Contents

Week 10: Using the Gurobi Solver

By the end of this week, you will be able to use Gurobi to solve all of the linear optimization formulations we created in the previous weeks, while using for-loops and list-comprehension to automate repetitive patterns. Being able to do this allows you to numerically solve much larger optimization models.

Session 19: Automating Patterns using For-Loops and List-Comprehension

Example: Numerical Solution for Assortment Planning

Decision variables:

- x_i: 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

```
[1]: 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} x7={x7.x} x8={x8.
↪x} x9={x9.x} x10={x10.x}')
```

```
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=' ')
```

```
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=' ')
```

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

Version 4: Using for loops to generate repetitive constraints

```
[8]: 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
```

Let Gurobi Display the Concrete Formulation

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

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

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

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

```
[11]: 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, 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)
```

```python
mod.write('10-books.lp')
%cat 10-books.lp
# %cat works only for Mac and Linux
# For Windows, replace %cat with !type
```

```
\ 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

Following the assortment planning example, 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[4] + 9.0 y[4] + 20.0 x[5] + 4.0 y[5]>
```

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

```
<gurobi.TempConstr: 1300.0 x[1] + 950.0 x[2] + 1000.0 x[3] + 1400.0 x[4] + 1600.0 x[5]␣
↪<= 4800>
```

```
[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
```

Final code

Use the final version of your formulation to produce the following output.

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

Exercise 10.2: Numerical Solution for Warehouse Planning

The concrete formulation of Exercise 8.5 is reproduced below:

Decision Variables: Let $X_1, \cdots, X_7$ denote whether to use each FC. (Binary)

Objective and constraints:

$$\begin{aligned} \text{Minimize} \quad & X_1 + X_2 + \cdots + X_7 \\ \text{s.t.} \quad & X_2 + X_5 + X_6 + X_7 \geq 1 \\ & X_3 + X_4 \geq 1 \\ & X_3 \geq 1 \\ & X_1 + X_2 + X_4 + X_6 \geq 1 \\ & X_5 + X_7 \geq 1 \\ & X_4 \leq X_1 \\ & X_2 + X_3 \leq 1 \end{aligned}$$

a) Implement the above using Gurobi, while using for loops and list comprehensions as much as possible to automate recurring patterns.

After you are done, use `mod.write`, and `%cat` in Mac or `!type` in Windows to output what the linear optimization formulation looks like according to Gurobi. You can use this to verify that you have indeed implemented the above.

```
\ 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]
Subject To
 R0: X[2] + X[5] + X[6] + X[7] >= 1
 R1: X[3] + X[4] >= 1
 R2: X[3] >= 1
 R3: X[1] + X[2] + X[4] + X[6] >= 1
 R4: X[5] + X[7] >= 1
 R5: - X[1] + X[4] <= 0
 R6: X[2] + X[3] <= 1
Bounds
Binaries
 X[1] X[2] X[3] X[4] X[5] X[6] X[7]
End
```

b) Solve the MIP and print the minimum number of FCs needed, as well as where to stock the items. The output format should match the sample output below.

```
Minimum # of FCs needed: 3
Stock item in the following:
        FC1
        FC3
        FC7
```

Session 20: Automating Patterns with Multiple Indices

Example: Numerical Solution for Supply Chain Planning

Recap of Exercise 8.2: The following table provides the shipping cost for a certain item, from three of Amazon's fulfillment centers (FC) to four regions (A, B, C and D).

Region FC	1	2	3
A. Kings County, NY	20	8	25
B. Los Angeles County, CA	18	23	8
C. King County, WA	21	24	8
D. Harris County, TX	8	8	19

The following table summarizes the weekly demand for the item from each region.

Region A	Region B	Region C	Region D
30	50	10	20

Suppose that each FC is able to ship up to 40 units each week in total. Formulate a linear program to determine the minimum transportation cost needed to satisfy all demand while respecting FC capacities, as well as the optimal shipment plan.

