# Operations Research, Homework 3

B09705039 LIU, WEI-EN

## 1 Problem 1

### 1.1 (a)

Set $I = \{1, 2, ..., 4\}$, as the set of unknown points from left to right.
Set $x_a = 0, y_a = 100$, $x_b = 100, y_b = 150$.
Let $x_i$ be the x-coordinate of point i, $\forall i \in I$.
Let $y_i$ be the y-coordinate of point i, $\forall i \in I$.

Formulation:

$$
\begin{aligned}
\min \quad & \sum_{i \in I} y_i \\
\text{s.t.} \quad & (x_i - x_{i-1})^2 + (y_i - y_{i-1})^2 = L^2 \quad \forall i \in \{2, ..., 4\} \\
& (x_1 - x_a)^2 + (y_1 - y_a)^2 = L^2 \\
& (x_b - x_4)^2 + (y_b - y_4)^2 = L^2 \\
& x_{i-1} \leq x_i \quad \forall i \in \{2, ..., 4\} \\
& x_a \leq x_1 \\
& x_4 \leq x_b \\
& x_i \geq 0 \quad \forall i \in I
\end{aligned} \tag{1}
$$

Convex analysis:
Since the objective function is a linear function, it must be a convex function. However, the feasible region is not a convex set since the first three constraints are multiple circle lines instead of a convex area. Thus, this formulation is not a convex program.

### 1.2 (b)

Python Code:

```
from gurobipy import *
import pandas as pd

# Input
x_a = 0
y_a = 100
x_b = 100
y_b = 150
L = 25  # change this to 35 in the second instance

I = range(1, 5)
I_2 = range(2, 5)

# Modeling
# add var
p3_1 = Model("problem1-1")      # build a new model
```

```
x = p3_1.addVars(I, lb = 0, vtype = GRB.CONTINUOUS, name = "x")
y = p3_1.addVars(I, lb = 0, vtype = GRB.CONTINUOUS, name = "y")

# setting the objective function
p3_1.setObjective(
    (quicksum(y[i] for i in I))
    , GRB.MINIMIZE)

# add constraints and name them
for i in I_2:
    p3_1.addConstr((x[i] - x[i - 1]) ** 2 + (y[i] - y[i-1]) ** 2 == L **
        2, name = f"length")

p3_1.addConstr((x[1] - x_a) ** 2 + (y[1] - y_a) ** 2 == L ** 2, name = f"
    length_1")

p3_1.addConstr((x_b - x[4]) ** 2 + (y_b - y[4]) ** 2 == L ** 2, name = f"
    length_5")

for i in I_2:
    p3_1.addConstr(x[i - 1] <= x[i], name = f"xorder")

p3_1.addConstr(x_a <= x[1], name = f"xorder_1")

p3_1.addConstr(x[4] <= x_b, name = f"xorder_5")

p3_1.params.NonConvex = 2

p3_1.optimize()

# Results
print("Results:\n")

# objective value
LR_ov = p3_1.objVal
print("objective value =", LR_ov)
print("")

# x, y
for i in I:
    print("dot", i, ":(", x[i].x, y[i].x, ")")
```

Graph Results:

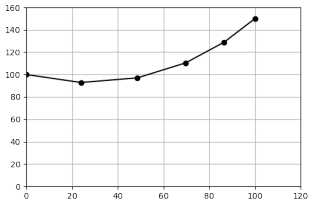


Figure 1: The position of the board when L = 25.

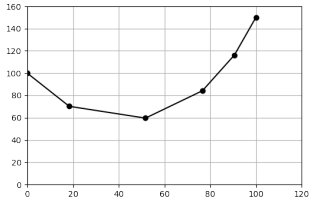


Figure 2: The position of the board when L = 35.

# 2 Problem 2

## 2.1 (a)

$o_i = 1$ if patent i is obtained, otherwise 0, $\forall i \in I$.
$x_j = 1$ if product j can be produced, otherwise 0, $\forall j \in J$.

Formulation:

$$\begin{aligned}
\max \quad & \sum_{j \in J} P_j x_j \\
\text{s.t.} \quad & x_j \leq \frac{1}{\sum_{i \in I} A_{ij}} \sum_{i \in I} (A_{ij} o_i) \quad \forall j \in J \\
& \sum_{i \in I} F_i o_i \leq B \quad \forall i \in I \\
& o_i \in \{0, 1\} \quad \forall i \in I \\
& x_j \in \{0, 1\} \quad \forall j \in J
\end{aligned} \tag{2}$$

