MSOR Yi Ping Tseng

prob 1

x_{ij} = flow run through edge(i,j) y_i : whether run through edge (5, i).

obj : $\max\ 190x_{11} + 200x_{12} + 100x_{21} + 200x_{23} + 400x_{33}$
$+ 150x_{34} + 570x_{43} + 70x_{54} - 1000y_1 - 2000y_2 - 700y_3 - 2000y_4$
$- 1500y_5 - 102(x_{11} + x_{21}) - 88 \cdot x_{12} - 107(x_{23} + x_{33} + x_{43})$
$- 234(x_{34} + x_{54})$

s.t. $\underline{x_{ij} \le y_i \times 30}$

$\Downarrow$

$x_{11} \le 30y_1$
$x_{12} \le 30y_1$
$x_{21} \le 30y_2$
$x_{23} \le 30y_2$
$x_{33} \le 30y_3$
$x_{34} \le 30y_3$
$x_{43} \le 30y_4$
$x_{54} \le 30y_5$

$\sum_{i=1}^{5}\sum_{j=1}^{4} x_{ij} \le 100$

run by gurobi : $x_{11} = 10$ $x_{12} = 30$ $x_{33} = 30$ $x_{43} = 30$ $x_{21} = x_{23} = x_{34} = x_{54} = 0$

$y_1 = y_3 = y_4 = 1$ $y_2 = y_5 = 0$

obj value : 20220 #

Problem1

December 10, 2019

0.1 Problem 1

name: Yi Ping Tseng

uni: yt2690

github: https://github.com/r50206v/Optimization-Homework/tree/master/homework6

[ ]:
```
from gurobipy import *



# create a model
m = Model()

# create variables
x11 = m.addVar(vtype=GRB.CONTINUOUS, name="x11", lb=0)
x12 = m.addVar(vtype=GRB.CONTINUOUS, name="x12", lb=0)
x21 = m.addVar(vtype=GRB.CONTINUOUS, name="x21", lb=0)
x23 = m.addVar(vtype=GRB.CONTINUOUS, name="x23", lb=0)
x33 = m.addVar(vtype=GRB.CONTINUOUS, name="x33", lb=0)
x34 = m.addVar(vtype=GRB.CONTINUOUS, name="x34", lb=0)
x43 = m.addVar(vtype=GRB.CONTINUOUS, name="x43", lb=0)
x54 = m.addVar(vtype=GRB.CONTINUOUS, name="x54", lb=0)
y1 = m.addVar(vtype=GRB.BINARY, name="y1", lb=0, ub=1)
y2 = m.addVar(vtype=GRB.BINARY, name="y2", lb=0, ub=1)
y3 = m.addVar(vtype=GRB.BINARY, name="y3", lb=0, ub=1)
y4 = m.addVar(vtype=GRB.BINARY, name="y4", lb=0, ub=1)
y5 = m.addVar(vtype=GRB.BINARY, name="y5", lb=0, ub=1)


# integrate new variables
m.update()

# set objective
m.setObjective(
    190*x11 + 200*x12 + 100*x21 + 300*x23 + 400*x33 + 150*x34 + 570*x43 +␣
 ↪70*x54 \
    - (1000*y1 + 3000*y2 + 700*y3 + 2000*y4 + 1500*y5 \
    + 102*(x11 + x21) + 88*(x12) + 157*(x23 + x33 + x43) + 234*(x34 + x54)),
    GRB.MAXIMIZE
)

# add constraints
m.addConstr(x11 <= 30*y1)
m.addConstr(x12 <= 30*y1)
m.addConstr(x21 <= 30*y2)
m.addConstr(x23 <= 30*y2)
m.addConstr(x33 <= 30*y3)
m.addConstr(x34 <= 30*y3)
m.addConstr(x43 <= 30*y4)
m.addConstr(x54 <= 30*y5)
m.addConstr(x11 + x12 + x21 + x23 + x33 + x34 + x43 + x54 <= 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)


'''
x11 10.0
x12 30.0
x21 0.0
x23 0.0
x33 30.0
x34 0.0
x43 30.0
x54 0.0
y1 1.0
y2 0.0
y3 1.0
y4 1.0
y5 0.0
---------------
Obj Value:  20220.0
'''
```

prob 2

$y_i = \begin{cases} 1, & \text{if we use color } i \\ 0, & \text{o.w.} \end{cases}$

$x_{ij} = \begin{cases} 1, & \text{if we use color } j \text{ on node } i \\ 0, & \text{o.w.} \end{cases}$

p_v = the profit we earn to broadcasting in city v.