Concrete Formulation

Decision variables:

- x_{ij}: the amount to transport from FC i to region j. (Integer)

Objective:

$$\begin{aligned} \text{Minimize: } & 20x_{1A} + 18x_{1B} + 21x_{1C} + 8x_{1D} + \\ & 8x_{2A} + 23x_{2B} + 24x_{2C} + 8x_{2D} + \\ & 25x_{3A} + 8x_{3B} + 8x_{3C} + 19x_{3D} \end{aligned}$$

Constraints:

$$\begin{aligned} \text{(Capacity 1)} \quad & x_{1A} + x_{1B} + x_{1C} + x_{1D} \leq 40 \\ \text{(Capacity 2)} \quad & x_{2A} + x_{2B} + x_{2C} + x_{2D} \leq 40 \\ \text{(Capacity 3)} \quad & x_{3A} + x_{3B} + x_{3C} + x_{3D} \leq 40 \\ \text{(Demand A)} \quad & x_{1A} + x_{2A} + x_{3A} \geq 30 \\ \text{(Demand B)} \quad & x_{1B} + x_{2B} + x_{3B} \geq 50 \\ \text{(Demand C)} \quad & x_{1C} + x_{2C} + x_{3C} \geq 10 \\ \text{(Demand D)} \quad & x_{1D} + x_{2D} + x_{3D} \geq 20 \\ \text{(Non-negativity)} \quad & x_{ij} \geq 0 \qquad \text{for all } i \text{ and } j. \end{aligned}$$

The following code implements the above, while avoiding hard-coding in the numbers by obtaining them from appropriate data structures.

Input Data

```
[22]: import pandas as pd
      cost=pd.DataFrame([[20,18,21,8],[8,23,24,8],[25,8,8,19]],\
                         index=[1,2,3],columns=['A','B','C','D'])
      cost
```

```
    A   B   C   D
1  20  18  21   8
2   8  23  24   8
3  25   8   8  19
```

```
[23]: demand=pd.Series([30,50,10,20],index=['A','B','C','D'])
      demand
```

```
A    30
B    50
C    10
D    20
dtype: int64
```

```
[24]: capacity=pd.Series([40]*3, index=[1,2,3])
      capacity
```

```
1    40
2    40
3    40
dtype: int64
```

Implementing the Formulation Incrementally

```
[25]: # Creating variables
      from gurobipy import Model,GRB
      mod=Model()
      FCs=cost.index
      regions=cost.columns
      x=mod.addVars(FCs,regions,name='x')
      mod.update()
      x
```

```
{(1, 'A'): <gurobi.Var x[1,A]>,
 (1, 'B'): <gurobi.Var x[1,B]>,
 (1, 'C'): <gurobi.Var x[1,C]>,
 (1, 'D'): <gurobi.Var x[1,D]>,
 (2, 'A'): <gurobi.Var x[2,A]>,
 (2, 'B'): <gurobi.Var x[2,B]>,
 (2, 'C'): <gurobi.Var x[2,C]>,
 (2, 'D'): <gurobi.Var x[2,D]>,
 (3, 'A'): <gurobi.Var x[3,A]>,
 (3, 'B'): <gurobi.Var x[3,B]>,
 (3, 'C'): <gurobi.Var x[3,C]>,
 (3, 'D'): <gurobi.Var x[3,D]>}
```

```
[26]: # Objective
      sum(cost.loc[f,r]*x[f,r] for f in FCs for r in regions)
```

```
<gurobi.LinExpr: 20.0 x[1,A] + 18.0 x[1,B] + 21.0 x[1,C] + 8.0 x[1,D] + 8.0 x[2,A] + 23.
 ↪0 x[2,B] + 24.0 x[2,C] + 8.0 x[2,D] + 25.0 x[3,A] + 8.0 x[3,B] + 8.0 x[3,C] + 19.0␣
 ↪x[3,D]>
```

```
[27]: # Capacity 1 Constraint
      sum(x[1,r] for r in regions)<=capacity[1]
```

```
<gurobi.TempConstr: x[1,A] + x[1,B] + x[1,C] + x[1,D] <= 40>
```

```
[28]: # Demand 1 Constraint
      sum(x[f,'A'] for f in FCs)>=demand['A']
```

```
<gurobi.TempConstr: x[1,A] + x[2,A] + x[3,A] >= 30>
```

```
[29]: # Full model
      mod=Model()
      x=mod.addVars(FCs,regions,name='x')
      mod.setObjective(sum(cost.loc[f,r]*x[f,r] for f in FCs for r in regions))
      for f in FCs:
          mod.addConstr(sum(x[f,r] for r in regions)<=capacity[f],name=f'Capacity_{f}')
      for r in regions:
          mod.addConstr(sum(x[f,r] for f in FCs)>=demand[r],name=f'Demand_{r}')
      mod.write('10-supplyChain.lp')
      %cat 10-supplyChain.lp
```

```
\ LP format - for model browsing. Use MPS format to capture full model detail.
Minimize
  20 x[1,A] + 18 x[1,B] + 21 x[1,C] + 8 x[1,D] + 8 x[2,A] + 23 x[2,B]
   + 24 x[2,C] + 8 x[2,D] + 25 x[3,A] + 8 x[3,B] + 8 x[3,C] + 19 x[3,D]
Subject To
 Capacity_1: x[1,A] + x[1,B] + x[1,C] + x[1,D] <= 40
 Capacity_2: x[2,A] + x[2,B] + x[2,C] + x[2,D] <= 40
 Capacity_3: x[3,A] + x[3,B] + x[3,C] + x[3,D] <= 40
 Demand_A: x[1,A] + x[2,A] + x[3,A] >= 30
 Demand_B: x[1,B] + x[2,B] + x[3,B] >= 50
 Demand_C: x[1,C] + x[2,C] + x[3,C] >= 10
 Demand_D: x[1,D] + x[2,D] + x[3,D] >= 20
Bounds
End
```

```
[30]: # Numerical Solution
      mod.setParam('OutputFlag',False)
      mod.optimize()
      print('Minimum cost:',mod.objval)
```

```
Minimum cost: 1080.0
```

```
[31]: # Creating the output table
      shipment=pd.DataFrame(index=FCs, columns=regions)
      shipment
```

