILITY for i in CUSTOMERS)
```

```
# CONSTRAINTS
for i in CUSTOMERS:
    prob += lpSum(serv_vars[(i,j)] for j in FACILITY) == 1

for i in CUSTOMERS:
    for j in FACILITY:
        prob += serv_vars[(i,j)] <= use_vars[j]

for i in CUSTOMERS:
    for j in FACILITY:
        prob += serv_vars[(i,j)] >= 0


#SOLUTION
prob.solve()
print("Status: ", LpStatus[prob.status])

# PRINT DECISION VARIABLES
TOL = .0001 #tolerance for identifying which locations the algorithm has chosen
for i in FACILITY:
  if use_vars[i].varValue > TOL:
      print("Establish facility at site ", i)

for v in prob.variables():
  print(v.name, " = ", v.varValue)

#PRINT OPTIMAL SOLUTION
print("The cost of production in dollars for one year: ", value(prob.objective))
```

Solution:

Optimal Facility locations:

```
Establish facility at site  1
Establish facility at site  2
Establish facility at site  3
Establish facility at site  4
Establish facility at site  5
Establish facility at site  7
Establish facility at site  8
Establish facility at site  9
```

Optimal Assignments & Cost

```
Service_(1,_1)  =  1.0    Service_(5,_1)  =  0.0
Service_(1,_10)  =  0.0   Service_(5,_10)  =  0.0
Service_(1,_2)  =  0.0    Service_(5,_2)  =  0.0
Service_(1,_3)  =  0.0    Service_(5,_3)  =  0.0
Service_(1,_4)  =  0.0    Service_(5,_4)  =  0.0
Service_(1,_5)  =  0.0    Service_(5,_5)  =  1.0
Service_(1,_6)  =  0.0    Service_(5,_6)  =  0.0
Service_(1,_7)  =  0.0    Service_(5,_7)  =  0.0
Service_(1,_8)  =  0.0    Service_(5,_8)  =  0.0
Service_(1,_9)  =  0.0    Service_(5,_9)  =  0.0
Service_(10,_1)  =  0.0   Service_(6,_1)  =  0.0
Service_(10,_10)  =  0.0  Service_(6,_10)  =  0.0
Service_(10,_2)  =  0.0   Service_(6,_2)  =  0.0
Service_(10,_3)  =  0.0   Service_(6,_3)  =  0.0
Service_(10,_4)  =  0.0   Service_(6,_4)  =  0.0
Service_(10,_5)  =  0.0   Service_(6,_5)  =  0.0
Service_(10,_6)  =  0.0   Service_(6,_6)  =  0.0
Service_(10,_7)  =  0.0   Service_(6,_7)  =  0.0
Service_(10,_8)  =  0.0   Service_(6,_8)  =  1.0
Service_(10,_9)  =  1.0   Service_(6,_9)  =  0.0
Service_(2,_1)  =  0.0    Service_(7,_1)  =  0.0
Service_(2,_10)  =  0.0   Service_(7,_10)  =  0.0
Service_(2,_2)  =  1.0    Service_(7,_2)  =  0.0
Service_(2,_3)  =  0.0    Service_(7,_3)  =  0.0
Service_(2,_4)  =  0.0    Service_(7,_4)  =  0.0
Service_(2,_5)  =  0.0    Service_(7,_5)  =  0.0
Service_(2,_6)  =  0.0    Service_(7,_6)  =  0.0
Service_(2,_7)  =  0.0    Service_(7,_7)  =  1.0
Service_(2,_8)  =  0.0    Service_(7,_8)  =  0.0
Service_(2,_9)  =  0.0    Service_(7,_9)  =  0.0
Service_(3,_1)  =  0.0    Service_(8,_1)  =  0.0
Service_(3,_10)  =  0.0   Service_(8,_10)  =  0.0
Service_(3,_2)  =  0.0    Service_(8,_2)  =  0.0
Service_(3,_3)  =  1.0    Service_(8,_3)  =  0.0
Service_(3,_4)  =  0.0    Service_(8,_4)  =  0.0
Service_(3,_5)  =  0.0    Service_(8,_5)  =  0.0
Service_(3,_6)  =  0.0    Service_(8,_6)  =  0.0
Service_(3,_7)  =  0.0    Service_(8,_7)  =  0.0
Service_(3,_8)  =  0.0    Service_(8,_8)  =  1.0
Service_(3,_9)  =  0.0    Service_(8,_9)  =  0.0
Service_(4,_1)  =  0.0    Service_(9,_1)  =  0.0
Service_(4,_10)  =  0.0   Service_(9,_10)  =  0.0
Service_(4,_2)  =  0.0    Service_(9,_2)  =  0.0
Service_(4,_3)  =  0.0    Service_(9,_3)  =  0.0
Service_(4,_4)  =  1.0    Service_(9,_4)  =  0.0
Service_(4,_5)  =  0.0    Service_(9,_5)  =  0.0
Service_(4,_6)  =  0.0    Service_(9,_6)  =  0.0
Service_(4,_7)  =  0.0    Service_(9,_7)  =  0.0
Service_(4,_8)  =  0.0    Service_(9,_8)  =  0.0
Service_(4,_9)  =  0.0    Service_(9,_9)  =  1.0

