Algorithm Fourth Homework

Zhang Yuan

Student ID: 2015E8013261189

1 Problem 3: Interval Scheduling Problem

1.

This problem can be formulated as follows:

$$maximize \quad \sum_{i,j} y_{ij}$$

y_{ij} means in the ith classroom, whether arranging the jth class.
s.t.

$$y_{ij} = 0 \quad or \quad 1$$
$$\sum_{i=1}^{m} y_{ij} \leq 1 \quad j = 1, ..., n$$
$$y_{ij} + y_{ik} \leq 1 \quad i = 1, ..., m$$

jth job can not arrange in several classrooms.
jth and kth job can not arrange in the same classroom.
I construct an instance as follows:

$$\begin{bmatrix} j & 1 & 2 & 3 & 4 & 5 & 6 \\ S & 0 & 1 & 1 & 3 & 4 & 5 \\ F & 5 & 3 & 4 & 8 & 6 & 7 \end{bmatrix}$$

My input as follows:

```
/* Variables */
var y_11 >= 0, integer;
var y_21 >= 0, integer;
var y_12 >= 0, integer;
var y_22 >= 0, integer;
var y_13 >= 0, integer;
var y_23 >= 0, integer;
var y_14 >= 0, integer;
var y_24 >= 0, integer;
var y_15 >= 0, integer;
var y_25 >= 0, integer;
var y_16 >= 0, integer;
var y_26 >= 0, integer;
```

```
/*var x >= 0, integer;
var y >= 0, integer;
var z >= 0, integer;*/

/* Object function */
maximize q: y_11 + y_12 + y_21 + y_22 + y_13 + y_23 +y_14+ y_24 + y_15 + y_
/*minimize q: -2*x + -3*y + -2*z;  */

/* Constraints */
s.t. con1: y_11 + y_21 <= 1;
s.t. con2: y_12 + y_22 <= 1;
s.t. con3: y_13 + y_23 <= 1;
s.t. con4: y_14 + y_24 <= 1;
s.t. con5: y_15 + y_25 <= 1;
s.t. con6: y_16 + y_26 <= 1;
s.t. con7: y_11 + y_12 <= 1;
s.t. con8: y_21 + y_22 <= 1;
s.t. con9: y_11 + y_13 <= 1;
s.t. con10: y_21 + y_23 <= 1;
s.t. con11: y_11 + y_14 <= 1;
s.t. con12: y_21 + y_24 <= 1;
s.t. con13: y_11 + y_15 <= 1;
s.t. con14: y_21 + y_25 <= 1;
s.t. con15: y_12 + y_13 <= 1;
s.t. con16: y_22 + y_23 <= 1;
s.t. con17: y_13 + y_14 <= 1;
s.t. con18: y_23 + y_24 <= 1;
s.t. con19: y_14 + y_15 <= 1;
s.t. con20: y_24 + y_25 <= 1;
s.t. con21: y_14 + y_16 <= 1;
s.t. con22: y_24 + y_26 <= 1;
s.t. con23: y_15 + y_16 <= 1;
s.t. con24: y_25 + y_26 <= 1;
/*s.t. con1: -2*x + -y + -z >= -4;
s.t. con2: -x + -2*y + -z >= -7;
s.t. con3: -z >= -5;  */

end;
```

