IEOR E4004 Assignment 2 Code

Mark Ma km4054

October 6, 2024

[1]:
```
import numpy as np
import gurobipy as gp
```

1 Question 1

1.1 1) Formulate and solve a linear program to minimize the dispatcher labor costs.

[2]:
```
# Create a model
model = gp.Model("q1.1")
```

```
Set parameter Username
Academic license - for non-commercial use only - expires 2025-09-09
```

[3]:
```
# Define decision variables
x = model.addVars(6, vtype=gp.GRB.INTEGER, name="x")  # 8-hour shift variables
y = model.addVars(4, vtype=gp.GRB.INTEGER, name="y")  # 12-hour shift variables
```

[4]:
```
# Objective function: minimize labor costs
model.setObjective(
    40 * 8 * sum(x[i] for i in range(6)) + 35 * 12 * sum(y[j] for j in␣
 ↪range(4)),
    gp.GRB.MINIMIZE
)
```

[5]:
```
# Add constraints based on the problem formulation
model.addConstr(x[0] + x[5] + y[0] + y[3] >= 8)
model.addConstr(x[0] + x[1] + y[0] + y[3] >= 10)
model.addConstr(x[1] + x[2] + y[0] + y[1] >= 16)
model.addConstr(x[2] + x[3] + y[1] + y[2] >= 21)
model.addConstr(x[3] + x[4] + y[1] + y[2] >= 18)
model.addConstr(x[4] + x[5] + y[2] + y[3] >= 12)

# Non-negativity constraints:
for i in range(6):
    model.addConstr(x[i] >= 0)
for j in range(4):
    model.addConstr(y[j] >= 0)
```

[6]:
```
model.optimize()
```

```
Gurobi Optimizer version 11.0.3 build v11.0.3rc0 (win64 - Windows 10.0
(19045.2))

CPU model: Intel(R) Core(TM) i7-10875H CPU @ 2.30GHz, instruction set
[SSE2|AVX|AVX2]
Thread count: 8 physical cores, 16 logical processors, using up to 16 threads

Optimize a model with 16 rows, 10 columns and 34 nonzeros
Model fingerprint: 0xa99bbcf8
Variable types: 0 continuous, 10 integer (0 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [3e+02, 4e+02]
  Bounds range     [0e+00, 0e+00]
  RHS range        [8e+00, 2e+01]
Found heuristic solution: objective 13020.000000
Presolve removed 10 rows and 0 columns
Presolve time: 0.00s
Presolved: 6 rows, 10 columns, 24 nonzeros
Variable types: 0 continuous, 10 integer (0 binary)
Found heuristic solution: objective 12720.000000

Root relaxation: objective 1.222000e+04, 7 iterations, 0.00 seconds (0.00 work
units)

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

*    0     0               0    12220.000000 12220.0000  0.00%     -    0s

Explored 1 nodes (7 simplex iterations) in 0.01 seconds (0.00 work units)
Thread count was 16 (of 16 available processors)

Solution count 3: 12220 12720 13020

Optimal solution found (tolerance 1.00e-04)
Best objective 1.222000000000e+04, best bound 1.222000000000e+04, gap 0.0000%
```

[10]:
```
if model.status == gp.GRB.OPTIMAL:
    print("Optimal solution found:")
    print(f"x (8-hour shift): {[x[i].x for i in range(6)]}")
    print(f"y (12-hour shift): {[y[j].x for j in range(4)]}")
    print(f"Total labor cost: ${model.objVal:.2f}")
else:
    print("No optimal solution found.")
```

```
Optimal solution found:
x (8-hour shift): [-0.0, 2.0, 3.0, 3.0, -0.0, -0.0]
y (12-hour shift): [8.0, 3.0, 12.0, -0.0]
Total labor cost: $12220.00
```

1.2 2 Suppose at most one-third of its controllers can work 12-hour shifts. Repeat (1).

