IEOR 4004
Prof. Kroer

Prob #4 collaborated with Ming Jin Date .

Zihui Zhou
HW#6

1. 1) * define the variables:

let X_i denotes the number of trucks on route i.

let C_k denotes binary variables $\in \{0,1\}$

$i \in \{1,2,3,4,5,6,7,8\}$, $k \in \{1,2,3,4,5\}$

* create LP:

objective: max { Revenue: $(190X_1 + 200X_2 + 100X_3 + 360X_4 + 400X_5 + 150X_6 + 570X_7 + 70X_8)$

Binary fixed cost: $-(1000C_1 + 3000C_2 + 700C_3 + 2000C_4 + 1500C_5)$

Right per-truck cost: $-[102(X_1+X_3) + 88X_2 + 157(X_4+X_5+X_7) + 224(X_6+X_8)]$ }

constraints: $X_1+X_2 \le MC_1$; $X_1+X_2 \ge C_1$

$X_3+X_4 \le MC_2$; $X_3+X_4 \ge C_2$

$X_5+X_6 \le MC_3$; $X_5+X_6 \ge C_3$

$X_7 \le MC_4$; $X_7 \ge C_4$

$X_8 \le MC_5$; $X_8 \ge C_5$

total 100 trucks. $X_1+X_2+X_3+X_4+X_5+X_6+X_7+X_8 \le 100$

$X_i \le 30$

$C_i \in \{0,1\}$

M is large enough. for convenience, set to 100.

2) Using Gurobi to solve LP, we achieve:

$X_1 = 10$ → route s-1-1-t

$X_2 = 30$ → route s-1-2-t

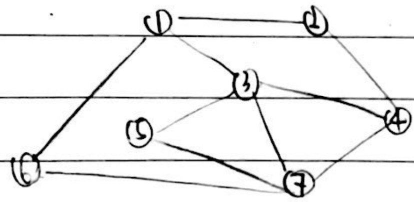
$X_5 = 30$ → route s-3-3-t

$X_7 = 30$ → route s-4-3-t

$C_1 = C_3 = C_4 = 1$

with objective value = 20220

2.



1) * define variables:

p_v: profit gained through broadcasting in city v

x_i: binary variable $= \begin{cases} 1 & \text{if we use frequency } i \\ 0 & \text{otherwise.} \end{cases}$ $\quad i \in \{1,2,3,4,5,6,7\}$

f_{vi} = binary variable $= \begin{cases} 1 & \text{if we use freq } i \text{ in city } c \\ 0 & \text{otherwise} \end{cases}$

* writing out LP

obj: $\max\left[\sum_{v=1}^{7}\left(\sum_{i=1}^{7} f_{vi}\right) \times p_v\right] - 500000 \sum_{i=1}^{7} x_i$ 有这个freq.

binary ⇒ constraints: $f_{vi} \le x_i \quad \forall\, v, i$ ← use freq i in city v only if $x_i = 1$

相邻城市不同freq ⇒ $f_{mi} + f_{ni} \le 1 \quad \forall j.\ (m,n) \in E = \{(1,2)(1,3)(1,6)(2,4)(5,3)(3,4)(3,7)(5,7)(4,7)(6,7)\}$

每个城市最多一个frequency ⇒ $\sum_{i=1}^{7} f_{vi} \le 1 \quad \forall\, v \in \{1,2,3,4,5,6,7\}$

$x_i,\ f_{vi} \in \{0,1\}$

2). Since $p_1 = p_2 = p_7 = 300000$

$p_3 = 450000$

$p_4 = p_5 = 100000$

$p_6 = 400000$

Then the LP becomes:

obj: max $\left(\sum_{i=1}^{7} f_{1i}\right)\times 3\times10^5 + \left(\sum_{i=1}^{7} f_{2i}\right)\times 3\times10^5 + \left(\sum_{i=1}^{7} f_{3i}\right)\times 4.5\times10^5$
$+ \left(\sum_{i=1}^{7} f_{4i}\right)\times 10^5 + \left(\sum_{i=1}^{7} f_{5i}\right)\times 10^5 + \left(\sum_{i=1}^{7} f_{6i}\right)\times 4\times10^5 + \left(\sum_{i=1}^{7} f_{7i}\right)\times 3\times10^5$
$- 5\times10^5 \sum_{i=1}^{7} x_i$

constraints: remains the same as 2/1)