The output solved by GLPK as follows:

```
Problem:    glpsolEx
Rows:       25
Columns:    12 (12 integer, 0 binary)
Non-zeros:  60
Status:     INTEGER OPTIMAL
Objective:  q = 4 (MAXimum)

   No.   Row name        Activity     Lower bound   Upper bound
```

```
------ ------------    ------------- ------------- -------------
     1 q                           4
     2 con1                        1                             1
     3 con2                        1                             1
     4 con3                        0                             1
     5 con4                        1                             1
     6 con5                        0                             1
     7 con6                        1                             1
     8 con7                        1                             1
     9 con8                        1                             1
    10 con9                        0                             1
    11 con10                       1                             1
    12 con11                       1                             1
    13 con12                       1                             1
    14 con13                       0                             1
    15 con14                       1                             1
    16 con15                       1                             1
    17 con16                       0                             1
    18 con17                       1                             1
    19 con18                       0                             1
    20 con19                       1                             1
    21 con20                       0                             1
    22 con21                       1                             1
    23 con22                       1                             1
    24 con23                       0                             1
    25 con24                       1                             1

   No. Column name       Activity     Lower bound   Upper bound
------ ------------    ------------- ------------- -------------
     1 y_11         *              0             0
     2 y_21         *              1             0
     3 y_12         *              1             0
     4 y_22         *              0             0
     5 y_13         *              0             0
     6 y_23         *              0             0
     7 y_14         *              1             0
     8 y_24         *              0             0
     9 y_15         *              0             0
    10 y_25         *              0             0
    11 y_16         *              0             0
    12 y_26         *              1             0

Integer feasibility conditions:

KKT.PE: max.abs.err = 0.00e+00 on row 0
        max.rel.err = 0.00e+00 on row 0
        High quality
```

```
KKT.PB: max.abs.err = 0.00e+00 on row 0
        max.rel.err = 0.00e+00 on row 0
        High quality

End of output
```

We can see the optimal solution is 4, which arrange class 1 and class 6 on room 2, and arrange class 2 and class 4 on room 1. After analyzing the instance, we know the result is right, but we can also find other suitable arrangements.

2.

No, if we change the requirement above, set $s_6 = 10$ and $f_6 = 12$, apparently we will get answer is 5, we can get the answer as follows:

```
Problem:    glpsolEx
Rows:       21
Columns:    12
Non-zeros:  52
Status:     OPTIMAL
Objective:  q = 6 (MAXimum)

   No.   Row name   St   Activity     Lower bound   Upper bound    Marginal
------ ------------ -- ------------- ------------- ------------- ----------
     1 q            B              6
     2 con1         NU             1                           1
     3 con2         NU             1                           1
     4 con3         NU             1                           1
     5 con4         NU             1                           1          <
     6 con5         B              1                           1
     7 con6         NU             1                           1
     8 con7         NU             1                           1          <
     9 con8         B              1                           1
    10 con9         NU             1                           1          <
    11 con10        B              1                           1
    12 con11        B              1                           1
    13 con12        B              1                           1
    14 con13        NU             1                           1          <
    15 con14        B              1                           1
    16 con15        B              1                           1
    17 con16        NU             1                           1          <
    18 con17        B              1                           1
    19 con18        B              1                           1
    20 con19        NU             1                           1
    21 con20        NU             1                           1

   No. Column name  St   Activity     Lower bound   Upper bound    Marginal
------ ------------ -- ------------- ------------- ------------- ----------
     1 y_11         B            0.5             0
     2 y_21         B            0.5             0
     3 y_12         B            0.5             0
```

```
     4 y_22         B             0.5             0
     5 y_13         B             0.5             0
     6 y_23         B             0.5             0
     7 y_14         B             0.5             0
     8 y_24         B             0.5             0
     9 y_15         B             0.5             0
    10 y_25         B             0.5             0
    11 y_16         B               1             0
    12 y_26         NL              0             0                       <

Karush-Kuhn-Tucker optimality conditions:

KKT.PE: max.abs.err = 0.00e+00 on row 0
        max.rel.err = 0.00e+00 on row 0
        High quality

KKT.PB: max.abs.err = 0.00e+00 on row 0
        max.rel.err = 0.00e+00 on row 0
        High quality

KKT.DE: max.abs.err = 0.00e+00 on column 0
        max.rel.err = 0.00e+00 on column 0
        High quality

KKT.DB: max.abs.err = 0.00e+00 on row 0
        max.rel.err = 0.00e+00 on row 0
        High quality

End of output
```