There are at most 7 colors, since there are only 7 cities.

$$\max \sum_{i=1}^{7} \boxed{\sum_{j=1}^{7}} p_i \cdot x_{ij} - 500000 \underline{\sum_{i=1}^{7} y_i}$$

s.t. $x_{ij} \le y_j \quad \forall i, j$ we color j only if $y_j = 1$

$\sum_{j=1}^{7} x_{ij} \le 1 \quad \forall i$, each node should only have one color

$x_{ij} + x_{kj} \le 1 \quad \forall j \quad (i,k) \in E$, adjacent nodes must have different colors.

run by gurobi,

$x_{11} = x_{71} = 1$

$x_{24} = x_{34} = x_{64} = 1$

$y_1 = y_4 = 1$

other variables are all 0

obj value: 750000 #

Problem2

December 10, 2019

0.1 Problem 2

name: Yi Ping Tseng

uni: yt2690

github: https://github.com/r50206v/Optimization-Homework/tree/master/homework6

[ ]:
```
from gurobipy import *



# create a model
m = Model()

# create variables
x11 = m.addVar(vtype=GRB.BINARY, name="x11", lb=0)
x21 = m.addVar(vtype=GRB.BINARY, name="x21", lb=0)
x31 = m.addVar(vtype=GRB.BINARY, name="x31", lb=0)
x41 = m.addVar(vtype=GRB.BINARY, name="x41", lb=0)
x51 = m.addVar(vtype=GRB.BINARY, name="x51", lb=0)
x61 = m.addVar(vtype=GRB.BINARY, name="x61", lb=0)
x71 = m.addVar(vtype=GRB.BINARY, name="x71", lb=0)
x12 = m.addVar(vtype=GRB.BINARY, name="x12", lb=0)
x22 = m.addVar(vtype=GRB.BINARY, name="x22", lb=0)
x32 = m.addVar(vtype=GRB.BINARY, name="x32", lb=0)
x42 = m.addVar(vtype=GRB.BINARY, name="x42", lb=0)
x52 = m.addVar(vtype=GRB.BINARY, name="x52", lb=0)
x62 = m.addVar(vtype=GRB.BINARY, name="x62", lb=0)
x72 = m.addVar(vtype=GRB.BINARY, name="x72", lb=0)
x13 = m.addVar(vtype=GRB.BINARY, name="x13", lb=0)
x23 = m.addVar(vtype=GRB.BINARY, name="x23", lb=0)
x33 = m.addVar(vtype=GRB.BINARY, name="x33", lb=0)
x43 = m.addVar(vtype=GRB.BINARY, name="x43", lb=0)
x53 = m.addVar(vtype=GRB.BINARY, name="x53", lb=0)
x63 = m.addVar(vtype=GRB.BINARY, name="x63", lb=0)
x73 = m.addVar(vtype=GRB.BINARY, name="x73", lb=0)
x14 = m.addVar(vtype=GRB.BINARY, name="x14", lb=0)
```

```
x24 = m.addVar(vtype=GRB.BINARY, name="x24", lb=0)
x34 = m.addVar(vtype=GRB.BINARY, name="x34", lb=0)
x44 = m.addVar(vtype=GRB.BINARY, name="x44", lb=0)
x54 = m.addVar(vtype=GRB.BINARY, name="x54", lb=0)
x64 = m.addVar(vtype=GRB.BINARY, name="x64", lb=0)
x74 = m.addVar(vtype=GRB.BINARY, name="x74", lb=0)
x15 = m.addVar(vtype=GRB.BINARY, name="x15", lb=0)
x25 = m.addVar(vtype=GRB.BINARY, name="x25", lb=0)
x35 = m.addVar(vtype=GRB.BINARY, name="x35", lb=0)
x45 = m.addVar(vtype=GRB.BINARY, name="x45", lb=0)
x55 = m.addVar(vtype=GRB.BINARY, name="x55", lb=0)
x65 = m.addVar(vtype=GRB.BINARY, name="x65", lb=0)
x75 = m.addVar(vtype=GRB.BINARY, name="x75", lb=0)
x16 = m.addVar(vtype=GRB.BINARY, name="x16", lb=0)
x26 = m.addVar(vtype=GRB.BINARY, name="x26", lb=0)
x36 = m.addVar(vtype=GRB.BINARY, name="x36", lb=0)
x46 = m.addVar(vtype=GRB.BINARY, name="x46", lb=0)
x56 = m.addVar(vtype=GRB.BINARY, name="x56", lb=0)