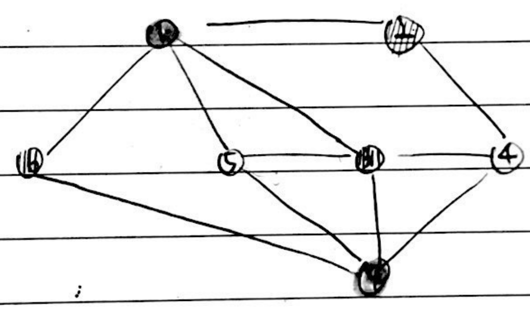
Solving the LP using Gurobi gives:

$f_{11} = f_{71} = f_{26} = f_{36} = f_{66} = 1$ other variables $= 0$

$x_1 = x_6 = 1$

with objective value 750000

visualization:



denotes frequency 1

denotes frequency 6

3 * define variables:

x_h: # of homes will be built

x_a: # of apartments that will be built

$t = \begin{cases} 1 & \text{if swimming-tennis complex will be built} \\ 0 & \text{otherwise} \end{cases}$

$m = \begin{cases} 1 & \text{if sailboat marina will be built} \\ 0 & \text{otherwise} \end{cases}$

$c =$ binary variable $\in \{0,1\}$

* writting out the LP:

obj: $(48000-40000)x_a + (46000-40000)x_h - (1.2\times10^6 m + 2.8\times10^6 t)$

constraints: $x_a + x_h \le 10000$

$t \le c;\ m \le 1-c$ t either t or m equals to 1

$x_h \ge 7 \cdot x_a \cdot m$

* 注意为1！限制 需有一个为1 $t + m = 1$

$x_a, x_h, m, t, c \ge 0$

$m, t, c \le 1$

Using Gurobi to solve the LP, gives the results:

$x_a = 10000$

$c = t = 1$

$x_h = m = 0$

objective value: 77200000

⇒ Lotus Point maximise net profits when it builds a sailboat marina, not the swimming-tennis complex, and build all 10000 dwelling units as apartments, which achieves the profit: \$77 200000

Date . .

4.1) Using Dynamic Programming to solve the problem

* define variables:

m: # of coins

n: (possible) values that could be created by m coins.

$$f(m,n) = \begin{cases} 1 & \text{if we could create } n \text{ value using } m \text{ coins} \\ 0 & \text{otherwise} \end{cases}$$

* setup:

$$f(m,n) = \begin{cases} 0 & m=0 \\ 1 & m=1,\ n \in \{1, 2, 5\} \\ \max\{f(m-1, n-1),\ f(m-1, n-2),\ f(m-1, n-5),\ f(m-1, n)\} & \text{for } i, j \geq 1 \end{cases}$$

* the smallest value that can't be created and fit into the wallet is \$23

see the following dataframe:

	0	1	2	3	4	5	6	7	8	9	...	15	16	17	18	19	20	21	22	23	24
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	1.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	...	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	...	1.0	1.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
5	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	...	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	0.0

2). Using Python to solve the problem described in coins.dat gives \$1509 as the smallest value that cannot be created and fit into the wallet.