But in ILP, the answer as follows:

```
Problem:    glpsolEx
Rows:       21
Columns:    12 (12 integer, 0 binary)
Non-zeros:  52
Status:     INTEGER OPTIMAL
Objective:  q = 5 (MAXimum)

   No.   Row name        Activity     Lower bound   Upper bound
------ ------------    ------------- ------------- -------------
     1 q                           5
     2 con1                        0                           1
     3 con2                        1                           1
     4 con3                        1                           1
     5 con4                        1                           1
     6 con5                        1                           1
     7 con6                        1                           1
     8 con7                        1                           1
```

```
     9 con8                          0                            1
    10 con9                          0                            1
    11 con10                         1                            1
    12 con11                         1                            1
    13 con12                         0                            1
    14 con13                         0                            1
    15 con14                         1                            1
    16 con15                         1                            1
    17 con16                         1                            1
    18 con17                         1                            1
    19 con18                         1                            1
    20 con19                         1                            1
    21 con20                         1                            1

   No. Column name       Activity     Lower bound   Upper bound
------ ------------    ------------- ------------- -------------
     1 y_11         *              0             0
     2 y_21         *              0             0
     3 y_12         *              1             0
     4 y_22         *              0             0
     5 y_13         *              0             0
     6 y_23         *              1             0
     7 y_14         *              1             0
     8 y_24         *              0             0
     9 y_15         *              0             0
    10 y_25         *              1             0
    11 y_16         *              0             0
    12 y_26         *              1             0

Integer feasibility conditions:

KKT.PE: max.abs.err = 0.00e+00 on row 0
        max.rel.err = 0.00e+00 on row 0
        High quality

KKT.PB: max.abs.err = 0.00e+00 on row 0
        max.rel.err = 0.00e+00 on row 0
        High quality

End of output
```

According to lemma, we can not always set each column of A has at most one +1 entry and at most one -1 entry.

2 Problem 5: Stable Matching Problem

I select condition 1 as my condition, the problem can be formulated as follows:

$$maximize \quad \sum_{i,j,l,k} y_{i,j,l,k}$$

$$s.t. \quad \sum_{i,l \in p} y_{i,j(i),l,k(l)} S_{i,j(i),l,k(l)} = 0 \quad p \in P$$

P means permutations including all pairs between man and woman, we can conclude the number of permutations is $(n!)^2$. So we wanna find a permutation can meet the constraint and in the mean time, its optimal function can also be max of all p in P. In an instance p of P, pairs are fixed, all we need to do is to meet the constraints and get the maximum.

I give an instance as follows:

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ A & 3 & 2 & 1 \\ B & 2 & 1 & 3 \\ C & 3 & 1 & 2 \end{bmatrix}$$

$$\begin{bmatrix} 0 & A & B & C \\ 1 & C & B & A \\ 2 & A & B & C \\ 3 & C & B & A \end{bmatrix}$$

