INDR371 - HW5

Umur Berkay Karakaş

December 19, 2022

Question 1

Part A

```
In [24]: a = [2,4,3,1,2,4,1,5]
         b = [3,4,5,3,6,6,5,5]
         W = [5,10,8,7,6,10,6,5]

         m = gp.Model("q1_parta")
         m.ModelSense = GRB.MINIMIZE
         x = m.addVar(name="x",vtype=GRB.CONTINUOUS)
         y = m.addVar(name="y",vtype=GRB.CONTINUOUS)
         r = m.addVars(range(len(a)), name="r",vtype=GRB.CONTINUOUS)
         s = m.addVars(range(len(b)), name="s",vtype=GRB.CONTINUOUS)
         m.addConstrs(r[i] >= x - a[i] for i in range(len(a)))
         m.addConstrs(r[i] >= a[i] - x for i in range(len(a)))
         m.addConstrs(s[i] >= y - b[i] for i in range(len(b)))
         m.addConstrs(s[i] >= b[i] - y for i in range(len(b)))
         m.setObjective(gp.quicksum(W[i]*(r[i]+s[i]) for i in range(len(W))))

In [25]: m.optimize()
```

(a) Code

```
In [3]: m.optimize()

        Gurobi Optimizer version 9.5.2 build v9.5.2rc0 (mac64[arm])
        Thread count: 8 physical cores, 8 logical processors, using up to 8 threads
        Optimize a model with 32 rows, 18 columns and 64 nonzeros
        Model fingerprint: 0xcba38835
        Coefficient statistics:
          Matrix range     [1e+00, 1e+00]
          Objective range  [5e+00, 1e+01]
          Bounds range     [0e+00, 0e+00]
          RHS range        [1e+00, 6e+00]
        Presolve time: 0.00s
        Presolved: 32 rows, 18 columns, 64 nonzeros

        Iteration    Objective       Primal Inf.    Dual Inf.      Time
               0    0.0000000e+00   5.900000e+01   0.000000e+00      0s
              14    1.1700000e+02   0.000000e+00   0.000000e+00      0s

        Solved in 14 iterations and 0.01 seconds (0.00 work units)
        Optimal objective  1.170000000e+02

In [4]: print(x.X)
        print(y.X)

        3.0
        5.0
```

(b) Output

Figure 1: Code and output of Question 1, Part A

In minisum algorithm, there are absolute valued terms. So I had to convert them into linear first. In order to do that, I added 2 variables r and s and added 2 constraints for each, to ensure that $r_i = |x - a_i|$ and $s_i = |y - b_i|$. Then, I wrote r and s for absolute valued term in the objective function.

The objective function is 117 and the optimal location for the repair shop is (3,5).

Part B

```
In [27]: a.append(x.X)
         b.append(y.X)
         W.append(np.mean(W))
         m = gp.Model("q1_partb")
         m.ModelSense = GRB.MINIMIZE
         x = m.addVar(name="x",vtype=GRB.CONTINUOUS)
         y = m.addVar(name="y",vtype=GRB.CONTINUOUS)
         c = m.addVar(name="c",vtype=GRB.CONTINUOUS)
         r = m.addVars(range(len(a)), name="r",vtype=GRB.CONTINUOUS)
         s = m.addVars(range(len(b)), name="s",vtype=GRB.CONTINUOUS)
         m.addConstrs(r[i] >= x - a[i] for i in range(len(a)))
         m.addConstrs(r[i] >= a[i] - x for i in range(len(a)))
         m.addConstrs(s[i] >= y - b[i] for i in range(len(b)))
         m.addConstrs(s[i] >= b[i] - y for i in range(len(b)))
         m.addConstrs(c >= W[i]*(r[i] + s[i]) for i in range(len(a)))
         m.setObjective(c)

In [28]: m.optimize()
```

(a) Code

```
In [28]: m.optimize()

         Gurobi Optimizer version 9.5.2 build v9.5.2rc0 (mac64[arm])
         Thread count: 8 physical cores, 8 logical processors, using up to 8 threads
         Optimize a model with 45 rows, 21 columns and 99 nonzeros
         Model fingerprint: 0xb5f01c80
         Coefficient statistics:
           Matrix range     [1e+00, 1e+01]
           Objective range  [1e+00, 1e+00]
           Bounds range     [0e+00, 0e+00]
           RHS range        [1e+00, 6e+00]
         Presolve removed 1 rows and 1 columns
         Presolve time: 0.01s
         Presolved: 44 rows, 20 columns, 104 nonzeros

         Iteration    Objective       Primal Inf.    Dual Inf.      Time
                0    0.0000000e+00   6.700000e+01   0.000000e+00      0s
               14    2.4705882e+01   0.000000e+00   0.000000e+00      0s

         Solved in 14 iterations and 0.01 seconds (0.00 work units)
         Optimal objective  2.470588235e+01

In [30]: print(x.X)
         print(y.X)

         2.529411764705882
         5.0
```