Problem#1

December 9, 2019

[1]:
```
from gurobipy import *


# create a model
m = Model()

# create variables
x1 = m.addVar(vtype=GRB.CONTINUOUS, name="x1", lb=0, ub=30)
x2 = m.addVar(vtype=GRB.CONTINUOUS, name="x2", lb=0, ub=30)
x3 = m.addVar(vtype=GRB.CONTINUOUS, name="x3", lb=0, ub=30)
x4 = m.addVar(vtype=GRB.CONTINUOUS, name="x4", lb=0, ub=30)
x5 = m.addVar(vtype=GRB.CONTINUOUS, name="x5", lb=0, ub=30)
x6 = m.addVar(vtype=GRB.CONTINUOUS, name="x6", lb=0, ub=30)
x7 = m.addVar(vtype=GRB.CONTINUOUS, name="x7", lb=0, ub=30)
x8 = m.addVar(vtype=GRB.CONTINUOUS, name="x8", lb=0, ub=30)
c1 = m.addVar(vtype=GRB.BINARY, name="c1", lb=0, ub=1)
c2 = m.addVar(vtype=GRB.BINARY, name="c2", lb=0, ub=1)
c3 = m.addVar(vtype=GRB.BINARY, name="c3", lb=0, ub=1)
c4 = m.addVar(vtype=GRB.BINARY, name="c4", lb=0, ub=1)
c5 = m.addVar(vtype=GRB.BINARY, name="c5", lb=0, ub=1)


# integrate new variables
m.update()

# set objective
m.setObjective(
    (190*x1 + 200*x2 + 100*x3 + 300*x4 + 400*x5 + 150*x6 + 570*x7 + 70*x8)
    - (1000*c1 + 3000*c2 + 700*c3 + 2000*c4 + 1500*c5)
    - (102*(x1 + x3) + 88*x2 + 157*(x4 + x5 + x7) + 234*(x6 + x8)),
    GRB.MAXIMIZE
)

# add constraints
# m.addConstr(x1 <= 30*c1)
# m.addConstr(x2 <= 30*c1)
# m.addConstr(x3 <= 30*c2)
```

```python
# m.addConstr(x4 <= 30*c2)
# m.addConstr(x5 <= 30*c3)
# m.addConstr(x6 <= 30*c3)
# m.addConstr(x7 <= 30*c4)
# m.addConstr(x8 <= 30*c5)
m.addConstr(x1+x2 <= 100*c1)
m.addConstr(x1+x2 >= c1)
m.addConstr(x3+x4 <= 100*c2)
m.addConstr(x3+x4 >= c2)
m.addConstr(x5+x6 <= 100*c3)
m.addConstr(x5+x6 >= c3)
m.addConstr(x7 <= 100*c4)
m.addConstr(x7 >= c4)
m.addConstr(x8 <= 100*c5)
m.addConstr(x8 >= c5)
m.addConstr(x1+x2+x3+x4+x5+x6+x7+x8 <= 100)

# optimize
m.optimize()
print("Model status: ", m.status)

# print out decision variables
for v in m.getVars():
    print(v.varName, v.x, "\n")

print("-"*15)
print("Obj Value: ", m.objVal)
```

```
Academic license - for non-commercial use only
Optimize a model with 11 rows, 13 columns and 34 nonzeros
Variable types: 8 continuous, 5 integer (5 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+02]
  Objective range  [2e+00, 3e+03]
  Bounds range     [1e+00, 3e+01]
  RHS range        [1e+02, 1e+02]
Found heuristic solution: objective -0.0000000
Presolve removed 2 rows and 2 columns
Presolve time: 0.00s
Presolved: 9 rows, 11 columns, 30 nonzeros
Variable types: 7 continuous, 4 integer (4 binary)

Root relaxation: objective 2.035643e+04, 1 iterations, 0.00 seconds

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

```
     0     0 20356.4268    0    2   -0.00000 20356.4268      -     -    0s
H    0     0                     19949.230769 20356.4268  2.04%     -    0s
*    0     0               0    20220.000000 20220.0000  0.00%     -    0s

Explored 1 nodes (3 simplex iterations) in 0.08 seconds
Thread count was 4 (of 4 available processors)