My input as follows:

```
/* Variables */
var y_1_B_2_A >= 0, binary;
var y_1_A_2_B >= 0, binary;
var y_1_C_2_B >= 0, binary;
var y_1_B_2_C >= 0, binary;
var y_1_A_3_C >= 0, binary;
var y_1_C_3_A >= 0, binary;
var y_1_C_2_A >= 0, binary;
var y_1_A_2_C >= 0, binary;
var y_1_B_3_A >= 0, binary;
var y_1_A_3_B >= 0, binary;
var y_1_B_3_C >= 0, binary;
var y_1_C_3_B >= 0, binary;
var y_2_B_3_A >= 0, binary;
var y_2_A_3_B >= 0, binary;
var y_2_A_3_C >= 0, binary;
var y_2_C_3_A >= 0, binary;
var y_2_B_3_C >= 0, binary;
var y_2_C_3_B >= 0, binary;

/* Object function */
/*maximize q: y_1_A_2_B + y_1_A_3_C + y_2_B_3_C;*/
/*maximize r: y_1_A_2_C + y_1_A_3_B + y_2_C_3_B;*/
maximize s: y_1_B_2_A + y_1_B_3_C + y_2_A_3_C;
/*maximize t: y_1_B_2_C + y_1_B_3_A + y_2_C_3_A;*/
/*maximize u: y_1_C_2_A + y_1_C_3_B + y_2_A_3_B;*/
```

```
/*maximize v: y_1_C_2_B + y_1_C_3_A + y_2_B_3_A;*/

/* Constraints */
s.t. con1: y_1_A_2_B * 1 + y_1_A_3_C * 0 + y_2_B_3_C * 0 = 0;
s.t. con2: y_1_A_2_C * 1 + y_1_A_3_B * 1 + y_2_C_3_B * 1 = 0;
s.t. con3: y_1_B_2_A * 0 + y_1_B_3_C * 0 + y_2_A_3_C * 0 = 0;
s.t. con4: y_1_B_2_C * 1 + y_1_B_3_A * 0 + y_2_C_3_A * 1 = 0;
s.t. con5: y_1_C_2_A * 0 + y_1_C_3_B * 1 + y_2_A_3_B * 1 = 0;
s.t. con6: y_1_C_2_B * 0 + y_1_C_3_A * 1 + y_2_B_3_A * 0 = 0;
/*s.t. con1: S_1_B_2_A = 0;
s.t. con2: S_1_A_2_B = 1;
s.t. con3: S_1_C_2_B = 0;
s.t. con4: S_1_B_2_C = 1;
s.t. con5: S_1_A_3_C = 0;
s.t. con6: S_1_C_3_A = 1;
s.t. con7: S_1_C_2_A = 0;
s.t. con8: S_1_A_2_C = 1;
s.t. con9: S_1_B_3_A = 0;
s.t. con10: S_1_A_3_B = 1;
s.t. con11: S_1_B_3_C = 0;
s.t. con12: S_1_C_3_B = 1;
s.t. con13: S_2_B_3_A = 0;
s.t. con14: S_2_A_3_B = 1;
s.t. con15: S_2_A_3_C = 0;
s.t. con16: S_2_C_3_A = 1;
s.t. con17: S_2_B_3_C = 0;
s.t. con18: S_2_C_3_B = 1;*/

end;
```

My output as follows:

```
Problem:    StableMatch
Rows:       7
Columns:    12 (12 integer, 12 binary)
Non-zeros:  12
Status:     INTEGER OPTIMAL
Objective:  s = 3 (MAXimum)

   No.   Row name        Activity     Lower bound   Upper bound
------ ------------    ------------- ------------- -------------
     1 s                           3
     2 con1                        0            -0             =
     3 con2                        0            -0             =
     4 con3                        0            -0             =
     5 con4                        0            -0             =
     6 con5                        0            -0             =
     7 con6                        0            -0             =
```

```
   No. Column name       Activity     Lower bound   Upper bound
------ ------------    ------------- ------------- -------------
     1 y_1_B_2_A    *              1             0             1
     2 y_1_A_2_B    *              0             0             1
     3 y_1_B_2_C    *              0             0             1
     4 y_1_C_3_A    *              0             0             1
     5 y_1_A_2_C    *              0             0             1
     6 y_1_A_3_B    *              0             0             1
     7 y_1_B_3_C    *              1             0             1
     8 y_1_C_3_B    *              0             0             1
     9 y_2_A_3_B    *              0             0             1
    10 y_2_A_3_C    *              1             0             1
    11 y_2_C_3_A    *              0             0             1
    12 y_2_C_3_B    *              0             0             1

Integer feasibility conditions:

KKT.PE: max.abs.err = 0.00e+00 on row 0
        max.rel.err = 0.00e+00 on row 0
        High quality

KKT.PB: max.abs.err = 0.00e+00 on row 0
        max.rel.err = 0.00e+00 on row 0
        High quality

End of output
```

So we can see that, the answer means 1 and B, 2 and A, 3 and C are the stable pairs, we can verify it.