(b) Output

Figure 2: Code and output of Question 1, Part B

Also in minimax algorithm, there are absolute valued terms. So I had to convert them into linear first. In order to do that, I added 2 variables r and s and added 2 constraints for each, to ensure that $r_i = |x - a_i|$ and $s_i = |y - b_i|$. Then, I defined a new variable c and added a constraint $c \geq W_i * (r_i + s_i)$ for each point and set the objective function to be "c", so that we can minimize the maximum distance to a point. Additionally, I set the weight of the repair shop equal to the average of the weight of the facilities.

The objective function is 24.706 and the optimal location for the fire station is (2.53,5).

Part C

- z_{ij}: whether fire station i is assigned to facility j or not
- x_i: x coordinate of fire station i
- y_i: y coordinate of fire station i
- a_j: x coordinate of facility j
- b_j: y coordinate of facility j
- r_{ij}: $|x_i - a_j|$
- s_{ij}: $|y_i - b_j|$
- W_j: weight of facility j

$$
\begin{aligned}
\min \quad & c_1 + c_2 && && \text{(1a)}\\
\text{s.t.} \quad & \sum_{i=1}^{2} z_{ij} = 1 && \forall j && \text{(1b)}\\
& r_{ij} \geq x_i - a_j && \forall i, j && \text{(1c)}\\
& r_{ij} \geq a_j - x_i && \forall i, j && \text{(1d)}\\
& s_{ij} \geq y_i - b_j && \forall i, j && \text{(1e)}\\
& s_{ij} \geq b_j - y_i && \forall i, j && \text{(1f)}\\
& c_i \geq z_{ij} * W_j (r_{ij} + s_{ij}) && \forall i, j && \text{(1g)}\\
& x_i, y_i \in \mathbb{Z} && \forall i && \text{(1h)}\\
& z_{ij} \in \{0, 1\} && \forall i, j && \text{(1i)}
\end{aligned}
$$

- Constraint 1b is for the assignment of 1 fire station to each facility.
- Constraints 1c, 1d, 1e and 1f are for the linearization of absolute valued terms.
- Constraint 1g is to keep the maximum distance between each fire station and the facilities assigned to it.
- Constraints 1h and 1i are for the domains of decision variables.

```
In [8]: m = gp.Model("q1_partc")
m.ModelSense = GRB.MINIMIZE
x = m.addVars(range(2), name="x",vtype=GRB.INTEGER)
y = m.addVars(range(2), name="y",vtype=GRB.INTEGER)
z = m.addVars(range(2), range(len(a)), name="z", vtype=GRB.BINARY)
r = m.addVars(range(2), range(len(a)), name="r",vtype=GRB.CONTINUOUS)
s = m.addVars(range(2), range(len(a)), name="s",vtype=GRB.CONTINUOUS)
c = m.addVars(range(2), name="c", vtype=GRB.CONTINUOUS)

m.addConstrs((gp.quicksum(z[i,j] for i in range(2)) == 1) for j in range(len(a)))
m.addConstrs(r[i,j] >= x[i] - a[j] for i in range(2) for j in range(len(a)))
m.addConstrs(r[i,j] >= a[j] - x[i] for i in range(2) for j in range(len(a)))
m.addConstrs(s[i,j] >= y[i] - b[j] for i in range(2) for j in range(len(b)))
m.addConstrs(s[i,j] >= b[j] - y[i] for i in range(2) for j in range(len(b)))
m.addConstrs(c[i] >= z[i,j]*W[j]*(r[i,j] + s[i,j]) for i in range(2) for j in range(len(a)))
m.setObjective(gp.quicksum(c[i] for i in range(2)))
```

(a) Code

```
In [9]: m.optimize()
     0     0   15.72917    0   32   20.00000   15.72917  21.4%     -    0s
     0     0   16.66667    0   31   20.00000   16.66667  16.7%     -    0s
     0     0   16.66667    0   33   20.00000   16.66667  16.7%     -    0s

Cutting planes:
  Gomory: 3
  MIR: 17
  StrongCG: 3
  Flow cover: 4
  Mod-K: 3
  RLT: 14

Explored 1 nodes (134 simplex iterations) in 0.03 seconds (0.01 work units)
Thread count was 8 (of 8 available processors)

Solution count 9: 20 24 25 ... 62

Optimal solution found (tolerance 1.00e-04)
Best objective 2.000000000000e+01, best bound 2.000000000000e+01, gap 0.0000%

In [10]: print("First fire station location:  ({},{})".format(x[0].X, y[0].X))
print("Second fire station location: ({},{})".format(x[1].X, y[1].X))

First fire station location:  (3.0,5.0)
Second fire station location: (1.0,3.0)
```

(b) Output

Figure 3: Code and output of Question 1, Part C

The objective function is 20 and the optimal locations for the fire stations are (1,3) and (3,5). The objective function in part C is expected to be less than part B since we do not consider the fixed cost of opening fire stations, therefore opening as much fire stations as possible would decrease the cost.

