IEOR E4004: Optimization Models and Methods
Instructor: Dr. Yaren Bilge Kaya
Due: 10/06/2024, 11:59 pm

Assignment 2

September 30, 2024

Question 1 (35 Points). *Three airports in New York City run 24 hours a day, 7 days a week. In a given day, there are requirements for the total number of air traffic controllers that must be at the airports. These are given in Table 1.*

Air traffic controllers can either work 8-hour or 12-hour shifts, starting at the times stated earlier (12-hour shifts can start only at 12 a.m./p.m. or 8 a.m./p.m.). Those working 8-hour shifts cost $40/h in salary and benefits, and those working 12-hour shifts cost $35/h.

1. *Formulate and solve a linear program to minimize the dispatcher labor costs. [20 pts]*
2. *Suppose at most one-third of its controllers can work 12-hour shifts. Repeat (1).*
   You may ignore the requirement that the number of employees must be an integer. [15 p]

Hours	Controllers needed
12 a.m. to 4 a.m.	8
4 a.m. to 8 a.m.	10
8 a.m. to 12 p.m.	16
12 p.m. to 4 p.m.	21
4 p.m. to 8 p.m.	18
8 p.m. to 12 a.m.	12

Table 1: Number of controllers needed during different time intervals.

Assignment 2

September 30, 2024

Question 2 (10 Points)**.** *Consider the LP below and write out its dual. Solve the LP and its dual via your favorite solver. State whether the optimal solution for primal and dual problem yield to the same optimal value.*

$$
\begin{aligned}
max\ Z = \quad & 10x_1 + 14x_2 + 20x_3 \\
s.t. \quad & 2x_1 + 3x_2 + 4x_3 \leq 220 \\
& 4x_1 + 2x_2 - x_3 \leq 385 \\
& x_1 + 4x_3 \leq 160 \\
& x_1, x_2, x_3 \geq 0
\end{aligned}
$$

Assignment 2

Question 3 (55 Points). *Suppose that a lumberyard has a supply of 10-ft boards, which are cut into 3-ft, 4-ft, and 5-ft boards according to customer demand. The 10-ft boards can be cut into several sensible patterns, each in such a way that there the leftover material is less than 3-ft. The lumberyard just received an order for 90 3-ft boards, 60 4-ft boards, and 60 5-ft boards.*

1. *Determine all sensible patterns the lumberyard may use to cut the 10-ft boards. [10 pts]*
2. *The lumberyard would like to use as few 10-ft boards as possible in meeting the demand.*
   (a) *Write out a mathematical formulation for the problem as an integer linear program. [10 pts]*
   (b) *Solve this problem using Gurobi solver via the Python interface. What is the optimal number of each pattern, and what is the minimum number of boards to cut? [15 pts]*
3. *If it is possible to maintain the minimum number of 10-ft boards to be used, the lumberyard would also like to minimize the amount of scrap pieces leftover (those pieces smaller than 3-ft).*
   (a) *With respect to the previous solution, model this desire algebraically. [10 pts]*
   (b) *Modify the model and resolve it. Summarize the results here. [10 pts]*

Alexandre
Duhamel
afd2153
MSOR

HW2 : Optimization

Exercise 1:

1) Air trafic control where $\begin{cases} \text{8h shifts are paid } \$40/h \\ \text{12h shifts are paid } \$35/h \end{cases}$ starting at $\begin{cases} 12am/pm \\ 8am/pm \end{cases}$

The amount of controllers needed depend on the time of the day, following this table:

Hours	
12 a.m. to 4 a.m.	$x_5 + x_0 + y_0 + y_5 \geq 8$
4 a.m. to 8 a.m.	$x_0 + x_1 + y_0 + y_5 \geq 10$
8 a.m. to 12 p.m.	$x_1 + x_2 + y_1 + y_2 \geq 16$
12 p.m. to 4 p.m.	$x_2 + x_3 + y_2 + y_3 \geq 21$
4 p.m. to 8 p.m.	$x_3 + x_4 + y_1 + y_3 \geq 18$
8 p.m. to 12 a.m.	$x_4 + x_5 + y_3 + y_5 \geq 12$

We want to minimize the dispatcher labor costs.