x66 = m.addVar(vtype=GRB.BINARY, name="x66", lb=0)
x76 = m.addVar(vtype=GRB.BINARY, name="x76", lb=0)
x17 = m.addVar(vtype=GRB.BINARY, name="x17", lb=0)
x27 = m.addVar(vtype=GRB.BINARY, name="x27", lb=0)
x37 = m.addVar(vtype=GRB.BINARY, name="x37", lb=0)
x47 = m.addVar(vtype=GRB.BINARY, name="x47", lb=0)
x57 = m.addVar(vtype=GRB.BINARY, name="x57", lb=0)
x67 = m.addVar(vtype=GRB.BINARY, name="x67", lb=0)
x77 = m.addVar(vtype=GRB.BINARY, name="x77", lb=0)
y1 = m.addVar(vtype=GRB.BINARY, name="y1", lb=0)
y2 = m.addVar(vtype=GRB.BINARY, name="y2", lb=0)
y3 = m.addVar(vtype=GRB.BINARY, name="y3", lb=0)
y4 = m.addVar(vtype=GRB.BINARY, name="y4", lb=0)
y5 = m.addVar(vtype=GRB.BINARY, name="y5", lb=0)
y6 = m.addVar(vtype=GRB.BINARY, name="y6", lb=0)
y7 = m.addVar(vtype=GRB.BINARY, name="y7", lb=0)
p1 = 300000
p2 = 300000
p3 = 450000
p4 = 100000
p5 = 100000
p6 = 400000
p7 = 300000

# integrate new variables
m.update()

# set objective
m.setObjective(
    p1*x11 + p1*x12 + p1*x13 + p1*x14 + p1*x15 + p1*x16 + p1*x17 \
    + p2*x21 + p2*x22 + p2*x23 + p2*x24 + p2*x25 + p2*x26 + p2*x27 \
    + p3*x31 + p3*x32 + p3*x33 + p3*x34 + p3*x35 + p3*x36 + p3*x37 \
    + p4*x41 + p4*x42 + p4*x43 + p4*x44 + p4*x45 + p4*x46 + p4*x47 \
    + p5*x51 + p5*x52 + p5*x53 + p5*x54 + p5*x55 + p5*x56 + p5*x57 \
    + p6*x61 + p6*x62 + p6*x63 + p6*x64 + p6*x65 + p6*x66 + p6*x67 \
    + p7*x71 + p7*x72 + p7*x73 + p7*x74 + p7*x75 + p7*x76 + p7*x77 \
    - 500000*(y1 + y2 + y3 + y4 + y5 + y6 + y7),
    GRB.MAXIMIZE
)

# add constraints
# xij node use color j only if yj = 1
m.addConstr(x11 <= y1)
m.addConstr(x21 <= y1)
m.addConstr(x31 <= y1)
m.addConstr(x41 <= y1)
m.addConstr(x51 <= y1)
m.addConstr(x61 <= y1)
m.addConstr(x71 <= y1)
m.addConstr(x12 <= y2)
m.addConstr(x22 <= y2)
m.addConstr(x32 <= y2)
m.addConstr(x42 <= y2)
m.addConstr(x52 <= y2)
m.addConstr(x62 <= y2)
m.addConstr(x72 <= y2)
m.addConstr(x13 <= y3)
m.addConstr(x23 <= y3)
m.addConstr(x33 <= y3)
m.addConstr(x43 <= y3)
m.addConstr(x53 <= y3)
m.addConstr(x63 <= y3)
m.addConstr(x73 <= y3)
m.addConstr(x14 <= y4)
m.addConstr(x24 <= y4)
m.addConstr(x34 <= y4)
m.addConstr(x44 <= y4)
m.addConstr(x54 <= y4)
m.addConstr(x64 <= y4)
m.addConstr(x74 <= y4)
m.addConstr(x15 <= y5)
m.addConstr(x25 <= y5)
m.addConstr(x35 <= y5)
m.addConstr(x45 <= y5)
m.addConstr(x55 <= y5)
m.addConstr(x65 <= y5)
```

```
m.addConstr(x75 <= y5)
m.addConstr(x16 <= y6)
m.addConstr(x26 <= y6)
m.addConstr(x36 <= y6)
m.addConstr(x46 <= y6)
m.addConstr(x56 <= y6)
m.addConstr(x66 <= y6)
m.addConstr(x76 <= y6)
m.addConstr(x17 <= y7)
m.addConstr(x27 <= y7)
m.addConstr(x37 <= y7)
m.addConstr(x47 <= y7)
m.addConstr(x57 <= y7)
m.addConstr(x67 <= y7)
m.addConstr(x77 <= y7)
# one node should only have one frequency
m.addConstr(x11 + x12 + x13 + x14 + x15 + x16 + x17 <= 1)
m.addConstr(x21 + x22 + x23 + x24 + x25 + x26 + x27 <= 1)