Question 2

Part A

- x_{ir}: whether facility i is located at location r or not
- r_{ij}: flow between facility i and j
- d_{rs}: distance between location r and s

$$\begin{aligned}
\min \quad & \sum_{i=1}^{5} \sum_{j=1, i \neq j}^{5} \sum_{r=1}^{12} \sum_{s=1, r \neq s}^{12} (r_{ij} * d_{rs} + r_{ji} * d_{sr}) * x_{ir} * x_{js} && && \text{(2a)} \\
\text{s.t.} \quad & \sum_{i=1}^{5} x_{ir} \leq 1 && \forall r && \text{(2b)} \\
& \sum_{r=1}^{12} x_{ir} = 1 && \forall i && \text{(2c)} \\
& \sum_{i=1}^{5} \sum_{j=1}^{5} d_{ij} * x_{(D_1,i)} * x_{(D_4,j)} \geq 2 && && \text{(2d)} \\
& x_{ir} = 0 && \forall i \in \{D_1, D_2, D_5\}, \forall r \in \{A, D, L, I\} && \text{(2e)} \\
& x_{(D_3),r} = 0 && \forall r \notin A, D, L, I && \text{(2f)} \\
& x_{ir} \in \{0, 1\} && \forall i, r && \text{(2g)}
\end{aligned}$$

- Constraint 2b is for that at most one facility can be assigned to a single grid.
- Constraint 2c is for that a facility must be assigned to a location.
- Constraint 2d is for that facilities D_1 and D_4 must not share an edge i.e. the distances between locations of those facilities should be at least 2.
- Constraint 2e is for that D_1, D_2, and D_5 cannot be placed in grids A,D,L or I.
- Constraint 2f is for that D_3 cannot be placed in grid locations other than A,D,L and I.
- Constraint 2g is for the binary domain of x_{ir}.

Part B

```
In [11]: locations = list(map(chr, range(65, 77)))
         facilities = ["D"+str(i) for i in range(1,6)]
         flow = pd.DataFrame(np.array([[0,100,50,0,0],
                         [75,0,50,50,25],
                         [100,25,0,0,0],
                         [0,0,50,0,50],
                         [50,0,50,25,0]]),index=facilities,columns=facilities)
         distance = pd.DataFrame(np.array([[0,1,2,3,1,2,3,4,2,3,4,5],
                               [1,0,1,2,2,1,2,3,3,2,3,4],
                               [2,1,0,1,3,2,1,2,4,3,2,3],
                               [3,2,1,0,4,3,2,1,5,4,3,2],
                               [1,2,3,4,0,1,2,3,1,2,3,4],
                               [2,1,2,3,1,0,1,2,2,1,2,3],
                               [3,2,1,2,2,1,0,1,3,2,1,2],
                               [4,3,2,1,3,2,1,0,4,3,2,1],
                               [2,3,4,5,1,2,3,4,0,1,2,3],
                               [3,2,3,4,2,1,2,3,1,0,1,2],
                               [4,3,2,3,3,2,1,2,2,1,0,1],
                               [5,4,3,2,4,3,2,1,3,2,1,0]]),index=locations,columns=locations)
         assignments = [(i,j,r,s) for i in facilities for j in facilities
                        for r in locations for s in locations if i != j and r != s]
```

```
In [12]: m = gp.Model("q2")
         m.ModelSense = GRB.MINIMIZE

         x=m.addVars(facilities, locations, name="x", vtype=GRB.BINARY)
         c1 = m.addConstrs(gp.quicksum(x[i,r] for i in facilities) <= 1 for r in locations)
         c2 = m.addConstrs(gp.quicksum(x[i,r] for r in locations) == 1 for i in facilities)
         c3 = m.addConstr(gp.quicksum(distance.at[i,j] * x["D1",i] * x["D4",j] for i in locations for j in locations) >= 2)
         c4 = m.addConstrs(x[i,r] == 0 for i in ["D1", "D2", "D5"] for r in ["A", "D", "L", "I"])
         c5 = m.addConstrs(x["D3",r] == 0 for r in locations if (r in ["A", "D", "L", "I"]) == False)
         m.setObjective(gp.quicksum((flow.at[i,j]*distance.at[r,s] + flow.at[j,i]*distance.at[s,r])*x[i,r]*x[j,s]
                                    for i in facilities for j in facilities for r in locations for s in locations
                                    if i != j and r != s))
```

```
In [13]: m.optimize()
```