Decision variables:
- x_i = number of workers on an 8h shift starting at time i, where $i = 0,1,2,3,4,5$
- y_i = number of workers on an 12h shift starting at time i, where $i = 0,2,3,5$

Objective function: $\min\ 40\times 8\sum_{i\in S} x_i + 35\times 12\sum_{i\in L} y_i$ where $\begin{cases} S = \{1,\dots,5\} & \text{short shift} \\ L = \{0,2,3,5\} & \text{long shift} \end{cases}$

Constraints:
$$\begin{cases} x_5 + x_0 + y_0 + y_5 \geq 8 \\ x_0 + x_1 + y_0 + y_5 \geq 10 \\ x_1 + x_2 + y_1 + y_2 \geq 16 \\ x_2 + x_3 + y_2 + y_3 \geq 21 \\ x_3 + x_4 + y_1 + y_3 \geq 18 \\ x_4 + x_5 + y_3 + y_5 \geq 12 \end{cases}$$

Non negativity: $\begin{cases} \forall i \in S,\ x_i \geq 0 \\ \forall i \in L,\ y_i \geq 0 \end{cases}$

Here is the code, solving the LP:

```
m = Model("Air Traffic Controllers")

time_intervals = ['12am to 4am', '4am to 8am', '8am to 12pm', '12pm to 4pm', '4pm to 8pm', '8pm to 12am']
requirements = [8, 10, 16, 21, 18, 12]

# Decision variables
x = {i: m.addVar(vtype=GRB.CONTINUOUS, name=f"x_{i}") for i in range(6)}  # 8-hour shifts
y = {i: m.addVar(vtype=GRB.CONTINUOUS, name=f"y_{i}") for i in [0, 2, 3, 5]}  # 12-hour shifts

# Objective function
m.setObjective(
    40 * 8 * quicksum(x[i] for i in range(6)) + 35 * 12 * quicksum(y[i] for i in [0, 2, 3, 5]),
    GRB.MINIMIZE
)

# Constraints:
m.addConstr(x[5] + x[0] + y[0] + y[5] >= requirements[0], "12am to 4am")
m.addConstr(x[0] + x[1] + y[0] + y[5] >= requirements[1], "4am to 8am")
m.addConstr(x[1] + x[2] + y[0] + y[2] >= requirements[2], "8am to 12pm")
m.addConstr(x[2] + x[3] + y[2] + y[3] >= requirements[3], "12pm to 4pm")
m.addConstr(x[3] + x[4] + y[2] + y[3] >= requirements[4], "4pm to 8pm")
m.addConstr(x[4] + x[5] + y[3] + y[5] >= requirements[5], "8pm to 12am")

# Optimize the model
m.setParam('OutputFlag', 0)  # cleaner output
m.optimize()

# Output results
if m.status == GRB.OPTIMAL:
    print("Optimal solution found:\n")
    print(f"8-hour shift starting at :")
    for i in range(6):
        print(f"\t {time_intervals[i]}: {x[i].x} controllers")
    print(f"12-hour shift starting at : ")
    for i in [0, 2, 3, 5]:
        print(f"\t {time_intervals[i]}: {y[i].x} controllers")
    print(f"\nTotal cost: ${m.objVal}")
else:
    print("No optimal solution found.")
```

```
Optimal solution found:

8-hour shift starting at :
         12am to 4am: 0.0 controllers
         4am to 8am: 2.0 controllers
         8am to 12pm: 3.0 controllers
         12pm to 4pm: 3.0 controllers
         4pm to 8pm: 0.0 controllers
         8pm to 12am: 0.0 controllers
12-hour shift starting at :
         12am to 4am: 8.0 controllers
         8am to 12pm: 3.0 controllers
         12pm to 4pm: 12.0 controllers
         8pm to 12am: 0.0 controllers

Total cost: $12220.0
```

2) Now we suppose that : all y workers $\leq \frac{1}{2}$ all workers

So :

$$\sum_{i \in L} y_i \leq \frac{1}{3} \left(\sum_{i \in S} x_i + \sum_{i \in L} y_i \right)$$

