Week 10: Using the Gurobi Solver

Session 19: Automating Patterns using For Loops and List Comprehension

Example: Numerical Solution for Assortment Planning

Decision variables: Let x_i denote whether to carry book i. (Binary)
Objective:

$$\text{Minimize:} \quad x_1 + x_2 + \cdots + x_{10}$$

Constraints:

$$\begin{aligned}
\text{(Literary)} \quad & x_1 + x_4 + x_5 + x_9 \geq 2 \\
\text{(Sci-Fi)} \quad & x_2 + x_7 + x_9 \geq 2 \\
\text{(Romance)} \quad & x_3 + x_4 + x_6 + x_{10} \geq 2 \\
\text{(Thriller)} \quad & x_2 + x_3 + x_8 \geq 2
\end{aligned}$$

Version 1: Hard-coding in everything

```
[22]: from gurobipy import Model, GRB
      mod=Model()
      x1=mod.addVar(vtype=GRB.BINARY)
      x2=mod.addVar(vtype=GRB.BINARY)
      x3=mod.addVar(vtype=GRB.BINARY)
      x4=mod.addVar(vtype=GRB.BINARY)
      x5=mod.addVar(vtype=GRB.BINARY)
      x6=mod.addVar(vtype=GRB.BINARY)
      x7=mod.addVar(vtype=GRB.BINARY)
      x8=mod.addVar(vtype=GRB.BINARY)
      x9=mod.addVar(vtype=GRB.BINARY)
      x10=mod.addVar(vtype=GRB.BINARY)
      mod.setObjective(x1+x2+x3+x4+x5+x6+x7+x8+x9+x10)
      mod.addConstr(x1+x4+x5+x9>=2)
      mod.addConstr(x2+x7+x9>=2)
      mod.addConstr(x3+x4+x6+x10>=2)
      mod.addConstr(x2+x3+x8>=2)
      mod.setParam('OutputFlag',False)
      mod.optimize()
      print('Optimal objective:',mod.objVal)
      print(f'Optimal solution: x1={x1.x} x2={x2.x} x3={x3.x} x4={x4.x} x5={x5.x} x6={x6.x} x
```

args: vtype
default: GRB.CONTINUOUS

lb
default: 0
can be set as
-GRB.INFINITY

Sense:
default: GRB.Minimize

Solves

```
Optimal objective: 4.0
Optimal solution: x1=0.0 x2=1.0 x3=1.0 x4=1.0 x5=0.0 x6=0.0 x7=0.0 x8=0.0 x9=1.0 x10=0.0
```

```
[2]: type(mod)
```

```
gurobipy.Model
```

```
[3]: type(x1)
```

```
gurobipy.Var
```

```
[4]: type(x1+x4+x5+x9)
```

```
gurobipy.LinExpr
```

```
[5]: type(x1+x4+x5+x9>=2)
```

```
gurobipy.TempConstr
```

Version 2: Using addVars to create multiple variables at once

```
[6]: from gurobipy import Model, GRB
     mod=Model()
     books=range(1,11)
     x=mod.addVars(books, vtype=GRB.BINARY)
     mod.setObjective(x[1]+x[2]+x[3]+x[4]+x[5]+x[6]+x[7]+x[8]+x[9]+x[10])
     mod.addConstr(x[1]+x[4]+x[5]+x[9]>=2)
     mod.addConstr(x[2]+x[7]+x[9]>=2)
     mod.addConstr(x[3]+x[4]+x[6]+x[10]>=2)
     mod.addConstr(x[2]+x[3]+x[8]>=2)
     mod.setParam('OutputFlag',False)
     mod.optimize()
     print('Optimal objective:',mod.objVal)
     print('Optimal solution: carry books ',end='')
     for b in books:
         if x[b].x==1:
             print(b, end=' ')
```

creates a dictionary of decision variables.

```
Optimal objective: 4.0
Optimal solution: carry books 2 3 4 9
```

Version 3: Using list comprehension to generate sums

```
[7]: from gurobipy import Model, GRB
     mod=Model()
     books=range(1,11)
     literary=[1,4,5,9]
     scifi=[2,7,9]
     romance=[3,4,6,10]
     thriller=[2,3,8]
     x=mod.addVars(books, vtype=GRB.BINARY)
     mod.setObjective(sum(x[b] for b in books))
     mod.addConstr(sum(x[b] for b in literary)>=2)
     mod.addConstr(sum(x[b] for b in scifi)>=2)
     mod.addConstr(sum(x[b] for b in romance)>=2)
     mod.addConstr(sum(x[b] for b in thriller)>=2)
     mod.setParam('OutputFlag',False)
     mod.optimize()
     print('Optimal objective:',mod.objVal)
     print('Optimal solution: carry books ',end='')
     for b in books:
         if x[b].x==1:
             print(b, end=' ')
```

list comprehension to generate sums.

```
Optimal objective: 4.0
Optimal solution: carry books 2 3 4 9
```