Figure 4: Code of Question 2

```
In [13]: m.optimize()
         0     0 1516.66667    0   19 1900.00000 1516.66667  20.2%     -    0s
         0     0 1525.00000    0   17 1900.00000 1525.00000  19.7%     -    0s
         0     0 1700.00000    0   37 1900.00000 1700.00000  10.5%     -    0s
         0     0 1700.00000    0    8 1900.00000 1700.00000  10.5%     -    0s
    H    0     0                    1850.0000000 1700.00000  8.11%     -    0s

    Cutting planes:
      Implied bound: 1
      MIR: 2
      Zero half: 6
      RLT: 16
      BQP: 1

    Explored 1 nodes (1088 simplex iterations) in 0.07 seconds (0.06 work units)
    Thread count was 8 (of 8 available processors)

    Solution count 9: 1850 1900 2100 ... 3650

    Optimal solution found (tolerance 1.00e-04)
    Best objective 1.850000000000e+03, best bound 1.850000000000e+03, gap 0.0000%
```

```
In [14]: for key in x.keys():
             if x[key].X > 0 and key[0] in ["D1", "D2", "D3", "D4", "D5"]:
                 print(key)

         ('D1', 'E')
         ('D2', 'F')
         ('D3', 'A')
         ('D4', 'C')
         ('D5', 'B')
```

Figure 5: Output of Question 2

The objective function is 1850 and the optimal grid locations are as follows:

- $D_1 \to E$
- $D_2 \to F$
- $D_3 \to A$
- $D_4 \to C$
- $D_5 \to B$

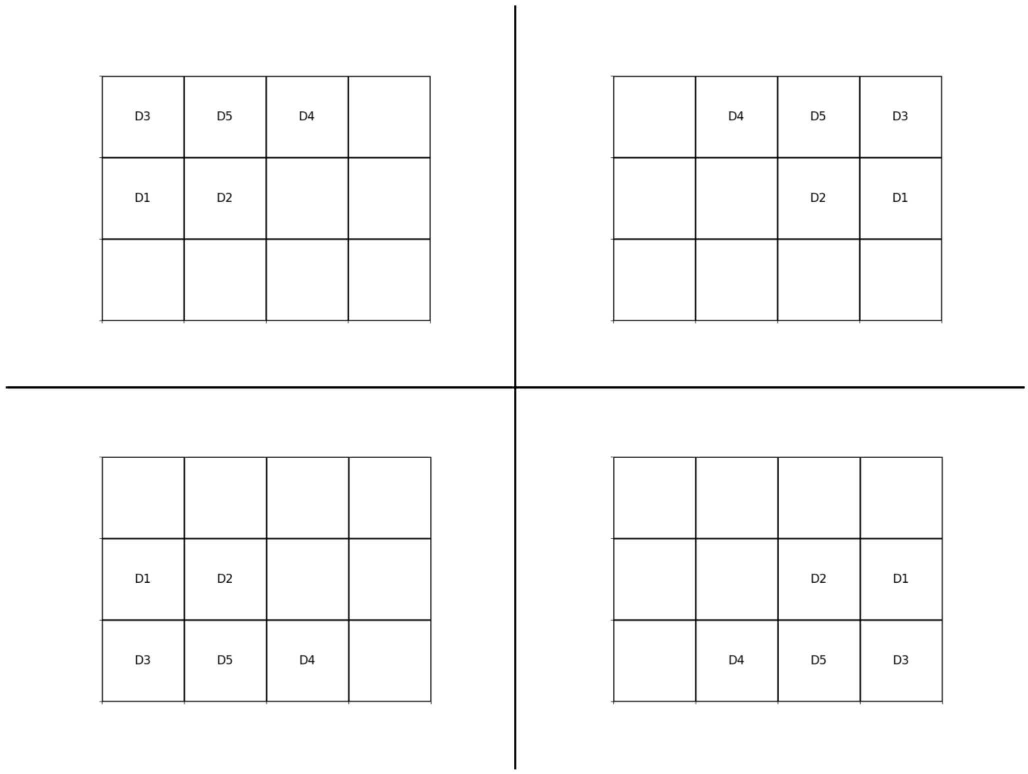

Figure 6: Layout of locations with all mirrored versions