## 2.2 (b)

Linear Relaxation of (a) after plug in the value and some rearrangement:

$$\begin{aligned}
\max \quad & 1600x_1 + 1000x_2 + 1600x_3 + 1200x_4 + 1100x_5 + 1400x_6 \\
\text{s.t.} \quad & 3x_1 - o_1 - o_2 - o_4 \leq 0 \\
& x_2 - o_4 \leq 0 \\
& 2x_3 - o_2 - o_3 \leq 0 \\
& 3x_4 - o_1 - o_2 - o_4 \leq 0 \\
& 2x_5 - o_3 - o_4 \leq 0 \\
& 2x_6 - o_1 - o_2 \leq 0 \\
& 1200o_1 + 1800o_2 + 2500o_3 + 1300o_4 \leq 4500 \\
& o_i \leq 1 \quad \forall i \in I \\
& x_j \leq 1 \quad \forall j \in J \\
& o_i \geq 0 \quad \forall i \in I \\
& x_j \geq 0 \quad \forall j \in J
\end{aligned} \tag{3}$$

Dual of the linear relaxation:

$$
\begin{aligned}
\max \quad & 4500y_7 + \sum_{i=8}^{17} y_i \\
\text{s.t.} \quad & 3y_1 + y_8 \geq 1600 \\
& x_2 + y_9 \geq 1000 \\
& 2x_3 + y_{10} \geq 1600 \\
& 3x_4 + y_{11} \geq 1200 \\
& 2x_5 + y_{12} \geq 1100 \\
& 2x_6 + y_{13} \geq 1400 \\
& y_i \geq 0 \quad \forall i \in \{1, 2, ..., 7\} \\
& y_i \leq 0 \quad \forall i \in \{8, 2, ..., 17\}
\end{aligned} \tag{4}
$$

## 2.3 (c)

Python Code:

```
from gurobipy import *
import pandas as pd

# Input
A = [[1, 0, 0, 1, 0, 1],
     [1, 0, 1, 1, 0, 1],
     [0, 0, 1, 0, 1, 0],
     [1, 1, 0, 1, 1, 0]]
F = [1200, 1800, 2500, 1300]
P = [1600, 1000, 1600, 1200, 1100, 1400]
B = 4500

I = range(1, 5)
J = range(1, 7)

a_sum = []

for j in J:
    sum1 = 0
    for i in I:
        sum1 += A[i - 1][j - 1]
    a_sum.append(sum1)

# Modeling
# add var
p3_1 = Model("problem1-1")      # build a new model

o = p3_1.addVars(I, lb = 0, ub = 1, vtype = GRB.CONTINUOUS, name = "o")
x = p3_1.addVars(J, lb = 0, ub = 1, vtype = GRB.CONTINUOUS, name = "x")

# setting the objective function
p3_1.setObjective(
    (quicksum(P[j - 1] * x[j] for j in J))
    , GRB.MAXIMIZE)

# add constraints and name them
for j in J:
    p3_1.addConstr(x[j] <= ((1 / a_sum[j - 1]) * (quicksum(A[i - 1][j -
        1] * o[i] for i in I))), name = f"length")
```

```
p3_1.addConstr(quicksum(F[i - 1] * o[i] for i in I) <= B, name = f"xorder
    ")


p3_1.optimize()


# Results
print("Results:\n")

# objective value
LR_ov = p3_1.objVal
print("objective value =", LR_ov)
print("")

# x, y
for i in I:
    print("o", i, o[i].x)

for j in J:
    print("x", j, x[j].x)
```

The optimal solution is $o_i = (1, 1, 0.08, 1), x_i = (1, 1, 0.54, 1, 0.54, 1)$, objective value = 6658.
Which implies that if we buy the patent for 1, 2 and 4, we can have the highest total profit earned at near 6658.
Since we solved the linear relaxation of this problem we may have some nonbinary variables¡ how ever we still can have some insights from it.

## 2.4 (d)

We can solve the dual program to find the shadow price of our budget constraint.

# 3 Problem 3

## 3.1 (a)

$o_i = 1$ if patent i is obtained, otherwise 0, $\forall i \in I$.
$x_j = 1$ if product j can be produced, otherwise 0, $\forall j \in J$.

$$\begin{aligned} \max \quad & \sum_{j \in J} P_j x_j \\ \text{s.t.} \quad & x_j \leq \frac{1}{\sum_{i \in I} A_{ij}} \sum_{i \in I} (A_{ij} o_i) \quad \forall j \in J \\ & o_i \in \{0, 1\} \quad \forall i \in I \\ & x_j \in \{0, 1\} \quad \forall j \in J \end{aligned} \tag{5}$$

## 3.2 (b)

It must satisfy the conditions below.
(1) All its elements are either 1, 0, or 1. Since $A_i j, o_i, x_j$ must be 1 or 0, all elements must be 1, -1 or 0.
(2) Each column contains at most two nonzero elements. Since there exist at most two variables in a row, this condition is satisfied.
(3) Rows can be divided into two groups so that for each column two nonzero elements are in the same group if and only if they are different. By switching the variables $o_i, x_i$ to the same side of their inequality, there will be only a 1 and $-1$ in each column. If we divide them into two groups they must satisfy this condition.
Thus, we can conclude that no matter how large I and J is, the coefficient matrix will always be totally unimodular.