	A	B	C	D
1	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN

```
[32]: # Filling in the output table
      for f in FCs:
          for r in regions:
              shipment.loc[f,r]=x[f,r].x
      shipment
```

```
      A     B     C     D
1   0.0  20.0   0.0  20.0
2  30.0   0.0   0.0   0.0
3   0.0  30.0  10.0   0.0
```

```
[33]: # Final code
      from gurobipy import Model,GRB
      mod=Model()
      FCs=cost.index
      regions=cost.columns
      x=mod.addVars(FCs,regions)
      mod.setObjective(sum(cost.loc[f,r]*x[f,r] for f in FCs for r in regions))
      for f in FCs:
          mod.addConstr(sum(x[f,r] for r in regions)<=capacity[f])
      for r in regions:
          mod.addConstr(sum(x[f,r] for f in FCs)>=demand[r])
      mod.setParam('OutputFlag',False)
      mod.optimize()
      print('Minimum cost:',mod.objval)
      shipment=pd.DataFrame(index=FCs, columns=regions)
      for f in FCs:
          for r in regions:
              shipment.loc[f,r]=x[f,r].x
      shipment
```

```
Minimum cost: 1080.0
```

```
      A     B     C     D
1   0.0  20.0   0.0  20.0
2  30.0   0.0   0.0   0.0
3   0.0  30.0  10.0   0.0
```

Exercise 10.3: Numerical Solution for Assignment of Consultants to Projects

Incrementally create Gurobi code to solve Exercise 8.3, following the example given in the lecture.

Recap of Exercise 8.3: There are two projects and four consultants: Alice, Bob, Charles, and Daphne. Each consultant can be assigned to at most one project, and each project requires at least two consultants. As a manager, you evaluated the relative fitness of the four consultants for each project on a scale of 1 to 5, with 5 being the best fit and 1 being the worst.

	Project 1	Project 2
Alice	5	2
Bob	3	2
Charles	4	5
Daphne	3	1

Furthermore, Alice, Bob and Daphne are senior consultants and each project requires at least one senior on the team.

Formulate a linear optimization problem to maximize the total fitness of the consultants to their assigned project, subject to all the business constraints.