Solution count 3: 20220 19949.2 -0

Optimal solution found (tolerance 1.00e-04)
Best objective 2.022000000000e+04, best bound 2.022000000000e+04, gap 0.0000%
Model status:  2
x1 10.0

x2 30.0

x3 0.0

x4 0.0

x5 30.0

x6 0.0

x7 30.0

x8 0.0

c1 1.0

c2 0.0

c3 1.0

c4 1.0

c5 0.0

---------------
Obj Value:  20220.0
```

[ ]:

Problem#2

December 9, 2019

[1]:
```
from gurobipy import *


# create a model
m = Model()

# create variables
f11 = m.addVar(vtype=GRB.BINARY, name="f11", lb=0)
f21 = m.addVar(vtype=GRB.BINARY, name="f21", lb=0)
f31 = m.addVar(vtype=GRB.BINARY, name="f31", lb=0)
f41 = m.addVar(vtype=GRB.BINARY, name="f41", lb=0)
f51 = m.addVar(vtype=GRB.BINARY, name="f51", lb=0)
f61 = m.addVar(vtype=GRB.BINARY, name="f61", lb=0)
f71 = m.addVar(vtype=GRB.BINARY, name="f71", lb=0)
f12 = m.addVar(vtype=GRB.BINARY, name="f12", lb=0)
f22 = m.addVar(vtype=GRB.BINARY, name="f22", lb=0)
f32 = m.addVar(vtype=GRB.BINARY, name="f32", lb=0)
f42 = m.addVar(vtype=GRB.BINARY, name="f42", lb=0)
f52 = m.addVar(vtype=GRB.BINARY, name="f52", lb=0)
f62 = m.addVar(vtype=GRB.BINARY, name="f62", lb=0)
f72 = m.addVar(vtype=GRB.BINARY, name="f72", lb=0)
f13 = m.addVar(vtype=GRB.BINARY, name="f13", lb=0)
f23 = m.addVar(vtype=GRB.BINARY, name="f23", lb=0)
f33 = m.addVar(vtype=GRB.BINARY, name="f33", lb=0)
f43 = m.addVar(vtype=GRB.BINARY, name="f43", lb=0)
f53 = m.addVar(vtype=GRB.BINARY, name="f53", lb=0)
f63 = m.addVar(vtype=GRB.BINARY, name="f63", lb=0)
f73 = m.addVar(vtype=GRB.BINARY, name="f73", lb=0)
f14 = m.addVar(vtype=GRB.BINARY, name="f14", lb=0)
f24 = m.addVar(vtype=GRB.BINARY, name="f24", lb=0)
f34 = m.addVar(vtype=GRB.BINARY, name="f34", lb=0)
f44 = m.addVar(vtype=GRB.BINARY, name="f44", lb=0)
f54 = m.addVar(vtype=GRB.BINARY, name="f54", lb=0)
f64 = m.addVar(vtype=GRB.BINARY, name="f64", lb=0)
f74 = m.addVar(vtype=GRB.BINARY, name="f74", lb=0)
f15 = m.addVar(vtype=GRB.BINARY, name="f15", lb=0)
f25 = m.addVar(vtype=GRB.BINARY, name="f25", lb=0)
```