UseLocation_1  =  1.0
UseLocation_10  =  0.0
UseLocation_2  =  1.0
UseLocation_3  =  1.0
UseLocation_4  =  1.0
UseLocation_5  =  1.0
UseLocation_6  =  0.0
UseLocation_7  =  1.0
UseLocation_8  =  1.0
UseLocation_9  =  1.0
The cost of production in dollars for one year 1940.0
```

The Set Covering Location Problem (SCLP)

8.8 (10-node SCLP Instance) Using the file `10node.xlsx` (see Problem 8.2), solve the SCLP exactly by implementing it in the modeling language of your choice and solving it with a MIP solver. Set the fixed cost of every facility equal to 1. Assume that facility j covers customer i if $c_{ij} \leq 2.5$. Report the optimal locations.

In the *set covering location problem* (SCLP), we are required to cover *every* demand node; the objective is to do so with the fewest possible number of facilities. The SCLP was first formulated in a facility location context by Hakimi (1965), though similar models appeared in graph-theoretic settings prior to that.

In addition to the notation introduced in earlier sections, we use the following new notation:

Parameters

$a_{ij} = 1$ if facility $j \in J$ can cover customer $i \in I$ (if it is open), 0 otherwise

The coverage parameter a_{ij} can be derived from a distance or cost parameter such as c_{ij} in the UFLP, for example:

$$a_{ij} = \begin{cases} 1, & \text{if } c_{ij} \leq r \\ 0, & \text{otherwise} \end{cases}$$

for a fixed coverage radius r. Or a_{ij} can be derived in other ways that are unrelated to distance, especially in the nonlocation applications of the SCLP discussed below.

The SCLP can be formulated as follows:

$$\text{(SCLP)} \quad \text{minimize} \quad \sum_{j \in J} x_j \tag{8.80}$$

$$\text{subject to} \quad \sum_{j \in J} a_{ij} x_j \geq 1 \qquad \forall i \in I \tag{8.81}$$

$$x_j \in \{0, 1\} \qquad \forall j \in J \tag{8.82}$$

```
import numpy as np
import pandas as pd
from pulp import *

#PARAMETERS
dist = [[0, 9.2195, 3, 7.0711, 8.544 , 4.4721, 6.7082, 3.6056, 5.099 , 5],
       [9.2195, 0, 7.6158, 5 , 4.4721, 5, 3.1623, 5.6569, 6.0828, 7.0711],
       [3, 7.6158, 0, 7.2801, 5.831, 4.1231, 6, 3.1623, 2.2361, 2],
       [7.0711, 5, 7.2801, 0, 8.0623, 3.1623, 2.2361, 4.1231, 7.2111, 8.0623],
       [8.544, 4.4721, 5.831 , 8.0623, 0, 6.0828, 5.831 , 6, 3.6056, 4.2426],
       [4.4721, 5, 4.1231, 3.1623, 6.0828, 0, 2.2361, 1, 4.2426, 5],
       [6.7082, 3.1623, 6, 2.2361, 5.831, 2.2361, 0, 3.1623, 5.3852, 6.3246],
       [3.6056, 5.6569, 3.1623, 4.1231, 6, 1, 3.1623, 0, 3.6056, 4.2426],
       [5.099, 6.0828, 2.2361, 7.2111, 3.6056, 4.2426, 5.3852, 3.6056, 0, 1],
       [5, 7.0711, 2, 8.0623, 4.2426, 5, 6.3246, 4.2426, 1, 0]]