The code is:

```
# New constraint :
m.addConstr(
    quicksum(y[i] for i in [0, 2, 3, 5]) <= (1/3) *
    (quicksum(x[i] for i in range(6)) + quicksum(y[i] for i in [0, 2, 3, 5])),
    "12-hour shift limit"
)

# Optimize the model
m.setParam('OutputFlag', 0)  # cleaner output
m.optimize()

# Output results
if m.status == GRB.OPTIMAL:
    print("Optimal solution found:\n")
    print(f"8-hour shift starting at :")
    for i in range(6):
        print(f"\t {time_intervals[i]}: {x[i].x} controllers")
    print(f"12-hour shift starting at : ")
    for i in [0, 2, 3, 5]:
        print(f"\t {time_intervals[i]}: {y[i].x} controllers")
    print(f"\nTotal cost: ${m.objVal}")
else:
    print("No optimal solution found.")
```

```
Optimal solution found:

8-hour shift starting at :
         12am to 4am: 2.428571428571427 controllers
         4am to 8am: 2.0 controllers
         8am to 12pm: 8.42857142857143 controllers
         12pm to 4pm: 6.0 controllers
         4pm to 8pm: 5.428571428571431 controllers
         8pm to 12am: 0.0 controllers
12-hour shift starting at :
         12am to 4am: 5.571428571428573 controllers
         8am to 12pm: 0.0 controllers
         12pm to 4pm: 6.571428571428569 controllers
         8pm to 12am: 0.0 controllers

Total cost: $12871.428571428572
```

non integers, ok !!

adding a new constraint costs us more money which is very natural, because it's "harder" to solve the issue

Exercise 2:

We can translate it to: $\begin{cases} \max\ c^T x \\ Ax \le b \\ x \ge 0 \end{cases}$

The primal is: $\max Z = 10x_1 + 14x_2 + 20x_3$
s.t. $2x_1 + 3x_2 + 4x_3 \le 220$
$4x_1 + 2x_2 - x_3 \le 385$
$x_1 + 4x_3 \le 160$
$x_1, x_2, x_3 \ge 0$

So $A = \begin{pmatrix} 2 & 3 & 4 \\ 4 & 2 & -1 \\ 1 & 0 & 3 \end{pmatrix}$, $b = \begin{pmatrix} 220 \\ 385 \\ 160 \end{pmatrix}$ and $c = \begin{pmatrix} 10 \\ 14 \\ 20 \end{pmatrix}$

The dual is: $\begin{cases} \min b^T y \\ A^T y \ge c \\ y \ge 0 \end{cases}$ so $\begin{cases} \min 220y_1 + 385y_2 + 160y_3 \\ 2y_1 + 4y_2 + y_3 \ge 10 \\ 3y_1 + 2y_2 + 0 \ge 14 \\ 4y_1 - y_2 + 3y_3 \ge 20 \\ y_1, y_2, y_3 \ge 0 \end{cases}$

```python
# Model of the primal
primal_model = Model("Primal_LP")

# Define decision variables
x1 = primal_model.addVar(vtype=GRB.CONTINUOUS, name="x1")
x2 = primal_model.addVar(vtype=GRB.CONTINUOUS, name="x2")
x3 = primal_model.addVar(vtype=GRB.CONTINUOUS, name="x3")

# Objective function
primal_model.setObjective(10*x1 + 14*x2 + 20*x3, GRB.MAXIMIZE)

# Constraints :
primal_model.addConstr(2*x1 + 3*x2 + 4*x3 <= 220, "c1")
primal_model.addConstr(4*x1 + 2*x2 - x3 <= 385, "c2")
primal_model.addConstr(x1 + 4*x3 <= 160, "c3")

# Optimize the model
primal_model.setParam('OutputFlag', 0)  # cleaner output
primal_model.optimize()

# Output results
if primal_model.status == GRB.OPTIMAL:
    print(f"Primal optimal solution: Z = {primal_model.objVal}")
    print(f"x1 = {x1.x}, x2 = {x2.x}, x3 = {x3.x}")
else:
    print("No optimal solution found.")
```

```
Primal optimal solution: Z = 1100.0
x1 = 60.0, x2 = 0.0, x3 = 25.0
```

