Operations Research II: Algorithms

Gurobi and Python for Integer Programming

Serena Chen

Department of Information Management
National Taiwan University

Road map

- **Introduction**.
- Facility location.

Introduction

- We know how to solve a linear program with `gurobipy`. What if we want to solve an **integer program**?
- Gurobi Optimizer can solve both linear and integer programs.
  - There are three attributes for variables in Gurobi Optimizer: `GRB.BINARY`, `GRB.INTEGER`, and `GRB.CONTINUOUS`.
  - To formulate an LP, set the variable type to `GRB.CONTINUOUS`.
  - To formulate an IP, set the variable type to `GRB.INTEGER`; if that variable can only be 0 or 1, set its type to `GRB.BINARY`.

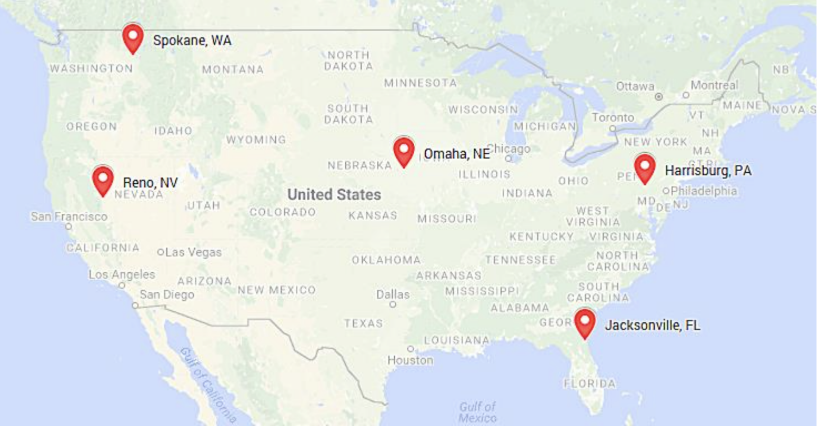
Road map

- Introduction.
- **Facility location**.

Facility location

- Consider the facility location problem we have introduced in *Operations Research I: Modeling and Applications*.
- A company wants to build a few distribution centers to serve five markets.
- There are five candidate centers. Each market has its own demand and each distribution ceter (once open) has its operating cost and capacity. There is also shipping cost from each facility to each market.
- Our goal is to choose one or multiple locations to build distribution centers, satisfy all demands, and minimize the total cost.

Facility location



Facility location

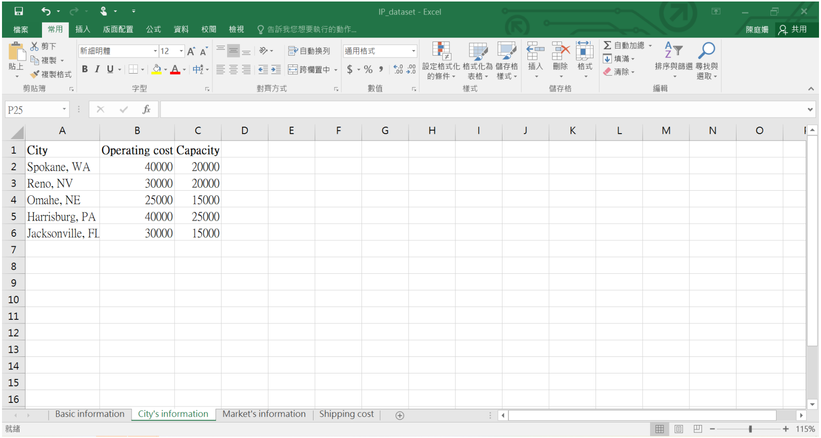
- The formulation is

$$
\begin{aligned}
\min \quad & \sum_{j=1}^{5} f_j x_j + \sum_{i=1}^{5} \sum_{j=1}^{5} c_{ij} y_{ij} & \\
\text{s.t.} \quad & \sum_{i=1}^{5} y_{ij} \leq K_j x_j & \forall j = 1, ..., 5 \\
& \sum_{j=1}^{5} y_{ij} \geq D_i & \forall i = 1, ..., 5 \\
& x_j \in \{0, 1\} & \forall j = 1, ..., 5 \\
& y_{ij} \geq 0 & \forall i = 1, ..., 5, j = 1, ..., 5.
\end{aligned}
$$

Decoupling the data from a model

- Again, it is better to do data-model decoupling to make our Python program more flexible and extendible.
- Moreover, we seldom store the data in the codes. Instead, we define our data in a **data file**.
- In this example, the data file should be read **before** the model part.

The data file



City	Operating cost	Capacity
Spokane, WA	40000	20000
Reno, NV	30000	20000
Omahe, NE	25000	15000
Harrisburg, PA	40000	25000
Jacksonville, FL	30000	15000

The data part

```
import pandas as pd

# Read excel sheets and turn them into lists
basic_info = pd.read_excel('IP_dataset.xlsx',
  'Basic information')
cities = range(len(basic_info['City']))
markets = range(len(basic_info['Market']))

city_info = pd.read_excel('IP_dataset.xlsx',
  'City\'s information')
operating_costs = city_info['Operating cost']
capacities = city_info['Capacity']
```

- The Python library pandas is widely used to handle data.
- We use the function `read_excel` to import the excel sheets and transform the dataframe into several lists.