```
f35 = m.addVar(vtype=GRB.BINARY, name="f35", lb=0)
f45 = m.addVar(vtype=GRB.BINARY, name="f45", lb=0)
f55 = m.addVar(vtype=GRB.BINARY, name="f55", lb=0)
f65 = m.addVar(vtype=GRB.BINARY, name="f65", lb=0)
f75 = m.addVar(vtype=GRB.BINARY, name="f75", lb=0)
f16 = m.addVar(vtype=GRB.BINARY, name="f16", lb=0)
f26 = m.addVar(vtype=GRB.BINARY, name="f26", lb=0)
f36 = m.addVar(vtype=GRB.BINARY, name="f36", lb=0)
f46 = m.addVar(vtype=GRB.BINARY, name="f46", lb=0)
f56 = m.addVar(vtype=GRB.BINARY, name="f56", lb=0)
f66 = m.addVar(vtype=GRB.BINARY, name="f66", lb=0)
f76 = m.addVar(vtype=GRB.BINARY, name="f76", lb=0)
f17 = m.addVar(vtype=GRB.BINARY, name="f17", lb=0)
f27 = m.addVar(vtype=GRB.BINARY, name="f27", lb=0)
f37 = m.addVar(vtype=GRB.BINARY, name="f37", lb=0)
f47 = m.addVar(vtype=GRB.BINARY, name="f47", lb=0)
f57 = m.addVar(vtype=GRB.BINARY, name="f57", lb=0)
f67 = m.addVar(vtype=GRB.BINARY, name="f67", lb=0)
f77 = m.addVar(vtype=GRB.BINARY, name="f77", lb=0)
x1 = m.addVar(vtype=GRB.BINARY, name="x1", lb=0)
x2 = m.addVar(vtype=GRB.BINARY, name="x2", lb=0)
x3 = m.addVar(vtype=GRB.BINARY, name="x3", lb=0)
x4 = m.addVar(vtype=GRB.BINARY, name="x4", lb=0)
x5 = m.addVar(vtype=GRB.BINARY, name="x5", lb=0)
x6 = m.addVar(vtype=GRB.BINARY, name="x6", lb=0)
x7 = m.addVar(vtype=GRB.BINARY, name="x7", lb=0)

# integrate new variables
m.update()

# set objective
m.setObjective(
      300000*(f11 + f12 + f13 + f14 + f15 + f16 + f17)
    + 300000*(f21 + f22 + f23 + f24 + f25 + f26 + f27)
    + 450000*(f31 + f32 + f33 + f34 + f35 + f36 + f37)
    + 100000*(f41 + f42 + f43 + f44 + f45 + f46 + f47)
    + 100000*(f51 + f52 + f53 + f54 + f55 + f56 + f57)
    + 400000*(f61 + f62 + f63 + f64 + f65 + f66 + f67)
    + 300000*(f71 + f72 + f73 + f74 + f75 + f76 + f77)
    - 500000*(x1 + x2 + x3 + x4 + x5 + x6 + x7),
    GRB.MAXIMIZE
)

# add constraints
# one node should only have at most one frequency
m.addConstr(f11 + f12 + f13 + f14 + f15 + f16 + f17 <= 1)
m.addConstr(f21 + f22 + f23 + f24 + f25 + f26 + f27 <= 1)
```

```python
m.addConstr(f31 + f32 + f33 + f34 + f35 + f36 + f37 <= 1)
m.addConstr(f41 + f42 + f43 + f44 + f45 + f46 + f47 <= 1)
m.addConstr(f51 + f52 + f53 + f54 + f55 + f56 + f57 <= 1)
m.addConstr(f61 + f62 + f63 + f64 + f65 + f66 + f67 <= 1)
m.addConstr(f71 + f72 + f73 + f74 + f75 + f76 + f77 <= 1)

# fvi node use color i only if xi = 1
m.addConstr(f11 <= x1)
m.addConstr(f21 <= x1)
m.addConstr(f31 <= x1)
m.addConstr(f41 <= x1)
m.addConstr(f51 <= x1)
m.addConstr(f61 <= x1)
m.addConstr(f71 <= x1)
m.addConstr(f12 <= x2)
m.addConstr(f22 <= x2)
m.addConstr(f32 <= x2)
m.addConstr(f42 <= x2)
m.addConstr(f52 <= x2)
m.addConstr(f62 <= x2)
m.addConstr(f72 <= x2)
m.addConstr(f13 <= x3)
m.addConstr(f23 <= x3)
m.addConstr(f33 <= x3)
m.addConstr(f43 <= x3)
m.addConstr(f53 <= x3)
m.addConstr(f63 <= x3)
m.addConstr(f73 <= x3)
m.addConstr(f14 <= x4)
m.addConstr(f24 <= x4)
m.addConstr(f34 <= x4)
m.addConstr(f44 <= x4)
m.addConstr(f54 <= x4)
m.addConstr(f64 <= x4)
m.addConstr(f74 <= x4)
m.addConstr(f15 <= x5)
m.addConstr(f25 <= x5)
m.addConstr(f35 <= x5)
m.addConstr(f45 <= x5)
m.addConstr(f55 <= x5)
m.addConstr(f65 <= x5)
m.addConstr(f75 <= x5)
m.addConstr(f16 <= x6)
m.addConstr(f26 <= x6)
m.addConstr(f36 <= x6)
m.addConstr(f46 <= x6)
m.addConstr(f56 <= x6)
```