m.addConstr(x31 + x32 + x33 + x34 + x35 + x36 + x37 <= 1)
m.addConstr(x41 + x42 + x43 + x44 + x45 + x46 + x47 <= 1)
m.addConstr(x51 + x52 + x53 + x54 + x55 + x56 + x57 <= 1)
m.addConstr(x61 + x62 + x63 + x64 + x65 + x66 + x67 <= 1)
m.addConstr(x71 + x72 + x73 + x74 + x75 + x76 + x77 <= 1)
# adjacent nodes should have different frequency
m.addConstr(x11 + x21 <= 1)
m.addConstr(x12 + x22 <= 1)
m.addConstr(x13 + x23 <= 1)
m.addConstr(x14 + x24 <= 1)
m.addConstr(x15 + x25 <= 1)
m.addConstr(x16 + x26 <= 1)
m.addConstr(x17 + x27 <= 1)
m.addConstr(x11 + x31 <= 1)
m.addConstr(x12 + x32 <= 1)
m.addConstr(x13 + x33 <= 1)
m.addConstr(x14 + x34 <= 1)
m.addConstr(x15 + x35 <= 1)
m.addConstr(x16 + x36 <= 1)
m.addConstr(x17 + x37 <= 1)
m.addConstr(x11 + x61 <= 1)
m.addConstr(x12 + x62 <= 1)
m.addConstr(x13 + x63 <= 1)
m.addConstr(x14 + x64 <= 1)
m.addConstr(x15 + x65 <= 1)
m.addConstr(x16 + x66 <= 1)
m.addConstr(x17 + x67 <= 1)
m.addConstr(x21 + x41 <= 1)
m.addConstr(x22 + x42 <= 1)
```

```
m.addConstr(x23 + x43 <= 1)
m.addConstr(x24 + x44 <= 1)
m.addConstr(x25 + x45 <= 1)
m.addConstr(x26 + x46 <= 1)
m.addConstr(x27 + x47 <= 1)
m.addConstr(x31 + x41 <= 1)
m.addConstr(x32 + x42 <= 1)
m.addConstr(x33 + x43 <= 1)
m.addConstr(x34 + x44 <= 1)
m.addConstr(x35 + x45 <= 1)
m.addConstr(x36 + x46 <= 1)
m.addConstr(x37 + x47 <= 1)
m.addConstr(x31 + x51 <= 1)
m.addConstr(x32 + x52 <= 1)
m.addConstr(x33 + x53 <= 1)
m.addConstr(x34 + x54 <= 1)
m.addConstr(x35 + x55 <= 1)
m.addConstr(x36 + x56 <= 1)
m.addConstr(x37 + x57 <= 1)
m.addConstr(x31 + x71 <= 1)
m.addConstr(x32 + x72 <= 1)
m.addConstr(x33 + x73 <= 1)
m.addConstr(x34 + x74 <= 1)
m.addConstr(x35 + x75 <= 1)
m.addConstr(x36 + x76 <= 1)
m.addConstr(x37 + x77 <= 1)
m.addConstr(x41 + x71 <= 1)
m.addConstr(x42 + x72 <= 1)
m.addConstr(x43 + x73 <= 1)
m.addConstr(x44 + x74 <= 1)
m.addConstr(x45 + x75 <= 1)
m.addConstr(x46 + x76 <= 1)
m.addConstr(x47 + x77 <= 1)
m.addConstr(x51 + x71 <= 1)
m.addConstr(x52 + x72 <= 1)
m.addConstr(x53 + x73 <= 1)
m.addConstr(x54 + x74 <= 1)
m.addConstr(x55 + x75 <= 1)
m.addConstr(x56 + x76 <= 1)
m.addConstr(x57 + x77 <= 1)
m.addConstr(x61 + x71 <= 1)
m.addConstr(x62 + x72 <= 1)
m.addConstr(x63 + x73 <= 1)
m.addConstr(x64 + x74 <= 1)
m.addConstr(x65 + x75 <= 1)
m.addConstr(x66 + x76 <= 1)
m.addConstr(x67 + x77 <= 1)
```

```
# 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)


'''
x11 1.0
x21 0.0
x31 0.0
x41 0.0
x51 -0.0
x61 -0.0
x71 1.0
x12 0.0
x22 -0.0
x32 0.0
x42 0.0
x52 -0.0
x62 -0.0
x72 0.0
x13 0.0
x23 0.0
x33 0.0
x43 0.0
x53 -0.0
x63 0.0
x73 -0.0
x14 0.0
x24 1.0
x34 1.0
x44 0.0
x54 -0.0
x64 1.0
x74 -0.0
x15 0.0
x25 0.0
x35 0.0
x45 0.0
```

```
x55 -0.0
x65 -0.0
x75 -0.0
x16 0.0
x26 -0.0
x36 0.0
x46 0.0
x56 -0.0
x66 0.0
x76 0.0
x17 0.0
x27 -0.0
x37 0.0
x47 0.0
x57 -0.0
x67 0.0
x77 0.0
y1 1.0
y2 0.0
y3 0.0
y4 1.0
y5 0.0
y6 0.0
y7 0.0
---------------
Obj Value:  750000.0
'''
```