Aij = np.zeros((10,10))
for i in range(len(dist)):
    for j in range(len(dist[0])):
        if dist[i][j] <= 2.5:
            Aij[i][j] = 1
        else:
            Aij[i][j] = 0

#SETS
CUSTOMERS = list(range(1,11))
FACILITY = list(range(1,11))

#SET PROBLEM VARIABLE
prob = LpProblem("FacilityLocation", LpMinimize)

#DECISION VARIABLE
use_vars = LpVariable.dicts("UseLocation", FACILITY, 0, 1, LpBinary)

#OBJECTIVE FUNCTION
prob += lpSum(use_vars[j] for j in FACILITY)

# CONSTRAINTS
for i in CUSTOMERS:
    prob += lpSum(Aij[i-1][j-1]*use_vars[j] for j in FACILITY) >= 1

#SOLUTION
prob.solve()
print("Status: ", LpStatus[prob.status])

# PRINT DECISION VARIABLES
TOL = .0001 #tolerance for identifying which locations the algorithm has chosen
for i in FACILITY:
  if use_vars[i].varValue > TOL:
      print("Establish facility at site ", i)

for v in prob.variables():
  print(v.name, " = ", v.varValue)

#PRINT OPTIMAL SOLUTION
print("Optimal number of locations: ", value(prob.objective))
```

Solution:

```
                                UseLocation_1  =  1.0
                                UseLocation_10  =  0.0
                                UseLocation_2  =  1.0
                                UseLocation_3  =  0.0
                                UseLocation_4  =  1.0
Establish facility at site  1   UseLocation_5  =  1.0
Establish facility at site  2   UseLocation_6  =  0.0
Establish facility at site  4   UseLocation_7  =  0.0
Establish facility at site  5   UseLocation_8  =  1.0
Establish facility at site  8   UseLocation_9  =  1.0
Establish facility at site  9   Optimal number of locations:  6.0
```

The Maximal Covering Location Problem (MCLP)

8.9 (10-node MCLP Instance) Using the file `10node.xlsx` (see Problem 8.2), solve the MCLP exactly by implementing it in the modeling language of your choice and solving it with a MIP solver. Set $p = 4$. Assume that facility j covers customer i if $c_{ij} \leq 2.5$. Report the optimal locations and the total number of demands covered.

The SCLP requires every customer to be covered by an open facility. Sometimes this is impractical, because complete coverage would require opening too many facilities. For example, it takes 10 facilities to cover 100% of the demand in the 88-node data set with a 400-mile coverage radius. (See Example 8.6.) But we know from Example 8.7 that we can cover 88.7% of the demand with only six facilities. In fact, if we are only allowed six facilities, we can do better than 88.7%, as we will see below.

The *maximal covering location problem* (MCLP) seeks to maximize the total number of demands covered subject to a limit on the number of open facilities. It was introduced by Church and ReVelle (1974). It uses the same notation as the SCLP, plus the usual parameter p that specifies the allowable number of facilities, as well as a new set of decision variables:

Decision Variables

z_i = 1 if customer $i \in I$ is covered by an open facility, 0 otherwise

The MCLP can be formulated as follows:

$$\text{(MCLP)} \quad \text{maximize} \quad \sum_{i \in I} h_i z_i \tag{8.85}$$

$$\text{subject to} \quad z_i \leq \sum_{j \in J} a_{ij} x_j \qquad \forall i \in I \tag{8.86}$$

$$\sum_{j \in J} x_j = p \tag{8.87}$$

$$x_j \in \{0, 1\} \qquad \forall j \in J \tag{8.88}$$

$$z_i \in \{0, 1\} \qquad \forall i \in I \tag{8.89}$$