# 4 Problem 4

## 4.1 (a)

$$f'(x) = 8x^3 - 6x + e^x$$
$$f''(x) = 24x^2 - 6 + e^x \geq 0$$
$$=> \{x \in R^2 | x \geq 0.431132, x \leq -0.4733339138247\}$$

## 4.2 (b)

$$G(x) = \begin{bmatrix} 3x_1^2 - 18x_1 + 72 \\ -2x_2 + 10 \end{bmatrix}, H(x) = \begin{bmatrix} 6x_1 - 18 & 0 \\ 0 & -2 \end{bmatrix}$$

To maintain PSD:

$$-2 < 0$$

It doesn't maintain PSD, the function is nowhere convex.

## 4.3 (c)

$$G(x) = \begin{bmatrix} 3x_1^2 - 2x_2 \\ -2x_1 + x_2 \\ x_3 \end{bmatrix}, H(x) = \begin{bmatrix} 6x_1 & -2 & 0 \\ -2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

To maintain PSD:

$$6x_1 \geq 0 => x_1 \geq 0$$

$$\begin{vmatrix} 6x_1 & -2 \\ -2 & 1 \end{vmatrix} \geq 0 => 6x_1 + 4 \geq 0 => x_1 \geq -\frac{2}{3}$$

$$\begin{vmatrix} 6x_1 & -2 & 0 \\ -2 & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix} \geq 0 => 6x_1 - 4 \geq 0 => x_1 \geq \frac{2}{3}$$

The rest principal minors are all nonnegative.

$$=> \{x \in R^2 | x_1 \geq \frac{2}{3}\}$$

# 5 Problem 5

## 5.1 (a)

Since we will move all the constraints to the $\geq 0$ side. To reward the solutions that satisfy these constraint the $\lambda$ should be nonnegative.

$$L(x|\lambda) = \sum_{i=1}^{n} v_i x_i + \lambda(B - \sum_{i=1}^{n}(w_i x_i))$$

Relaxed Program:

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} v_i x_i + \lambda(B - \sum_{i=1}^{n}(w_i x_i)) \\ \text{s.t.} \quad & x_i \in \{0,1\} \quad \forall i = 1, ..., n \end{aligned} \tag{6}$$

## 5.2 (b)

Rearanging the Relaxed Program:

$$\begin{aligned} \max \quad & \sum_{i=1}^{n}((x_i)(v_i - \lambda w_i)) + \lambda B \\ \text{s.t.} \quad & x_i \in \{0,1\} \quad \forall i = 1, ..., n \end{aligned} \tag{7}$$

After rearranging the program above, the only thing that will influence the objective function is the $((x_i)(v_i - \lambda w_i))$. So, we intuitively design the method below:
Since $\lambda$ is given, we can calculate $(v_i - \lambda w_i)$ to decide which item to pick.
If the $(v_i - \lambda w_i)$ of item i is positive, we select the item. Thus, its $x_i$ will equal 1 and the objective function gets better.
If the $(v_i - \lambda w_i)$ of item i is nonpositive, we do not select the item. Thus, its $x_i$ will equal 0 and the objective function will not get worse.

## 5.3 (c)

$(v_i - \lambda w_i) = (5.6, 4.2, 2.4, 3.0, 1.8, -1.6, 0.6)$
After calculating $(v_i - \lambda w_i)$ only item 6 is nonpositive. Thus we select all the others instead of item 6.
The objective value = 41.6, the optimal solution is $x_i = (1, 1, 1, 1, 1, 0, 1)$ for the relaxed problem.
The optimal solution is not feasible to the original knapsack problem since it violates the first constraint $(33 > 30)$.

## 5.4 (d)

$(v_i - \lambda w_i) = (3.2, 2.4, 0.3, 1.5, 0.6, -3.7, -0.3)$
After calculating $(v_i - \lambda w_i)$ only item 6 and 7 are nonpositive. Thus we select all the others instead of item 6 and 7.
The objective value = 41.0, the optimal solution is $x_i = (1, 1, 1, 1, 1, 0, 0)$ for the relaxed problem.
The optimal solution is feasible to the original knapsack problem since it does not violate the first constraint $(30 >= 30)$. The objective value = 41.0.

## 5.5 (e)

According to this algorithm, we obtain the best feasible optimal solution $w_i = (1, 1, 1, 1, 1, 0, 0)$ where $\lambda = 1.002$, and the objective value = 41, which is almost same as the solution in part (d). Thus, we

expect the $\lambda$ value to be close to 1 to get the best feasible solution from the relaxed program. By solving the original program, we can find the exact same feasible optimal solution and objective value. Thus we can conclude that this algorithm will performs very well nearly no error when $\lambda = 1.002$ or 1.1.