[15]:
```
# Create a model
model = gp.Model("q1.2")

# Define decision variables
x = model.addVars(6, vtype=gp.GRB.CONTINUOUS, name="x")  # 8-hour shift␣
 ↪variables
y = model.addVars(4, vtype=gp.GRB.CONTINUOUS, name="y")  # 12-hour shift␣
 ↪variables

model.setObjective(
    40 * 8 * sum(x[i] for i in range(6)) + 35 * 12 * sum(y[j] for j in␣
 ↪range(4)),
    gp.GRB.MINIMIZE
)

# Add constraints based on the problem formulation
model.addConstr(x[0] + x[5] + y[0] + y[3] >= 8)
model.addConstr(x[0] + x[1] + y[0] + y[3] >= 10)
model.addConstr(x[1] + x[2] + y[0] + y[1] >= 16)
model.addConstr(x[2] + x[3] + y[1] + y[2] >= 21)
model.addConstr(x[3] + x[4] + y[1] + y[2] >= 18)
model.addConstr(x[4] + x[5] + y[2] + y[3] >= 12)

# Non-negativity constraints:
for i in range(6):
    model.addConstr(x[i] >= 0)
for j in range(4):
    model.addConstr(y[j] >= 0)

# Additional Constraint:
model.addConstr(gp.quicksum(x[i] for i in range(6)) >= 2*gp.quicksum(y[j] for j␣
 ↪in range(4)))

model.optimize()
```

```
Gurobi Optimizer version 11.0.3 build v11.0.3rc0 (win64 - Windows 10.0
(19045.2))

CPU model: Intel(R) Core(TM) i7-10875H CPU @ 2.30GHz, instruction set
[SSE2|AVX|AVX2]
```

```
Thread count: 8 physical cores, 16 logical processors, using up to 16 threads

Optimize a model with 17 rows, 10 columns and 44 nonzeros
Model fingerprint: 0xfef9b289
Coefficient statistics:
  Matrix range     [1e+00, 2e+00]
  Objective range  [3e+02, 4e+02]
  Bounds range     [0e+00, 0e+00]
  RHS range        [8e+00, 2e+01]
Presolve removed 10 rows and 0 columns
Presolve time: 0.00s
Presolved: 7 rows, 10 columns, 34 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    0.0000000e+00   8.500000e+01   0.000000e+00      0s
       7    1.2871429e+04   0.000000e+00   0.000000e+00      0s

Solved in 7 iterations and 0.01 seconds (0.00 work units)
Optimal objective  1.287142857e+04
```

[16]:
```python
if model.status == gp.GRB.OPTIMAL:
    print("Optimal solution found:")
    print(f"x (8-hour shift): {[x[i].x for i in range(6)]}")
    print(f"y (12-hour shift): {[y[j].x for j in range(4)]}")
    print(f"Total labor cost: ${model.objVal:.2f}")
else:
    print("No optimal solution found.")
```

```
Optimal solution found:
x (8-hour shift): [2.428571428571428, 2.0, 8.428571428571429, 6.0,
5.428571428571429, 0.0]
y (12-hour shift): [5.571428571428572, 0.0, 6.571428571428571, 0.0]
Total labor cost: $12871.43
```

2 Question 2

2.1 1) Primal problem

[9]:
```python
model = gp.Model("q2.1")

# Define decision variables
x = model.addVars(3, vtype = gp.GRB.CONTINUOUS, name = "x")

# Objective function:
model.setObjective(10*x[0] + 14*x[1] + 20*x[2], gp.GRB.MAXIMIZE)

# Add constraints based on the problem formulation
model.addConstr(2*x[0] + 3*x[1] + 4*x[2] <= 220)
```

```
model.addConstr(4*x[0] + 2*x[1] - x[2] <= 385)
model.addConstr(x[0] + 4*x[2] <= 160)

# Non-negativity constraints:
for i in range(3):
    model.addConstr(x[i] >= 0)

model.optimize()
```

```
Gurobi Optimizer version 11.0.3 build v11.0.3rc0 (win64 - Windows 10.0
(19045.2))

CPU model: Intel(R) Core(TM) i7-10875H CPU @ 2.30GHz, instruction set
[SSE2|AVX|AVX2]
Thread count: 8 physical cores, 16 logical processors, using up to 16 threads

Optimize a model with 6 rows, 3 columns and 11 nonzeros
Model fingerprint: 0xc75614d9
Coefficient statistics:
  Matrix range     [1e+00, 4e+00]
  Objective range  [1e+01, 2e+01]
  Bounds range     [0e+00, 0e+00]
  RHS range        [2e+02, 4e+02]
Presolve removed 3 rows and 0 columns
Presolve time: 0.00s
Presolved: 3 rows, 3 columns, 8 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    1.1000000e+03   3.437500e+00   0.000000e+00      0s
       1    1.1000000e+03   0.000000e+00   0.000000e+00      0s

Solved in 1 iterations and 0.01 seconds (0.00 work units)
Optimal objective  1.100000000e+03
```