Version 4: Using for loops to generate repetitive constraints

[8]:
```python
from gurobipy import Model, GRB
mod=Model()
books=range(1,11)
booksInGenre={'Literary':[1,4,5,9],\
              'Sci-Fi':[2,7,9],\
              'Romance':[3,4,6,10],\
              'Thriller':[2,3,8]}
requirement={'Literary':2,'Sci-Fi':2,'Romance':2,'Thriller':2}
x=mod.addVars(books, vtype=GRB.BINARY)
mod.setObjective(sum(x[b] for b in books))
for genre in booksInGenre:
    mod.addConstr(sum(x[b] for b in booksInGenre[genre])>=requirement[genre])
mod.setParam('OutputFlag',False)
mod.optimize()
print('Optimal objective:',mod.objVal)
print('Optimal solution: carry books ',end='')
for b in books:
    if x[b].x==1:
        print(b, end=' ')
```

```
Optimal objective: 4.0
Optimal solution: carry books 2 3 4 9
```

Useful tool for debugging: Let Gurobi Display the Concrete Formulation

[9]:
```python
x=mod.addVars(books, vtype=GRB.BINARY, name='x')
mod.update()
sum(x[b] for b in books)
```

↳ required only for debugging.

```
<gurobi.LinExpr: x[1] + x[2] + x[3] + x[4] + x[5] + x[6] + x[7] + x[8] + x[9] + x[10]>
```

[10]:
```python
genre='Literary'
sum(x[b] for b in booksInGenre[genre])>=requirement[genre]
```

```
<gurobi.TempConstr: <gurobi.LinExpr: x[1] + x[4] + x[5] + x[9]> >= 2>
```

[11]:
```python
from gurobipy import Model, GRB
mod=Model()
books=range(1,11)
booksInGenre={'Literary':[1,4,5,9],'Sci-Fi':[2,7,9],'Romance':[3,4,6,10],'Thriller':[2,
requirement={'Literary':2,'Sci-Fi':2,'Romance':2,'Thriller':2}
x=mod.addVars(books, vtype=GRB.BINARY, name='x')
mod.setObjective(sum(x[b] for b in books))
for genre in booksInGenre:
    mod.addConstr(sum(x[b] for b in booksInGenre[genre])>=requirement[genre],\
                  name=genre)
mod.write('10-books.lp')
%cat 10-books.lp
# %cat works only for Mac and Linux
# For Windows, replace %cat with !type
```

→ for labelling in debugging.

```
\ LP format - for model browsing. Use MPS format to capture full model detail.
Minimize
  x[1] + x[2] + x[3] + x[4] + x[5] + x[6] + x[7] + x[8] + x[9] + x[10]
Subject To
 Literary: x[1] + x[4] + x[5] + x[9] >= 2
 Sci-Fi: x[2] + x[7] + x[9] >= 2
 Romance: x[3] + x[4] + x[6] + x[10] >= 2
 Thriller: x[2] + x[3] + x[8] >= 2
Bounds
Binaries
 x[1] x[2] x[3] x[4] x[5] x[6] x[7] x[8] x[9] x[10]
End
```

Exercise 10.1 Numerical Solution for Project Sub-Contracting

Download the Jupyter notebook attached to the Blackboard link for this exercise and submit it there after completing it. The notebook asks you to follow the above example to incrementally produce a version of the Gurobi code that does not hard-code in numbers but obtain them from appropriate data structures.
Decision variable:

- Let x_i denote whether to schedule job i for own company. (Binary)
- Let y_i denote whether to subcontract job i. (Binary)

Objective:

$$\text{Maximize:} \qquad 30x_1 + 10x_2 + 26x_3 + 18x_4 + 20x_5 + 6y_1 + 2y_2 + 8y_3 + 9y_4 + 4y_5$$

Constraints:

$$\begin{aligned}
\text{(Labor)} \quad & 1300x_1 + 950x_2 + 1000x_3 + 1400x_4 + 1600x_5 \leq 4800 \\
\text{(Doing every project)} \quad & x_1 + y_1 = 1 \\
& x_2 + y_2 = 1 \\
& x_3 + y_3 = 1 \\
& x_4 + y_4 = 1 \\
& x_5 + y_5 = 1
\end{aligned}$$