3 Problem 7: Simplex Algorithm

I use the big-M method to get the initial situation, and use simplex algorithm to solve the problem.

```
import numpy as np

class Simplex_Table:

    def __init__(self,obj):
        self.obj=obj
        self.rows=[]
        self.columns=[]

    def addConstraint(self,expression,value):
        self.rows.append(expression)
        self.columns.append(value)
```

```
def check(self):
    if max(self.obj[0:-1])<= 0: return 1
    else:return 0

def display(self):
    print '\n',np.matrix([self.obj]+self.rows)

def select_column(self):
    high=0
    index=[]
    for i in range(0,len(self.obj)-1):
        if self.obj[i]>high:
            index.append(i)
    if index==[]: return -1
    return index[0]

def select_row(self,col):
    right=[self.rows[i][-1] for i in range(len(self.rows))]
    left=[self.rows[i][col] for i in range(len(self.rows))]
    ratio=[]
    for i in range(len(right)):
        if left[i]==0:
            ratio.append(999*abs(max(right)))
            continue
        ratio.append(right[i]/left[i])
    if ratio[np.argmin(ratio)]<0:
        return -1
    return np.argmin(ratio)

def pivot(self,row,col):
    e=self.rows[row][col]
    self.rows[row]=self.rows[row]/e
    for r in range(len(self.rows)):
        if r==row:continue
        self.rows[r]=self.rows[r]-self.rows[r][col]*self.rows[row]
    self.obj-=self.obj[col]*self.rows[row]

def solve(self):
    # build a simplex table
    for i in range(len(self.rows)):
        self.obj+=[-99999]
        col=[0 for r in range(len(self.rows))]
        col[i]=-1
        self.rows[i]+=col+[self.columns[i]]
        self.rows[i]=np.array(self.rows[i],dtype=float)
    self.obj=np.array(self.obj+[0],dtype=float)
    self.display()
```

```
        checkNum=0
        value_value=0
        while not self.check():
            checkNum+=1
            if checkNum>9999:
                print "No answer!"
                break
            columns=self.select_column()
            rows=self.select_row(columns)
            if rows==-1:
                print "No optimal answer!"
                break
            self.pivot(rows,columns)
            print '\nPivot column: %s\n Pivot row: %s'%(columns+1,rows+1)
            self.display()


if __name__=='__main__':
    """
        In the program, we use the big-M method to set the initial situatio
        min z=-2x+-3y+-2z
        s.t.
        -2x+-y+-z-a=-4
        -x+-2y+-z-b=-7
        -z-c     =-5
        x,y,z,a,b,c>=0
        """
    t=Simplex_Table([2,3,2,0,0,0])
    t.addConstraint([-2,-1,-1,-1,0,0],-4)
    t.addConstraint([-1,-2,-1,0,-1,0],-7)
    t.addConstraint([0,0,-1,0,0,-1],-5)
    t.solve()
```

Compared the GLPK, we use two program solving the same problem, the answer shows below:
GLPK:

```
Problem:    glpsolEx
Rows:       4
Columns:    3
Non-zeros:  10
Status:     OPTIMAL
Objective:  q = -11 (MINimum)

   No.   Row name   St   Activity     Lower bound   Upper bound    Marginal
------ ------------ -- ------------- ------------- ------------- ----------
     1 q            B            -11
     2 con1         NL            -4            -4
     3 con2         NL            -7            -7
     4 con3         B             -1            -5
```

```
   No. Column name  St   Activity     Lower bound   Upper bound    Marginal
------ ------------ -- ------------- ------------- ------------- ----------
     1 x            NL             0             0
     2 y            B              3             0
     3 z            B              1             0

Karush-Kuhn-Tucker optimality conditions:

KKT.PE: max.abs.err = 0.00e+00 on row 0
        max.rel.err = 0.00e+00 on row 0
        High quality

KKT.PB: max.abs.err = 0.00e+00 on row 0
        max.rel.err = 0.00e+00 on row 0
        High quality

KKT.DE: max.abs.err = 0.00e+00 on column 0
        max.rel.err = 0.00e+00 on column 0
        High quality

KKT.DB: max.abs.err = 0.00e+00 on row 0
        max.rel.err = 0.00e+00 on row 0
        High quality

End of output
```