[10]:
```
if model.status == gp.GRB.OPTIMAL:
    print("Optimal solution found:")
    print(f"x: {[x[i].x for i in range(3)]}")
    print(f"Maximized Value: {model.objVal:.2f}")
else:
    print("No optimal solution found.")
```

```
Optimal solution found:
x: [97.77777777777777, 0.0, 6.111111111111114]
Maximized Value: 1100.00
```

2.2 2) Dual problem

[11]:
```
model = gp.Model("q2.2")

# Define decision variables
y = model.addVars(3, vtype = gp.GRB.CONTINUOUS, name = "y")

# Objective function:
model.setObjective(220*y[0] + 385*y[1] + 160*y[2], gp.GRB.MINIMIZE)

# Add constraints based on the problem formulation
model.addConstr(2*y[0] + 4*y[1] + y[2] >= 10)
model.addConstr(3*y[0] + 2*y[1] >= 14)
model.addConstr(4*y[0] - y[1] + 4*y[2] >= 20)

# Non-negativity constraints:
for i in range(3):
    model.addConstr(y[i] >= 0)

model.optimize()
```

```
Gurobi Optimizer version 11.0.3 build v11.0.3rc0 (win64 - Windows 10.0
(19045.2))

CPU model: Intel(R) Core(TM) i7-10875H CPU @ 2.30GHz, instruction set
[SSE2|AVX|AVX2]
Thread count: 8 physical cores, 16 logical processors, using up to 16 threads

Optimize a model with 6 rows, 3 columns and 11 nonzeros
Model fingerprint: 0xfb1fc6f3
Coefficient statistics:
  Matrix range     [1e+00, 4e+00]
  Objective range  [2e+02, 4e+02]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+01, 2e+01]
Presolve removed 3 rows and 0 columns
Presolve time: 0.00s
Presolved: 3 rows, 3 columns, 8 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    0.0000000e+00   2.900000e+01   0.000000e+00      0s
       3    1.1000000e+03   0.000000e+00   0.000000e+00      0s

Solved in 3 iterations and 0.01 seconds (0.00 work units)
Optimal objective  1.100000000e+03
```

[12]:
```
if model.status == gp.GRB.OPTIMAL:
    print("Optimal solution found:")
```

```
    print(f"y: {[y[i].x for i in range(3)]}")
    print(f"Minimized Value: {model.objVal:.2f}")
else:
    print("No optimal solution found.")
```

```
Optimal solution found:
y: [5.0, 0.0, 0.0]
Minimized Value: 1100.00
```

3 Question 3

3.1 1. 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?

[17]:
```
model = gp.Model("q3.2")

# Define decision variables
x = model.addVars(6, vtype = gp.GRB.INTEGER, name = "x")

# Objective function:
model.setObjective(gp.quicksum(x[i] for i in range(6)), gp.GRB.MINIMIZE)

# Add constraints based on the problem formulation
model.addConstr(3*x[0] + 2*x[1] + x[3] >= 90)
model.addConstr(x[1] + 2*x[2] + x[4] >= 60)
model.addConstr(x[3] + x[4] + 2*x[5] >= 60)

# Non-negativity constraints:
for i in range(6):
    model.addConstr(x[i] >= 0)

model.optimize()
```

```
Gurobi Optimizer version 11.0.3 build v11.0.3rc0 (win64 - Windows 10.0
(19045.2))

CPU model: Intel(R) Core(TM) i7-10875H CPU @ 2.30GHz, instruction set
[SSE2|AVX|AVX2]
Thread count: 8 physical cores, 16 logical processors, using up to 16 threads

Optimize a model with 9 rows, 6 columns and 15 nonzeros
Model fingerprint: 0xd42d0d5f
Variable types: 0 continuous, 6 integer (0 binary)
Coefficient statistics:
  Matrix range     [1e+00, 3e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [0e+00, 0e+00]
```

```
  RHS range        [6e+01, 9e+01]
Found heuristic solution: objective 150.0000000
Presolve removed 6 rows and 0 columns
Presolve time: 0.00s
Presolved: 3 rows, 6 columns, 9 nonzeros
Variable types: 0 continuous, 6 integer (0 binary)

Root relaxation: objective 8.250000e+01, 4 iterations, 0.00 seconds (0.00 work
units)

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

     0     0   82.50000    0    1  150.00000   82.50000  45.0%     -    0s
H    0     0                      83.0000000   82.50000  0.60%     -    0s
     0     0   82.50000    0    1   83.00000   82.50000  0.60%     -    0s

Explored 1 nodes (4 simplex iterations) in 0.02 seconds (0.00 work units)
Thread count was 16 (of 16 available processors)

Solution count 2: 83 150

Optimal solution found (tolerance 1.00e-04)
Best objective 8.300000000000e+01, best bound 8.300000000000e+01, gap 0.0000%
```