Concrete Formulation:

Decision variables:

- x_{ij}: whether to assign consultant i to project j. (Binary)

Objective:

$$\text{Maximize: } 5x_{A1} + 2x_{A2} + 3x_{B1} + 2x_{B2} + 4x_{C1} + 5x_{C2} + 3x_{D1} + x_{D2}$$

Constraints:

$$\begin{aligned}
\text{(Alice)} \quad && x_{A1} + x_{A2} \leq 1 \\
\text{(Bob)} \quad && x_{B1} + x_{B2} \leq 1 \\
\text{(Charles)} \quad && x_{C1} + x_{C2} \leq 1 \\
\text{(Daphne)} \quad && x_{D1} + x_{D2} \leq 1 \\
\text{(Project 1 Total)} \quad && x_{A1} + x_{B1} + x_{C1} + x_{D1} \geq 2 \\
\text{(Project 2 Total)} \quad && x_{A2} + x_{B2} + x_{C2} + x_{D2} \geq 2 \\
\text{(Project 1 Senior)} \quad && x_{A1} + x_{B1} + x_{D1} \geq 1 \\
\text{(Project 2 Senior)} \quad && x_{A2} + x_{B2} + x_{D2} \geq 1
\end{aligned}$$

Implement the above in Gurobi while obtaining all numbers from the below data structures. See the desired intermediate outputs for every step below. You should write your code in such a way such that if the input data is changed, the code will still work.

```
[34]: # Input Data
      import pandas as pd
      consultants=['Alice', 'Bob', 'Charles', 'Daphne']
      projects=[1,2]
      fitness=pd.DataFrame([[5,2],[3,2],[4,5],[3,1]],index=consultants,columns=projects)
      senior=['Alice','Bob','Daphne']
      capacity=pd.Series([1,1,1,1],index=consultants)
      demand=pd.Series([2,2],index=projects)
      seniorDemand=pd.Series([1,1],index=projects)
```

```
[35]: # Creating variables
```

```
{('Alice', 1): <gurobi.Var x[Alice,1]>,
 ('Alice', 2): <gurobi.Var x[Alice,2]>,
 ('Bob', 1): <gurobi.Var x[Bob,1]>,
 ('Bob', 2): <gurobi.Var x[Bob,2]>,
 ('Charles', 1): <gurobi.Var x[Charles,1]>,
 ('Charles', 2): <gurobi.Var x[Charles,2]>,
 ('Daphne', 1): <gurobi.Var x[Daphne,1]>,
 ('Daphne', 2): <gurobi.Var x[Daphne,2]>}
```

```
[36]: # Objective
```

```
<gurobi.LinExpr: 5.0 x[Alice,1] + 2.0 x[Alice,2] + 3.0 x[Bob,1] + 2.0 x[Bob,2] + 4.0␣
↪x[Charles,1] + 5.0 x[Charles,2] + 3.0 x[Daphne,1] + x[Daphne,2]>
```

```
[37]: # (Alice) constraint
```

```
<gurobi.TempConstr: x[Alice,1] + x[Alice,2] <= 1>
```

```
[38]: # (Project 1 Total) constraint
```

```
<gurobi.TempConstr: x[Alice,1] + x[Bob,1] + x[Charles,1] + x[Daphne,1] >= 2>
```

```
[39]: # (Project 1 Senior) constraint
```

```
<gurobi.TempConstr: x[Alice,1] + x[Bob,1] + x[Daphne,1] >= 1>
```

[40]: # Full model

```
\ LP format - for model browsing. Use MPS format to capture full model detail.
Maximize
  5 x[Alice,1] + 2 x[Alice,2] + 3 x[Bob,1] + 2 x[Bob,2] + 4 x[Charles,1]
   + 5 x[Charles,2] + 3 x[Daphne,1] + x[Daphne,2]
Subject To
 Alice: x[Alice,1] + x[Alice,2] <= 1
 Bob: x[Bob,1] + x[Bob,2] <= 1
 Charles: x[Charles,1] + x[Charles,2] <= 1
 Daphne: x[Daphne,1] + x[Daphne,2] <= 1
 Project_1: x[Alice,1] + x[Bob,1] + x[Charles,1] + x[Daphne,1] >= 2
 Project_2: x[Alice,2] + x[Bob,2] + x[Charles,2] + x[Daphne,2] >= 2
 Project_1_Senior: x[Alice,1] + x[Bob,1] + x[Daphne,1] >= 1
 Project_2_Senior: x[Alice,2] + x[Bob,2] + x[Daphne,2] >= 1
Bounds
Binaries
 x[Alice,1] x[Alice,2] x[Bob,1] x[Bob,2] x[Charles,1] x[Charles,2]
 x[Daphne,1] x[Daphne,2]
End
```