prob 3.

X_a : # of apartment

X_h : # of homes.

$r_m = \begin{cases} 1 & \text{, build marina} \\ 0 & \text{, ow} \end{cases}$

$r_s = \begin{cases} 1 & \text{, build swimming} \\ 0 & \text{, ow.} \end{cases}$

obj: max $X_a(48000 - 40000) + X_h(46000 - 40000)$

$- r_m \times 1200000 \quad - r_s \times 2800000.$

s.t. $X_a + X_h \leq 10000$

$(X_h - 3X_a) \geq -9999999(1 - r_m)$ if build r_m, X_h should be triple of X_a.

$r_m + r_s = 1$

run by gurobi:

$X_a = 10000$
$X_h = 0$
$r_m = 0$
$r_s = 1$

obj value = 77200000 #

Problem3

December 10, 2019

0.1 Problem 3

name: Yi Ping Tseng

uni: yt2690

github: https://github.com/r50206v/Optimization-Homework/tree/master/homework6

[ ]:
```
from gurobipy import *



# create a model
m = Model()

# create variables
xa = m.addVar(vtype=GRB.INTEGER, name="xa", lb=0)
xh = m.addVar(vtype=GRB.INTEGER, name="xh", lb=0)
rm = m.addVar(vtype=GRB.BINARY, name="rm", lb=0)
rs = m.addVar(vtype=GRB.BINARY, name="rs", lb=0)



# integrate new variables
m.update()

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

# add constraints
m.addConstr(xa + xh <= 10000)
m.addConstr((xh - 3*xa) >= -999999999999*(1 - rm))
m.addConstr(rm + rs == 1)

# optimize
m.optimize()
```

```python
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)

'''
xa 10000.0
xh 0.0
rm 0.0
rs 1.0
---------------
Obj Value:  77200000.0
'''
```

prob 4.

$$f(j) = \begin{cases} \min\left(f(j - \text{coin}) + 1,\ f(j) \right). & ,\ j \neq \text{coin} \\ 1 & ,\ \text{if } j = \text{coin} \end{cases}$$

the maximum value should always be max(coins) × wallet size

value	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
# of coins	1	1	2	2	1	2	2	3	3	2	3	3	3	3	3	4	4	5	5	4	5	5	X	X	5
			$\downarrow$ f(2)+f(1)	$\downarrow$ f(2)+f(2)		$\downarrow$ f(5)+f(1)	$\downarrow$ f(5)+f(2)	$\downarrow$ f(5)+f(3)	$\downarrow$ f(5)+f(4)	$\downarrow$ f(5)+f(5)															

the smallest value that cannot be created is $\bigcirc\!\!\!\!\!\!\;23$ #

Problem4

December 10, 2019

0.1 Problem 4

name: Yi Ping Tseng

uni: yt2690

github: https://github.com/r50206v/Optimization-Homework/tree/master/homework6

[ ]:
```
with open("./coins.dat", 'r') as f:
    x = f.readlines()
m = int(x[0].strip())
k = int(x[1].strip())
coins = [int(i) for i in x[2].strip().split(" ")]


def minCoin(m, k, coins):
    ans = [float("inf")] + [float("inf")] * len(range(1, max(coins) * k + 1))

    for i in range(1, max(coins) * k + 1):

        for c in sorted(coins, reverse=True):
            if c > i:
                continue

            if i == c:
                ans[i] = 1

            if ans[i - c] + 1 < ans[i]:
                ans[i] = ans[i - c] + 1

    return ans

if __name__ == "__main__":
    ans = minCoin(m, k, coins)
    for ind, n in enumerate(ans):
        if n > k and ind > 0:
            print("the smallest value: ", ind)
            break

'''
the smallest value:  1509
'''
```