[18]:
```python
if model.status == gp.GRB.OPTIMAL:
    print("Optimal solution found:")
    print(f"x: {[x[i].x for i in range(6)]}")
    print(f"Minimized Value: {model.objVal:.2f}")
else:
    print("No optimal solution found.")
```

```
Optimal solution found:
x: [-0.0, 46.0, 7.0, -0.0, -0.0, 30.0]
Minimized Value: 83.00
```

[4]:
```python
# Create a model
model = gp.Model("q3.2")

# Define decision variables
x = model.addVars(6, vtype=gp.GRB.INTEGER, name="x")

# Objective function: minimize the number of 10-ft boards used
model.setObjective(gp.quicksum(x[i] for i in range(6)), gp.GRB.MINIMIZE)

# Add constraints based on the problem formulation
model.addConstr(3 * x[0] + 2 * x[1] + x[3] >= 90)  # Demand for 3-ft boards
model.addConstr(x[1] + 2 * x[2] + x[4] >= 60)      # Demand for 4-ft boards
```

```python
model.addConstr(x[3] + x[4] + 2 * x[5] >= 60)      # Demand for 5-ft boards

# Non-negativity constraints (implicitly handled by integer type):
for i in range(6):
    model.addConstr(x[i] >= 0)

# Set parameters to find multiple solutions
model.setParam(gp.GRB.Param.PoolSearchMode, 2)  # Focus on finding multiple␣
 ↪solutions
model.setParam(gp.GRB.Param.PoolSolutions, 100)  # Limit to 100 solutions

# Optimize the model
model.optimize()
```

```
Set parameter PoolSearchMode to value 2
Set parameter PoolSolutions to value 100
Gurobi Optimizer version 11.0.3 build v11.0.3rc0 (win64 - Windows 10.0
(19045.2))

CPU model: Intel(R) Core(TM) i7-10875H CPU @ 2.30GHz, instruction set
[SSE2|AVX|AVX2]
Thread count: 8 physical cores, 16 logical processors, using up to 16 threads

Optimize a model with 9 rows, 6 columns and 15 nonzeros
Model fingerprint: 0xd42d0d5f
Variable types: 0 continuous, 6 integer (0 binary)
Coefficient statistics:
  Matrix range     [1e+00, 3e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [0e+00, 0e+00]
  RHS range        [6e+01, 9e+01]
Found heuristic solution: objective 150.0000000
Presolve removed 6 rows and 0 columns
Presolve time: 0.00s
Presolved: 3 rows, 6 columns, 9 nonzeros
Variable types: 0 continuous, 6 integer (0 binary)
Found heuristic solution: objective 90.0000000

Root relaxation: objective 8.250000e+01, 3 iterations, 0.00 seconds (0.00 work
units)

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

     0     0   82.50000    0    1   90.00000   82.50000  8.33%     -    0s
H    0     0                      83.0000000   82.50000  0.60%     -    0s
     0     0   82.50000    0    1   83.00000   82.50000  0.60%     -    0s
```

```
Optimal solution found at node 0 - now completing solution pool…

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

     0     0   82.50000    0    1          -   82.50000      -     -    0s
     0     2   82.50000    0    1          -   82.50000      -     -    0s

Cutting planes:
  Gomory: 1

Explored 3340 nodes (1627 simplex iterations) in 0.05 seconds (0.00 work units)
Thread count was 16 (of 16 available processors)

Solution count 100: 83 83 83 … 84
No other solutions better than 84

Optimal solution found (tolerance 1.00e-04)
Best objective 8.300000000000e+01, best bound 8.300000000000e+01, gap 0.0000%
```