```
m.addConstr(f66 <= x6)
m.addConstr(f76 <= x6)
m.addConstr(f17 <= x7)
m.addConstr(f27 <= x7)
m.addConstr(f37 <= x7)
m.addConstr(f47 <= x7)
m.addConstr(f57 <= x7)
m.addConstr(f67 <= x7)
m.addConstr(f77 <= x7)

# adjacent nodes should have different frequency
# edge(1,2)
m.addConstr(f11 + f21 <= 1)
m.addConstr(f12 + f22 <= 1)
m.addConstr(f13 + f23 <= 1)
m.addConstr(f14 + f24 <= 1)
m.addConstr(f15 + f25 <= 1)
m.addConstr(f16 + f26 <= 1)
m.addConstr(f17 + f27 <= 1)
# edge(1,3)
m.addConstr(f11 + f31 <= 1)
m.addConstr(f12 + f32 <= 1)
m.addConstr(f13 + f33 <= 1)
m.addConstr(f14 + f34 <= 1)
m.addConstr(f15 + f35 <= 1)
m.addConstr(f16 + f36 <= 1)
m.addConstr(f17 + f37 <= 1)
# edge(1,6)
m.addConstr(f11 + f61 <= 1)
m.addConstr(f12 + f62 <= 1)
m.addConstr(f13 + f63 <= 1)
m.addConstr(f14 + f64 <= 1)
m.addConstr(f15 + f65 <= 1)
m.addConstr(f16 + f66 <= 1)
m.addConstr(f17 + f67 <= 1)
# edge(2,4)
m.addConstr(f21 + f41 <= 1)
m.addConstr(f22 + f42 <= 1)
m.addConstr(f23 + f43 <= 1)
m.addConstr(f24 + f44 <= 1)
m.addConstr(f25 + f45 <= 1)
m.addConstr(f26 + f46 <= 1)
m.addConstr(f27 + f47 <= 1)
# edge(3,5)
m.addConstr(f31 + f51 <= 1)
m.addConstr(f32 + f52 <= 1)
m.addConstr(f33 + f53 <= 1)
```

```
m.addConstr(f34 + f54 <= 1)
m.addConstr(f35 + f55 <= 1)
m.addConstr(f36 + f56 <= 1)
m.addConstr(f37 + f57 <= 1)
# edge(3,4)
m.addConstr(f31 + f41 <= 1)
m.addConstr(f32 + f42 <= 1)
m.addConstr(f33 + f43 <= 1)
m.addConstr(f34 + f44 <= 1)
m.addConstr(f35 + f45 <= 1)
m.addConstr(f36 + f46 <= 1)
m.addConstr(f37 + f47 <= 1)
# edge(3,7)
m.addConstr(f31 + f71 <= 1)
m.addConstr(f32 + f72 <= 1)
m.addConstr(f33 + f73 <= 1)
m.addConstr(f34 + f74 <= 1)
m.addConstr(f35 + f75 <= 1)
m.addConstr(f36 + f76 <= 1)
m.addConstr(f37 + f77 <= 1)
# edge(4,7)
m.addConstr(f41 + f71 <= 1)
m.addConstr(f42 + f72 <= 1)
m.addConstr(f43 + f73 <= 1)
m.addConstr(f44 + f74 <= 1)
m.addConstr(f45 + f75 <= 1)
m.addConstr(f46 + f76 <= 1)
m.addConstr(f47 + f77 <= 1)
# edge(5,7)
m.addConstr(f51 + f71 <= 1)
m.addConstr(f52 + f72 <= 1)
m.addConstr(f53 + f73 <= 1)
m.addConstr(f54 + f74 <= 1)
m.addConstr(f55 + f75 <= 1)
m.addConstr(f56 + f76 <= 1)
m.addConstr(f57 + f77 <= 1)
# edge(6,7)
m.addConstr(f61 + f71 <= 1)
m.addConstr(f62 + f72 <= 1)
m.addConstr(f63 + f73 <= 1)
m.addConstr(f64 + f74 <= 1)
m.addConstr(f65 + f75 <= 1)
m.addConstr(f66 + f76 <= 1)
m.addConstr(f67 + f77 <= 1)