My Program:

```
[[  2.00000000e+00   3.00000000e+00   2.00000000e+00   0.00000000e+00
    0.00000000e+00   0.00000000e+00  -9.99990000e+04  -9.99990000e+04
   -9.99990000e+04   0.00000000e+00]
 [ -2.00000000e+00  -1.00000000e+00  -1.00000000e+00  -1.00000000e+00
    0.00000000e+00   0.00000000e+00  -1.00000000e+00   0.00000000e+00
    0.00000000e+00  -4.00000000e+00]
 [ -1.00000000e+00  -2.00000000e+00  -1.00000000e+00   0.00000000e+00
   -1.00000000e+00   0.00000000e+00   0.00000000e+00  -1.00000000e+00
    0.00000000e+00  -7.00000000e+00]
 [  0.00000000e+00   0.00000000e+00  -1.00000000e+00   0.00000000e+00
    0.00000000e+00  -1.00000000e+00   0.00000000e+00   0.00000000e+00
   -1.00000000e+00  -5.00000000e+00]]

Pivot column: 1
 Pivot row: 1

[[  0.00000000e+00   2.00000000e+00   1.00000000e+00  -1.00000000e+00
    0.00000000e+00   0.00000000e+00  -1.00000000e+05  -9.99990000e+04
```

```
   -9.99990000e+04  -4.00000000e+00]
 [  1.00000000e+00   5.00000000e-01   5.00000000e-01   5.00000000e-01
   -0.00000000e+00  -0.00000000e+00   5.00000000e-01  -0.00000000e+00
   -0.00000000e+00   2.00000000e+00]
 [  0.00000000e+00  -1.50000000e+00  -5.00000000e-01   5.00000000e-01
   -1.00000000e+00   0.00000000e+00   5.00000000e-01  -1.00000000e+00
    0.00000000e+00  -5.00000000e+00]
 [  0.00000000e+00   0.00000000e+00  -1.00000000e+00   0.00000000e+00
    0.00000000e+00  -1.00000000e+00   0.00000000e+00   0.00000000e+00
   -1.00000000e+00  -5.00000000e+00]]

Pivot column: 2
 Pivot row: 2

[[  0.00000000e+00   0.00000000e+00   3.33333333e-01  -3.33333333e-01
   -1.33333333e+00   0.00000000e+00  -9.99993333e+04  -1.00000333e+05
   -9.99990000e+04  -1.06666667e+01]
 [  1.00000000e+00   0.00000000e+00   3.33333333e-01   6.66666667e-01
   -3.33333333e-01   0.00000000e+00   6.66666667e-01  -3.33333333e-01
    0.00000000e+00   3.33333333e-01]
 [ -0.00000000e+00   1.00000000e+00   3.33333333e-01  -3.33333333e-01
    6.66666667e-01  -0.00000000e+00  -3.33333333e-01   6.66666667e-01
   -0.00000000e+00   3.33333333e+00]
 [  0.00000000e+00   0.00000000e+00  -1.00000000e+00   0.00000000e+00
    0.00000000e+00  -1.00000000e+00   0.00000000e+00   0.00000000e+00
   -1.00000000e+00  -5.00000000e+00]]

Pivot column: 3
 Pivot row: 1

[[ -1.00000000e+00   0.00000000e+00   0.00000000e+00  -1.00000000e+00
   -1.00000000e+00   0.00000000e+00  -1.00000000e+05  -1.00000000e+05
   -9.99990000e+04  -1.10000000e+01]
 [  3.00000000e+00   0.00000000e+00   1.00000000e+00   2.00000000e+00
   -1.00000000e+00   0.00000000e+00   2.00000000e+00  -1.00000000e+00
    0.00000000e+00   1.00000000e+00]
 [ -1.00000000e+00   1.00000000e+00   0.00000000e+00  -1.00000000e+00
    1.00000000e+00  -0.00000000e+00  -1.00000000e+00   1.00000000e+00
   -0.00000000e+00   3.00000000e+00]
 [  3.00000000e+00   0.00000000e+00   0.00000000e+00   2.00000000e+00
   -1.00000000e+00  -1.00000000e+00   2.00000000e+00  -1.00000000e+00
   -1.00000000e+00  -4.00000000e+00]]
```

We can see the results above, the answers are same, after testing several cases, I verify the fact that my program is correct, but wastes more time due to python's high time cost.