[5]:
```python
# Output all the solutions found
solution_count = min(10, model.SolCount)  # Number of solutions in the solution pool
print(f"Number of solutions found: {solution_count}")

if model.Status == gp.GRB.OPTIMAL or model.SolCount > 0:
    print('Optimal objective value: %g' % model.objVal)
    for i in range(min(10,solution_count)):
        model.setParam(gp.GRB.Param.SolutionNumber, i)
        print(f"Solution {i + 1}: {model.PoolObjVal}")
        print(f"x: {[x[j].xn for j in range(6)]}")
```

```
Number of solutions found: 10
Optimal objective value: 83
Solution 1: 83.0
x: [0.0, 46.0, 6.0, 0.0, 2.0, 29.0]
Solution 2: 82.99999999999999
x: [-0.0, 45.0, 7.0, -0.0, 1.9999999999999662, 29.000000000000018]
Solution 3: 83.0
x: [-0.0, 45.0, 8.0, -0.0, -0.0, 30.0]
Solution 4: 83.0
x: [-0.0, 46.0, 7.0, -0.0, -0.0, 30.0]
Solution 5: 83.0
x: [-0.0, 45.0, 7.0000000000000036, 1.0, 0.9999999999999929, 29.0]
Solution 6: 83.0
x: [-0.0, 45.0, 7.0000000000000036, -0.0, 0.9999999999999929, 30.0]
Solution 7: 83.0
```

```
x: [-0.0, 45.0, 6.0, -0.0, 3.0, 29.0]
Solution 8: 83.0
x: [-0.0, 45.0, 6.0, -0.0, 4.0, 28.0]
Solution 9: 83.0
x: [-0.0, 46.0, 5.0, -0.0, 4.0, 28.0]
Solution 10: 83.0
x: [1.0, 44.0, 7.0, -0.0, 2.0, 29.0]
```

3.2 2. Modify the model and resolve it. Summarize the results here

[9]:
```python
import gurobipy as gp

# Create a model
model = gp.Model("q3.3")

# Define decision variables (number of times each pattern is used)
x = model.addVars(6, vtype=gp.GRB.INTEGER, name="x")

# Objective function: minimize the number of 10-ft boards and then the total␣
 ↪scrap
model.setObjective((x[0] + 2 * x[2] + 2 * x[3] + x[4]), gp.GRB.MINIMIZE)

# Add constraints for meeting demand
model.addConstr(gp.quicksum(x[i] for i in range(6)) <= 83)
model.addConstr(3 * x[0] + 2 * x[1] + x[3] >= 90, "3ft_board_demand")
model.addConstr(x[1] + 2 * x[2] + x[4] >= 60, "4ft_board_demand")
model.addConstr(x[3] + x[4] + 2 * x[5] >= 60, "5ft_board_demand")

# Non-negativity constraints
for i in range(6):
    model.addConstr(x[i] >= 0)

# Optimize the model
model.optimize()
```

```
Gurobi Optimizer version 11.0.3 build v11.0.3rc0 (win64 - Windows 10.0
(19045.2))

CPU model: Intel(R) Core(TM) i7-10875H CPU @ 2.30GHz, instruction set
[SSE2|AVX|AVX2]
Thread count: 8 physical cores, 16 logical processors, using up to 16 threads

Optimize a model with 10 rows, 6 columns and 21 nonzeros
Model fingerprint: 0x380bd254
Variable types: 0 continuous, 6 integer (0 binary)
Coefficient statistics:
  Matrix range     [1e+00, 3e+00]
  Objective range  [1e+00, 2e+00]
```

```
  Bounds range     [0e+00, 0e+00]
  RHS range        [6e+01, 9e+01]
Presolve removed 6 rows and 0 columns
Presolve time: 0.00s
Presolved: 4 rows, 6 columns, 15 nonzeros
Variable types: 0 continuous, 6 integer (0 binary)

Root relaxation: objective 1.400000e+01, 2 iterations, 0.00 seconds (0.00 work
units)

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

*    0     0               0      14.0000000   14.00000  0.00%     -    0s

Explored 1 nodes (2 simplex iterations) in 0.01 seconds (0.00 work units)
Thread count was 16 (of 16 available processors)

Solution count 1: 14

Optimal solution found (tolerance 1.00e-04)
Best objective 1.400000000000e+01, best bound 1.400000000000e+01, gap 0.0000%
```

[12]:
```python
# Print the results
if model.status == gp.GRB.OPTIMAL:
    print("Optimal solution found:")
    for i in range(6):
        print(f"Pattern {i+1} (x[{i}]): {x[i].x}")
    print(f"Total number of 10-ft boards: {sum([x[i].x for i in range(6)]):.
 ↪0f}")
    total_scrap = x[0].x + 2 * x[2].x + 2 * x[3].x + x[4].x
    print(f"Total scrap generated: {total_scrap} inches")
else:
    print("No optimal solution found.")
```

```
Optimal solution found:
Pattern 1 (x[0]): -0.0
Pattern 2 (x[1]): 46.0
Pattern 3 (x[2]): 7.0
Pattern 4 (x[3]): -0.0
Pattern 5 (x[4]): -0.0
Pattern 6 (x[5]): 30.0
Total number of 10-ft boards: 83
Total scrap generated: 14.0 inches
```