# optimize
```

```python
m.optimize()
print("Model status: ", m.status)

# print out decision variables
for v in m.getVars():
    print(v.varName, v.x, "\n")

print("-"*15)
print("Obj Value: ", m.objVal)
```

```
Academic license - for non-commercial use only
Optimize a model with 126 rows, 56 columns and 287 nonzeros
Variable types: 0 continuous, 56 integer (56 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [1e+05, 5e+05]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+00]
Found heuristic solution: objective -0.0000000
Found heuristic solution: objective 450000.00000
Presolve removed 70 rows and 0 columns
Presolve time: 0.00s
Presolved: 56 rows, 56 columns, 210 nonzeros
Variable types: 0 continuous, 56 integer (56 binary)

Root relaxation: objective 7.500000e+05, 48 iterations, 0.00 seconds

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

*    0     0               0    750000.00000 750000.000  0.00%     -    0s

Explored 0 nodes (48 simplex iterations) in 0.12 seconds
Thread count was 4 (of 4 available processors)

Solution count 3: 750000 450000 -0

Optimal solution found (tolerance 1.00e-04)
Best objective 7.500000000000e+05, best bound 7.500000000000e+05, gap 0.0000%
Model status:  2
f11 1.0

f21 -0.0

f31 -0.0

f41 0.0
```

f51 0.0

f61 0.0

f71 1.0

f12 0.0

f22 0.0

f32 -0.0

f42 0.0

f52 0.0

f62 0.0

f72 0.0

f13 0.0

f23 -0.0

f33 0.0

f43 -0.0

f53 0.0

f63 -0.0

f73 -0.0

f14 0.0

f24 -0.0

f34 0.0

f44 0.0

f54 -0.0

f64 -0.0

f74 0.0

f15 0.0

f25 -0.0

f35 -0.0

f45 -0.0

f55 0.0

f65 -0.0

f75 0.0

f16 0.0

f26 1.0

f36 1.0

f46 0.0

f56 -0.0

f66 1.0

f76 -0.0

f17 0.0

f27 0.0

f37 0.0

f47 0.0

f57 0.0

f67 0.0

f77 0.0

x1 1.0

x2 0.0

x3 -0.0

```
x4 0.0

x5 -0.0

x6 1.0

x7 0.0

---------------
Obj Value:  750000.0
```

[ ]:

Problem#3

December 9, 2019

[1]:
```python
from gurobipy import *


# create a model
m = Model()

# create variables
xa = m.addVar(vtype=GRB.CONTINUOUS, name="xa", lb=0)
xh = m.addVar(vtype=GRB.CONTINUOUS, name="xh", lb=0)
sm = m.addVar(vtype=GRB.BINARY, name="sm", lb=0, ub=1)
t = m.addVar(vtype=GRB.BINARY, name="t", lb=0, ub=1)
c = m.addVar(vtype=GRB.BINARY, name="c", lb=0, ub=1)