The data part

```
market_info = pd.read_excel('IP_dataset.xlsx',
  'Market\'s information')
demands = market_info['Demand']

shipping_info = pd.read_excel('IP_dataset.xlsx',
  'Shipping cost', index_col = 0)
shipping_costs = []
for i in shipping_info.index:
  shipping_costs.append(list(shipping_info.loc[i]))
```

- Now the data is loaded and stored into some lists.

The model part

```
eg2 = Model("eg2")    # build a new model

# add variables as a list
x = []
for j in cities:
  x.append(eg2.addVar(lb = 0, vtype = GRB.BINARY,
    name = "x" + str(j+1)))

y = []
for i in markets:
  y.append([])
  for j in cities:
    y[i].append(eg2.addVar(lb = 0, vtype = GRB.CONTINUOUS,
      name = "y" + str(i+1) + str(j+1)))
```

- The length of list must be consistent. Use `append` to add a new item or list into a list.

The model part

```
# setting the objective function
eg2.setObjective(
  quicksum(operating_costs[j] * x[j] for j in cities) +
  quicksum(quicksum(shipping_costs[i][j] * y[i][j]
    for j in cities) for i in markets), GRB.MINIMIZE)

# add constraints and name them
eg2.addConstrs((quicksum(y[i][j] for i in markets) <=
  capacities[j] * x[j] for j in cities),
  "product_capacity")

eg2.addConstrs((quicksum(y[i][j] for j in cities) >=
  demands[i] for i in markets),
  "demand_fulfillment")
```

- In `setObjective`, use `GRB.MINIMIZE` for minimization problems.
- Use `quicksum` to sum up a group of variables.
- Use `addConstrs` to add multiple constraints in one command.

Solving the facility location problem

- Solve the problem by executing the following statement.

```
eg2.optimize()
```

```
Gurobi Optimizer version 9.0.2 build v9.0.2rc0 (win64)
Optimize a model with 10 rows, 30 columns and 55 nonzeros
Model fingerprint: 0x263201cb
Variable types: 25 continuous, 5 integer (5 binary)
Coefficient statistics:
  Matrix range     [1e+00, 3e+04]
  Objective range  [2e+00, 4e+04]
  Bounds range     [1e+00, 1e+00]
  RHS range        [8e+03, 2e+04]
Presolve time: 0.00s
Presolved: 10 rows, 30 columns, 55 nonzeros
Variable types: 25 continuous, 5 integer (5 binary)
```

Solving the facility location problem

```
Root relaxation: objective 2.528000e+05, 11 iterations, 0.00 seconds

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

     0     0 252800.000    0    5          - 252800.000      -     -    0s
H    0     0                    313600.00000 252800.000  19.4%     -    0s
H    0     0                    280400.00000 252800.000  9.84%     -    0s
     0     0 266813.478    0    3 280400.000 266813.478  4.85%     -    0s
H    0     0                    272800.00000 266813.478  2.19%     -    0s
H    0     0                    268950.00000 266813.478  0.79%     -    0s

Cutting planes:
  Gomory: 5
  Implied bound: 5
  MIR: 2
  Flow cover: 1

Explored 1 nodes (16 simplex iterations) in 0.02 seconds
Thread count was 8 (of 8 available processors)

Solution count 4: 268950 272800 280400 313600

Optimal solution found (tolerance 1.00e-04)
Best objective 2.689500000000e+05, best bound 2.689500000000e+05, gap 0.0000%
```

Solving the facility location problem

- To print out the solution (in a nice way), execute the following statements:

```
print("Result:")
for j in cities:
  print(x[j].varName, '=', x[j].x)

# head of the result table
print("\tMarket1\tMarket2\tMarket3\tMarket4\tMarket5")
for j in cities:
 # mark which product is printed now
   print("City" + str(j+1), "\t", end="")
     for i in markets:
       # print values of each kind of product
       if len(str(y[i][j].x)) < 7:
         print(y[i][j].x, "\t", end="")
       else:
         print(y[i][j].x, "", end="")
print("\nz* =", eg2.objVal)    # print objective value
```

Solving the facility location problem

```
Result:
x1 = 0.0
x2 = 1.0
x3 = 0.0
x4 = 1.0
x5 = 1.0
        Market1 Market2 Market3 Market4 Market5
City1   0.0     0.0     0.0     0.0     0.0
City2   8000.0  12000.0 0.0     0.0     0.0
City3   0.0     0.0     0.0     0.0     0.0
City4   0.0     0.0     8000.0  0.0     17000.0
City5   0.0     0.0     1000.0  14000.0 0.0
z* = 268950.0
```

Some modifications

- Suppose that the company wants to build at least four distribution centers for some reason.
- We may add a new constraint

```
B = 4
eg2.addConstr(quicksum(x[j] for j in City) >= B,
  "locations_limit")
```

- Modify the code and see how the optimal solution changes.

Some remarks

- Even with the help of a computer, solving a large-scale integer program still takes lots of time!
- Carefully determine whether it is necessary to set a variable as integer. It is a trade-off between precision and efficiency.