[41]: # Numerical Solution

```
Maximum total fitness: 15.0
```

[42]: # Creating the output table

```
           1    2
Alice    NaN  NaN
Bob      NaN  NaN
Charles  NaN  NaN
Daphne   NaN  NaN
```

[43]: # Filling in the output table

```
         1  2
Alice    1  0
Bob      0  1
Charles  0  1
Daphne   1  0
```

[44]: # Final code

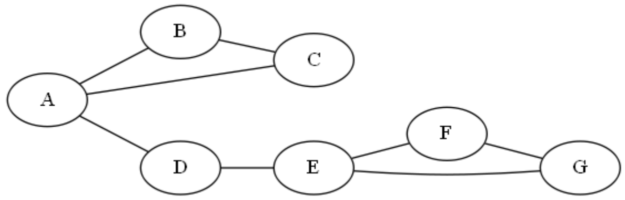
```
Maximum total fitness: 15.0

         1  2
Alice    1  0
Bob      0  1
Charles  0  1
Daphne   1  0
```

Exercise 10.4: Numerical Solution for Project Selection

Implement the linear optimization model for the “Project Selection” problem from last week.

Recap of the problem: Ebony is an ambitious master’s student who would like to maximize the number of extra-curricular business analytics projects she takes part of this year. However, projects may conflict with one another. The following graph summarizes the conflicts. (For example, project A conflicts with B, C and D, but projects B and D can be done together.)



Beside the conflict above,

- Project A is a prerequisite to project F (meaning that pursuing F requires also pursuing A.)
- Project B is a prerequisite to project G.

Formulate a linear optimization problem to help her decide which projects to pursue.

Concrete Formulation

Decision Variables:

X_i: whether to pursue project i. (Binary)

Objective and Constraints:

$$
\begin{aligned}
\text{Maximize} \quad & X_A + X_B + \cdots + X_G \\
\text{s.t.} \quad & \\
& X_A + X_B \leq 1 \\
& X_B + X_C \leq 1 \\
& X_A + X_C \leq 1 \\
& X_A + X_D \leq 1 \\
& X_D + X_E \leq 1 \\
& X_E + X_F \leq 1 \\
& X_F + X_G \leq 1 \\
& X_E + X_G \leq 1 \\
& X_A \geq X_F \\
& X_B \geq X_G
\end{aligned}
$$

Input data

[45]:
```python
projects=['A','B','C','D','E','F','G']
conflicts=[['A','B'],['B','C'],['A','C'],['A','D'],\
           ['D','E'],['E','F'],['F','G'],['E','G']]
prereqs=[['A','F'],['B','G']]
```

[46]:
```python
# Examples of looping through the above data structures
# Standard way
for pair in conflicts:
    print(f'{pair[0]} and {pair[1]} are in conflict.')
# Shortcut
for p1,p2 in prereqs:
    print(f'{p1} is a pre-requisite to {p2}')
```

```
A and B are in conflict.
B and C are in conflict.
A and C are in conflict.
A and D are in conflict.
```

```
D and E are in conflict.
E and F are in conflict.
F and G are in conflict.
E and G are in conflict.
A is a pre-requisite to F
B is a pre-requisite to G
```

Python Code

Write Python code to implement the above using Gurobi. Your code must obtain all data from the above input data structures, such that if new projects are added or the list of conflicts and pre-reqs change, the code will continue to work.

```
Optimal objective: 3.0
Optimal projects to pursue: B D G
```

Exercise 10.5: Numerical Solution for Food Production

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:

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

```
[48]: # Input data
      import pandas as pd
      months=range(1,7)
      price=pd.Series([150,160,180,170,180,160],index=months)
      usage=2000
      storage_capacity=1000
```

```
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
```

Exercise 10.6: Optimal Advertising Plan

In this exercise, you need to complete the English description, concrete formulation, and Gurobi code. It is not enough to have correct Python code; the correctness of the concrete formulation also counts.