We have $Z_{primal} = Z_{dual}$
So the primal and the dual yield the same optimal value.

```python
# Model of the dual
dual_model = Model("Dual_LP")

# Define dual decision variables
y1 = dual_model.addVar(vtype=GRB.CONTINUOUS, name="y1")
y2 = dual_model.addVar(vtype=GRB.CONTINUOUS, name="y2")
y3 = dual_model.addVar(vtype=GRB.CONTINUOUS, name="y3")

# Objective function
dual_model.setObjective(220*y1 + 385*y2 + 160*y3, GRB.MINIMIZE)

# Constraints :
dual_model.addConstr(2*y1 + 4*y2 + y3 >= 10, "d1")
dual_model.addConstr(3*y1 + 2*y2 >= 14, "d2")
dual_model.addConstr(4*y1 - y2 + 4*y3 >= 20, "d3")

# Optimize the model
dual_model.setParam('OutputFlag', 0)  # cleaner output
dual_model.optimize()

# Output results
if dual_model.status == GRB.OPTIMAL:
    print(f"Dual optimal solution: Z_dual = {dual_model.objVal}")
    print(f"y1 = {y1.x}, y2 = {y2.x}, y3 = {y3.x}")
else:
    print("No optimal solution found.")
```

```
Dual optimal solution: Z_dual = 1100.0
y1 = 5.0, y2 = 0.0, y3 = 0.0
```

Exercise 3:

1) We can make 3, 4 and 5 feet cuts.

We need sum(lenght cutted boards) ≤ 10

1. 3 cuts of 3 feets
2. 2 cuts of 4 feet
3. 1 cut of 5 feet + 1 cut of 4 feet
4. 2 cuts of 5 feet
5. 1 cut of 3 feet + 1 cut of 5 feet
6. 2 cuts of 3 feet + 1 cut of 4 feet

2) a. <u>Decision variables:</u> x_i represents the number of times the pattern i is used

<u>Objective function:</u> $\min \sum_{i \in \mathcal{E}} x_i$ where $\mathcal{E} = \{1,2,3,4,5,6\}$

Let's define the demand as
$$\begin{cases} d_3 = 90 \text{ , 3 ft boards} \\ d_4 = 60 \text{ , 4 ft boards} \\ d_5 = 60 \text{ , 5 ft boards} \end{cases}$$

<u>Constraints:</u>
$$\begin{cases} 3x_1 + x_5 + 2x_6 \geq d_3 \\ 2x_2 + x_3 + x_6 \geq d_4 \\ x_3 + 2x_4 + x_5 \geq d_5 \end{cases}$$

<u>Non negativity:</u> $x_i \geq 0$ where $i \in \mathcal{E}$

```python
m = Model("Cutting_Stock_4_Patterns")

demand_3 = 90
demand_4 = 60
demand_5 = 60
```