```
import numpy as np
import pandas as pd
from pulp import *

#PARAMETERS
demand = {1: 60, 2: 27, 3: 29, 4: 26, 5: 33,
          6: 15, 7: 17, 8: 97, 9: 97, 10: 19}

dist = [[0, 9.2195, 3, 7.0711, 8.544 , 4.4721, 6.7082, 3.6056, 5.099 , 5],
       [9.2195, 0, 7.6158, 5 , 4.4721, 5, 3.1623, 5.6569, 6.0828, 7.0711],
       [3, 7.6158, 0, 7.2801, 5.831, 4.1231, 6, 3.1623, 2.2361, 2],
       [7.0711, 5, 7.2801, 0, 8.0623, 3.1623, 2.2361, 4.1231, 7.2111, 8.0623],
       [8.544, 4.4721, 5.831 , 8.0623, 0, 6.0828, 5.831 , 6, 3.6056, 4.2426],
       [4.4721, 5, 4.1231, 3.1623, 6.0828, 0, 2.2361, 1, 4.2426, 5],
       [6.7082, 3.1623, 6, 2.2361, 5.831, 2.2361, 0, 3.1623, 5.3852, 6.3246],
       [3.6056, 5.6569, 3.1623, 4.1231, 6, 1, 3.1623, 0, 3.6056, 4.2426],
       [5.099, 6.0828, 2.2361, 7.2111, 3.6056, 4.2426, 5.3852, 3.6056, 0, 1],
       [5, 7.0711, 2, 8.0623, 4.2426, 5, 6.3246, 4.2426, 1, 0]]

Aij = np.zeros((10,10))
for i in range(len(dist)):
    for j in range(len(dist[0])):
        if dist[i][j] <= 2.5:
            Aij[i][j] = 1
        else:
            Aij[i][j] = 0


#SETS
CUSTOMERS = list(range(1,11))
FACILITY = list(range(1,11))

#SET PROBLEM VARIABLE
prob = LpProblem("FacilityLocation", LpMaximize)

#DECISION VARIABLE
use_vars = LpVariable.dicts("UseCustomers", CUSTOMERS, 0, 1, LpBinary)

use_facs = LpVariable.dicts("UseLocation", FACILITY, 0, 1, LpBinary)

#OBJECTIVE FUNCTION
prob += lpSum(demand[j]*use_vars[j] for j in CUSTOMERS)

# CONSTRAINTS
for i in CUSTOMERS:
    prob += lpSum(Aij[i-1][j-1]*use_facs[j] for j in FACILITY) >= use_vars[i]

prob += lpSum(use_facs[j] for j in FACILITY) == 4

#SOLUTION
prob.solve()
print("Status: ", LpStatus[prob.status])

# PRINT DECISION VARIABLES
TOL = .0001 #tolerance for identifying which locations the algorithm has chosen
for i in FACILITY:
  if use_vars[i].varValue > TOL:
      print("Customers whose demands are covered ", i)

for v in prob.variables():
  print(v.name, " = ", v.varValue)

#PRINT OPTIMAL SOLUTION
print("Total amount of demands covered: ", value(prob.objective))
```

Solution:

```
                                        UseCustomers_1  =  1.0
                                        UseCustomers_10  =  1.0
                                        UseCustomers_2  =  0.0
                                        UseCustomers_3  =  1.0
                                        UseCustomers_4  =  0.0
                                        UseCustomers_5  =  1.0
                                        UseCustomers_6  =  1.0
                                        UseCustomers_7  =  1.0
                                        UseCustomers_8  =  1.0
                                        UseCustomers_9  =  1.0
                                        UseLocation_1  =  1.0
                                        UseLocation_10  =  0.0
                                        UseLocation_2  =  0.0
Customers whose demands are covered  1  UseLocation_3  =  0.0
Customers whose demands are covered  3  UseLocation_4  =  0.0
Customers whose demands are covered  5  UseLocation_5  =  1.0
Customers whose demands are covered  6  UseLocation_6  =  1.0
Customers whose demands are covered  7  UseLocation_7  =  0.0
Customers whose demands are covered  8  UseLocation_8  =  0.0
Customers whose demands are covered  9  UseLocation_9  =  1.0
Customers whose demands are covered  10 Total amount of demands covered:  367.0
```