SALS Marketing Inc. is developing an advertising campaign for a large consumer goods corporation. An advertising plan specifies how many units of each kind of advertisement to purchase. SALS has promised a plan that will yield the highest possible "exposure rating," which is a measure of the ability to reach the appropriate demographic group and generate demand. The options for advertisements with their respective costs (per unit of advertising) and per-unit exposure ratings are given in the table below (K stands for thousands).

Category	Subcategory	Cost/Unit	Exposure/Unit
Magazines	Literary	\$7.5 K	15 K
	News	\$10 K	22.5 K
	Topical	\$15 K	24 K
Newspapers	Morning	\$2 K	37.5 K
	Evening	\$3 K	75 K
Television	Morning	\$20 K	275 K
	Midday	\$10 K	180 K
	Evening	\$60 K	810 K
Radio	Morning	\$15 K	180 K
	Midday	\$15 K	17 K
	Evening	\$10 K	16 K

Of course, certain restrictions exist for the advertising campaign. The client corporation has budgeted 800K for the campaign, but to restrict overexposure to any particular audience it wants no more than 300K put into any one category (Magazine, Newspaper, etc.). Also, to ensure a broad range of exposure, at least 100K must be spent in each category. Finally, one has to purchase an integer number of units of each kind of advertisement, as no fractional units are allowed. Formulate and solve a linear optimization model to determine the optimal advertising plan.

English Description

Decision:

Objective:

Constraints:

Concrete Formulation

Decision variables:

Objective and constraints:

Python Code

Write Gurobi code to implement the above formulation. Your code should read in the data from the following data structures rather than hard code in the numbers. For convenience, all numerical values are in the units of K (thousands).

The outputs should be in the same format as the sample outputs below. Note: Gurobi might output strangely formatted numbers like `-0`, and you can make it `0` by converting it to `int`.

[50]:
```
# Constructing the subcategories
subcat={}
subcat['Magazines']=['Literary Mag.','News Mag.','Topical Mag.']
subcat['Newspapers']=['Morning News','Evening News']
subcat['Television']=['Morning TV','Midday TV','Evening TV']
subcat['Radio']=['Morning Radio','Midday Radio','Evening Radio']
subcat
```

```
{'Magazines': ['Literary Mag.', 'News Mag.', 'Topical Mag.'],
 'Newspapers': ['Morning News', 'Evening News'],
 'Television': ['Morning TV', 'Midday TV', 'Evening TV'],
 'Radio': ['Morning Radio', 'Midday Radio', 'Evening Radio']}
```

[51]:
```
import pandas as pd
allSubCat=subcat['Magazines']+subcat['Newspapers']+subcat['Television']+subcat['Radio']
data=pd.DataFrame([[7.5,15],[10,22.5],[15,24],\
                  [2,37.5],[3,75],
                  [20,275],[10,180],[60,810],\
                  [15,180],[15,17],[10,16]],\
                  index=allSubCat,columns=['Cost','Exposure'])
data
```

	Cost	Exposure
Literary Mag.	7.5	15.0
News Mag.	10.0	22.5
Topical Mag.	15.0	24.0
Morning News	2.0	37.5
Evening News	3.0	75.0
Morning TV	20.0	275.0
Midday TV	10.0	180.0
Evening TV	60.0	810.0
Morning Radio	15.0	180.0
Midday Radio	15.0	17.0
Evening Radio	10.0	16.0

[52]:
```
# Final Output
```

```
Maximum total exposure (in thousands): 14235.0
# of units to purchase:
        Literary Mag.: 0
        News Mag.: 10
        Topical Mag.: 0
        Morning News: 0
        Evening News: 98
        Morning TV: 0
        Midday TV: 30
        Evening TV: 0
        Morning Radio: 7
        Midday Radio: 0
        Evening Radio: 0
```