Version 1: Hard-coding in everything

For comparison purposes, write a version of the code that hard-codes in everything, similar to version 1 of the previous example. Remember to set the sense in the objective to GRB.MAXIMIZE.

```
...
Optimal objective: 88.0
Optimal solution: x1=1.0, x2=1.0, x3=1.0, x4=1.0, x5=0.0
```

Version 2: Using addVars to create multiple variables at once

Using addVars, generate all of the x's using one command, and all of the y's using one command. Also, make the optimal solution easier to read, as in the output below.

```
...
Optimal objective: 88.0
Optimal solution: do projects 1 2 3 4 yourself
```

Version 3 and 4: Using list comprehension and for loops

Instead of hard-coding in the numbers, obtain them from the following data structures. Moreover, use list comprehension to generate the large sums, and for loops to generate the repetitive constraints. Build up the formulation part by part and double in the end check by making Gurobi display the entire concrete formulation.

```
[14]: import pandas as pd
      projects=range(1,6)
      ownLabor=4800
      profit=pd.DataFrame([[30,10,26,18,20],[6,2,8,9,4]], \
                          index=['Yourself','Subcontract'], columns=projects)
      profit
```

	1	2	3	4	5
Yourself	30	10	26	18	20
Subcontract	6	2	8	9	4

```
[15]: laborRequired=pd.Series([1300,950,1000,1400,1600],index=projects)
      laborRequired
```

```
1    1300
2     950
3    1000
4    1400
5    1600
dtype: int64
```

```
[16]: # Objective function
```

```
<gurobi.LinExpr: 30.0 x[1] + 6.0 y[1] + 10.0 x[2] + 2.0 y[2] + 26.0 x[3] + 8.0 y[3] + 18.0 x[
```

```
[17]: # Labor constraint
```

```
<gurobi.TempConstr: <gurobi.LinExpr: 1300.0 x[1] + 950.0 x[2] + 1000.0 x[3] + 1400.0 x[4] + 1
```

```
[18]: # Entire formulation
```

```
\ LP format - for model browsing. Use MPS format to capture full model detail.
Maximize
  30 x[1] + 10 x[2] + 26 x[3] + 18 x[4] + 20 x[5] + 6 y[1] + 2 y[2] + 8 y[3]
   + 9 y[4] + 4 y[5]
Subject To
 Labor: 1300 x[1] + 950 x[2] + 1000 x[3] + 1400 x[4] + 1600 x[5] <= 4800
 Project_1: x[1] + y[1] = 1
 Project_2: x[2] + y[2] = 1
 Project_3: x[3] + y[3] = 1
 Project_4: x[4] + y[4] = 1
 Project_5: x[5] + y[5] = 1
Bounds
Binaries
 x[1] x[2] x[3] x[4] x[5] y[1] y[2] y[3] y[4] y[5]
End
```

Exercise 10.2 Numerical Solution for Food Production

Download the Jupyter notebook attached to the Blackboard link for this exercise and submit it there after completing it. The notebook asks you to solve the following concrete formulation, while loading input data from the given data structures.

Decision Variables:

- $X_1, X_2, \cdots, X_6$: amount of oil to buy in each month. (continuous)
- $Y_1, Y_2, \cdots, Y_6$: amount of oil stored at the end of each month. (continuous)

Objective:

$$\text{Min. } 150X_1 + 160X_2 + 180X_3 + 170X_4 + 180X_5 + 160X_6$$

Constraints:

$$\begin{aligned}
Y_1 &= X_1 - 2000 \\
Y_2 &= X_2 + Y_1 - 2000 \\
Y_3 &= X_3 + Y_2 - 2000 \\
Y_4 &= X_4 + Y_3 - 2000 \\
Y_5 &= X_5 + Y_4 - 2000 \\
Y_6 &= X_6 + Y_5 - 2000 \\
Y_t &\leq 1000 && \text{for each month } t \in \{1, 2, \cdots, 6\}. \\
X_t, Y_t &\geq 0 && \text{for each month } t.
\end{aligned}$$

[20]:
```
# Input data
import pandas as pd
months=range(1,7)
price=pd.Series([150,160,180,170,180,160],index=months)
```

```
Minimum purchase cost: 1960000.0
Month          Buy          Store
1         3000.0          1000.0
2         2000.0          1000.0
3         1000.0          0.0
4         3000.0          1000.0
5         1000.0          0.0
6         2000.0          0.0
```