# integrate new variables
m.update()

# set objective
m.setObjective(
    (48000-40000)*xa + (46000-40000)*xh - (1200000*sm + 2800000*t),
    GRB.MAXIMIZE
)

# add constraints

m.addConstr(xa+xh <= 10000)
m.addConstr(t <= c)
m.addConstr(sm <= 1-c)
m.addConstr(t+sm == 1)
m.addConstr(xh >= 3*xa*sm) # Since the number of sailboat marina could be either␣
 ↪0 or 1

# optimize
m.optimize()
print("Model status: ", m.status)

# print out decision variables
```

```python
for v in m.getVars():
    print(v.varName, v.x, "\n")

print("-"*15)
print("Obj Value: ", m.objVal)
```

```
Academic license - for non-commercial use only
Optimize a model with 4 rows, 5 columns and 8 nonzeros
Model has 1 quadratic constraint
Variable types: 2 continuous, 3 integer (3 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  QMatrix range    [3e+00, 3e+00]
  QLMatrix range   [1e+00, 1e+00]
  Objective range  [6e+03, 3e+06]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+04]
Presolve removed 3 rows and 2 columns
Presolve time: 0.00s
Presolved: 3 rows, 4 columns, 7 nonzeros
Variable types: 3 continuous, 1 integer (1 binary)

Root relaxation: objective 7.720000e+07, 2 iterations, 0.00 seconds

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

*    0     0               0    7.720000e+07 7.7200e+07 -0.00%     -    0s

Explored 0 nodes (2 simplex iterations) in 0.07 seconds
Thread count was 4 (of 4 available processors)

Solution count 1: 7.72e+07

Optimal solution found (tolerance 1.00e-04)
Best objective 7.720000000000e+07, best bound 7.720000000000e+07, gap 0.0000%
Model status:  2
xa 10000.0

xh 0.0

sm 0.0

t 1.0

c 1.0
```

```
---------------
Obj Value:  77200000.0
```

[ ]:

```

```

Problem#4

December 9, 2019

[1]:
```
import pandas as pd
import numpy as np
```

[2]:
```
filename = 'Assignment6_files/coins.dat'
infile = open(filename, 'r')
```

[3]:
```
with open(filename, 'r') as infile:
    for line in infile:
        print(line)
```

```
15

40

1 3 5 8 10 11 14 16 20 23 25 28 30 34 38
```

Therefore, there are 15 different types of coins, with values 1,3,5,8,10,11,14,16,20,23,25,28,30,34,38. And, the wallet can fit at most 40 coins. The question becomes: Use Dynamic Programming to find the smallest value (in whole dollars) that cannot be created and fit into the wallet.

[4]:
```
m = 15
k = 40
values = np.array([1,3,5,8,10,11,14,16,20,23,25,28,30,34,38])
```

[5]:
```
def max_create(num_coins,capacity,coin_values):
    columns = capacity*max(coin_values)
    table = np.zeros((capacity+1,columns))

    # f(m,n)=0 for m=0 (when we use 0 coins)
    for n in range(columns):
        table[0,n] = 0

    # f(m,n)=1 for m=1, while the value in the coin value sets
    for k in range(num_coins):
        table[1,coin_values[k]]=1

    for n in range(columns):
        for m in range(1,capacity+1):
```

```
            table[m,n] = max(table[m-1,n],table[m,n])
            for value in range(num_coins):
                try:
                    v = table[m-1, n-coin_values[value]]
                except:
                    v = 0
                table[m,n] = max(v,table[m,n])
    return table
```

[6]:
```
def find_value(dataframe):
    last_row = df.iloc[-1]
    zeros = []
    count = 0
    for i in last_row:
        if i == 0:
            zeros.append(count)
        count += 1
    print('The smallest value that cannot be created and fit into the wallet is:
 ↪', zeros[1])
```

[7]:
```
result = max_create(m,k,values)
df = pd.DataFrame(result, columns = np.arange(k*max(values)) )

find_value(df)
```

```
The smallest value that cannot be created and fit into the wallet is: 1509
```

[ ]:
```
```