b.

```python
# Decision variables
x1 = m.addVar(vtype=GRB.INTEGER, name="x1")  # Three 3-ft boards
x2 = m.addVar(vtype=GRB.INTEGER, name="x2")  # Two 4-ft boards
x3 = m.addVar(vtype=GRB.INTEGER, name="x3")  # One 5-ft and one 4-ft board
x4 = m.addVar(vtype=GRB.INTEGER, name="x4")  # Two 5-ft boards
x5 = m.addVar(vtype=GRB.INTEGER, name="x5")  # One 5-ft and one 3-ft board
x6 = m.addVar(vtype=GRB.INTEGER, name="x6")  # Two 3-ft boards and one 4-ft board

# Set objective function: minimize the number of 10-ft boards used
m.setObjective(x1 + x2 + x3 + x4 + x5 + x6, GRB.MINIMIZE)

# Demand constraints :
m.addConstr(3*x1 + x5 + 2*x6 >= demand_3, "3-ft boards constraint")
m.addConstr(2*x2 + x3 + x6 >= demand_4, "4-ft boards constraint")
m.addConstr(x3 + 2*x4 + x5 >= demand_5, "5-ft boards constraint")

# Optimize the model
m.setParam('OutputFlag', 0)  # cleaner output
m.optimize()

# Output the optimal solution
if m.status == GRB.OPTIMAL:
    print(f"Optimal solution found:")
    print(f"\t x1 (Three 3-ft boards): {x1.x}")
    print(f"\t x2 (Two 4-ft boards): {x2.x}")
    print(f"\t x3 (One 5-ft and one 4-ft board): {x3.x}")
    print(f"\t x4 (Two 5-ft boards): {x4.x}")
    print(f"\t x5 (One 3-ft and one 5-feet board): {x5.x}")
    print(f"\t x6 (Two 3-ft and one 4-feet board): {x6.x}")
    print(f"\nTotal 10-ft boards used: {m.objVal}")

    total_scrap = 1*x1.x + 2*x2.x + 1*x3.x + 0*x4.x + 2*x5.x + 0*x6.x
    print(f"Total scrap: {total_scrap} feet")
else:
    print("No optimal solution found.")
```

```
Optimal solution found:
	 x1 (Three 3-ft boards): -0.0
	 x2 (Two 4-ft boards): 8.0
	 x3 (One 5-ft and one 4-ft board): -0.0
	 x4 (Two 5-ft boards): 30.0
	 x5 (One 3-ft and one 5-feet board): -0.0
	 x6 (Two 3-ft and one 4-feet board): 45.0

Total 10-ft boards used: 83.0
Total scrap: 16.0 feet
```

3) a. The patterns scraps are:
1. 3 cuts of 3 feets → 1 ft scrap
2. 2 cuts of 4 feets → 2 ft scrap
3. 1 cut of 5 feet + 1 cut of 4 feet → 1 ft scrap
4. 2 cuts of 5 feets → 0 ft scrap
5. 1 cut of 3 feet + 1 cut of 5 feet → 2 ft scrap
6. 2 cuts of 3 feet + 1 cut of 4 feet → 0 ft scrap

Total scrap = $1 \times x_1 + 2 \times x_2 + 1 \times x_3 + 2x_5$

<u>Objective function:</u> min $1 \times x_1 + 2 \times x_2 + 1 \times x_3 + 2x_5$ given we know the previous Z

optimal number of lumber

New constraint: $\sum_{i \in \mathcal{E}} x_i = Z$

we keep the previous constraints

code before

⋮

b) That way, we're making sure that we're choosing the patterns that leave the least amount of scrap.

```
# Total number of 10-ft boards must be equal to N_star
m_scrap.addConstr(x1 + x2 + x3 + x4 + x5 + x6 == m.objVal, "fixed number of boards")

# Optimize the model
m_scrap.setParam('OutputFlag', 0)  # cleaner output
m_scrap.optimize()

# Output the optimal solution
if m.status == GRB.OPTIMAL:
    print(f"Optimal solution found:")
    print(f"\t x1 (Three 3-ft boards): {x1.x}")
    print(f"\t x2 (Two 4-ft boards): {x2.x}")
    print(f"\t x3 (One 5-ft and one 4-ft board): {x3.x}")
    print(f"\t x4 (Two 5-ft boards): {x4.x}")
    print(f"\t x5 (One 3-ft and one 5-feet board): {x5.x}")
    print(f"\t x6 (Two 3-ft and one 4-feet board): {x6.x}")
    print(f"\nTotal 10-ft boards used: {m.objVal}")

    total_scrap = 1*x1.x + 2*x2.x + 1*x3.x + 0*x4.x + 2*x5.x + 0*x6.x
    print(f"Total scrap: {total_scrap} feet")
else:
    print("No optimal solution found.")
```

```
Optimal solution found:
         x1 (Three 3-ft boards): -0.0
         x2 (Two 4-ft boards): 7.0
         x3 (One 5-ft and one 4-ft board): -0.0
         x4 (Two 5-ft boards): 30.0
         x5 (One 3-ft and one 5-feet board): -0.0
         x6 (Two 3-ft and one 4-feet board): 46.0

Total 10-ft boards used: 83.0
Total scrap: 14.0 feet
```

- Using this approach, we managed to use as many 10ft boards and reduce by 2ft the amount of scrap done by the cuts.