Karen Loscocco

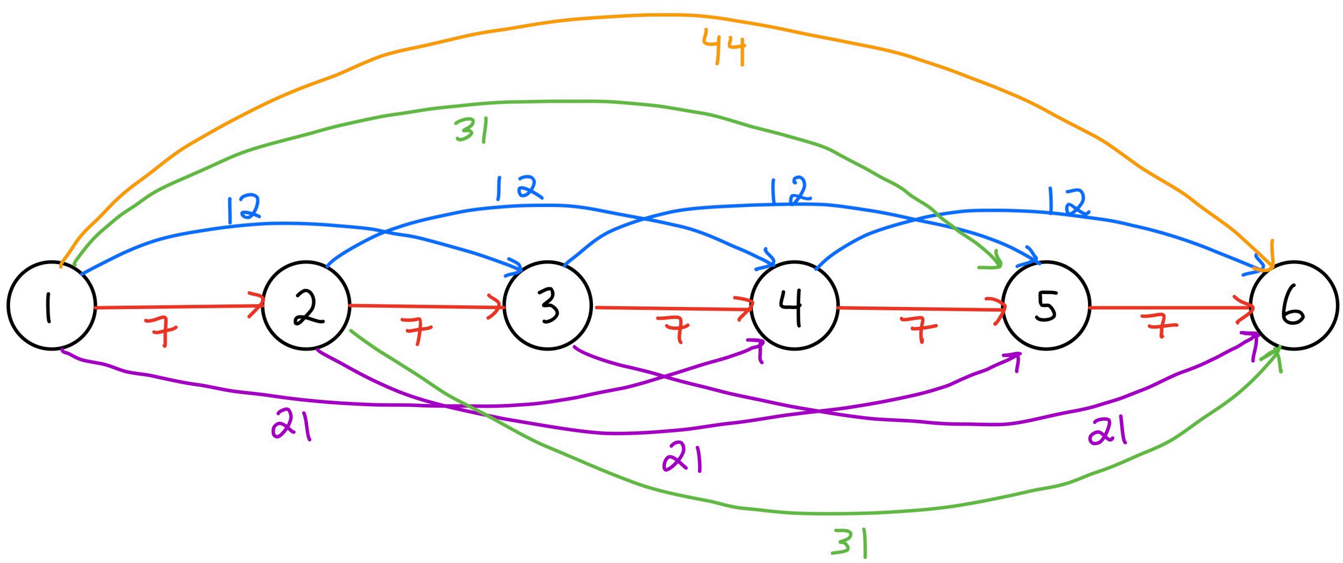
ISYE 4133 Assignment 5

I worked with Nicholas Loprinzo on this homework.

1

I used the following reference for the idea to model this problem as a shortest path problem: https://www.lix.polytechnique.fr/~liberti/teaching/isic/isc612-06/exercises.pdf (https://www.lix.polytechnique.fr/~liberti/teaching/isic/isc612-06/exercises.pdf)

The possibilities for this renewal plan can be modeled by the following graph:



The nodes 1 to 5 represent the start of each year. The 6th node represents the end of the 5th year or end of the planning period. For each node $i < 6$ and $i < j$, arc (i, j) represents the event that machinery bought at the beginning of the i-th year is sold at the beginning of the j-th year. The cost (in thousands of dollars) c_{ij} along each arc (i, j) is given by:

$$c_{ij} = 12 + \Big(\sum_{k=i}^{j-1} m_k\Big) - g_j$$

where m_k is the maintenance cost in the k-th year and g_j is the gain from selling the machinery at the beginning of the j-th year.

A renewal plan which minimizes the total operation cost is the shortest path from node 1 to 6 becasue the cost of the path is the cost of the plan.

Before formulating a linear integer program for this problem, we can use Dijkstra's algorithm to find the best renal plan. We have:

$$U = \{1, 2, 3, 4, 5, 6\}$$

Node	P_v	Predecessor of U
1	0	∅
2	∞	UD
3	∞	UD
4	∞	UD
5	∞	UD
6	∞	UD

$$U = \{2, 3, 4, 5, 6\}$$

Node	P_v	Predecessor of U
1	0	∅
2	7	1
3	12	1
4	21	1
5	31	1
6	44	1

$$U = \{3, 4, 5, 6\}$$

Node	P_v	Predecessor of U
1	0	∅
2	7	1
3	12	1
4	19	2
5	28	2
6	38	2

$$U = \{4, 5, 6\}$$

Node	P_v	Predecessor of U
1	0	∅
2	7	1
3	12	1
4	19	2
5	24	3
6	33	3

$$U = \{5, 6\}$$

Node	P_v	Predecessor of U
1	0	∅
2	7	1
3	12	1
4	19	2
5	24	3
6	31	4

$$U = \{6\}$$

Node	P_v	Predecessor of U

1	0	∅
2	7	1
3	12	1
4	19	2
5	24	3
6	31	4

$$\text{The shortest path has cost \$31,000:}$$
$$1 \rightarrow 2 \rightarrow 4 \rightarrow 6$$
$$\text{OR}$$
$$1 \rightarrow 3 \rightarrow 4 \rightarrow 6$$
$$\text{OR}$$
$$1 \rightarrow 3 \rightarrow 5 \rightarrow 6$$

The following is the integer program:

Nodes $V = \{1, 2, 3, 4, 5, 6\}$
Arcs $A = \{(1,2), (1,3), (1,4), (1,5), (1,6), (2,3), (2,4), (2,5), (2,6), (3,4), (3,5), (3,6), (4,5), (4,6), (5,6)\}$

$$x_{ij} = \begin{cases} 1 & \text{if } i \rightarrow j \text{ is a path } p \\ 0 & \text{otherwise} \end{cases}$$

$$\min \sum_{(i,j)\in A} c_{ij} x_{ij}$$
$$\sum_{i:(1,i)\in A} x_{1,i} = 1$$
$$\sum_{i:(i,6)\in A} x_{i,6} = 1$$
$$\forall j \in V \backslash \{1, 6\} \sum_{i:(i,j)\in A} x_{ij} - \sum_{i:(j,i)\in A} x_{ji} = 0$$
$$x_{ij} \in \{0, 1\} \forall i, j$$

The following is the solution in Gurobi:

In [2]:
```
V = set([1,2,3,4,5,6])
c = np.array([7,12,21,31,44,7,12,21,31,7,12,21,7,12,7])
m = Model()
X = m.addVars([i for i in permutations(V,2) if i[1]>i[0]],vtype = GRB.
BINARY, name = 'X')
m.addConstr(sum(X[1,i] for i in V if i!=1) == 1)
m.addConstr(sum(X[i,6] for i in V if i!=6) == 1)

j = V-{1,6}

for v in j:
    m.addConstr(sum(X[i,v] for i in V if i<v) - sum(X[v,i] for i in V
if i>v) == 0)

m.setObjective(np.dot(c,X.values()), GRB.MINIMIZE)

m.optimize()

printOptimal(m)
```

```
Academic license - for non-commercial use only
Optimize a model with 6 rows, 15 columns and 30 nonzeros
Variable types: 0 continuous, 15 integer (15 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [7e+00, 4e+01]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+00]
Found heuristic solution: objective 44.0000000
Presolve removed 6 rows and 15 columns
Presolve time: 0.00s
Presolve: All rows and columns removed

Explored 0 nodes (0 simplex iterations) in 0.01 seconds
Thread count was 1 (of 8 available processors)

Solution count 2: 31 44

Optimal solution found (tolerance 1.00e-04)
Best objective 3.100000000000e+01, best bound 3.100000000000e+01, ga
p 0.0000%


Variable Values:
X[1,2] 0.0
X[1,3] 1.0
X[1,4] 0.0
X[1,5] 0.0
X[1,6] 0.0
X[2,3] 0.0
X[2,4] 0.0
X[2,5] 0.0
X[2,6] 0.0
X[3,4] 0.0
X[3,5] 1.0
X[3,6] 0.0
X[4,5] 0.0
X[4,6] 0.0
X[5,6] 1.0

Objective Value: 31.0
```

The following is the linear program relaxation:

$$\min \sum_{(i,j)\in A} c_{ij}x_{ij}$$

$$\sum_{i:(1,i)\in A} x_{1,i} = 1$$

$$\sum_{i:(i,6)\in A} x_{i,6} = 1$$

$$\forall j \in V\backslash\{1,6\} \sum_{i:(i,j)\in A} x_{ij} - \sum_{i:(j,i)\in A} x_{ji} = 0$$

$$x_{ij} \geq 0 \; \forall i,j$$

The following is the solution in Gurobi:

In [3]:
```
V = set([1,2,3,4,5,6])
c = np.array([7,12,21,31,44,7,12,21,31,7,12,21,7,12,7])
m = Model()
X = m.addVars([i for i in permutations(V,2) if i[1]>i[0]], name = 'X')
m.addConstr(sum(X[1,i] for i in V if i!=1) == 1)
m.addConstr(sum(X[i,6] for i in V if i!=6) == 1)

j = V-{1,6}

for v in j:
    m.addConstr(sum(X[i,v] for i in V if i<v) - sum(X[v,i] for i in V
if i>v) == 0)

m.setObjective(np.dot(c,X.values()), GRB.MINIMIZE)

m.optimize()

printOptimal(m)
```

```
Optimize a model with 6 rows, 15 columns and 30 nonzeros
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [7e+00, 4e+01]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+00, 1e+00]
Presolve removed 2 rows and 9 columns
Presolve time: 0.01s
Presolved: 4 rows, 6 columns, 12 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    1.9000000e+01   2.000000e+00   0.000000e+00      0s
       2    3.1000000e+01   0.000000e+00   0.000000e+00      0s

Solved in 2 iterations and 0.01 seconds
Optimal objective  3.100000000e+01


Variable Values:
X[1,2] 1.0
X[1,3] 0.0
X[1,4] 0.0
X[1,5] 0.0
X[1,6] 0.0
X[2,3] 0.0
X[2,4] 1.0
X[2,5] 0.0
X[2,6] 0.0
X[3,4] 0.0
X[3,5] 0.0
X[3,6] 0.0
X[4,5] 0.0
X[4,6] 1.0
X[5,6] 0.0

Objective Value: 31.0
```

The linear program relaxation will always give the integer solution for free.

2

x_j: amount of widgets produced in time period j. $\forall j = 1, \ldots, K$
s_j: amount of widgets held in inventory over in time period j. $\forall j = 1, \ldots, K$

$$\min \sum_{j=1}^{K} x_j c_j + h_j s_j$$
$$x_j + s_{j-1} = d_j + s_j \ \forall j$$
$$p_j \geq 0 \ \forall j$$
$$s_j \geq 0 \ \forall j$$
$$s_0 = 0$$

3 (a)

x_j: if item j is included in the subset. $\forall j = 1, \ldots, n$

$$\min \sum_j c_j x_j$$
$$\sum_j a_j x_j \leq B$$
$$x_j \in \{0, 1\}$$

3 (b)

The following table outlines the penalties for the i-th unit of increase in B:

i	**penalty p_i for the i-th unit**
1	2
2	3
3	3
4	6
5	6
6	10
7	10
8	10

9	10
10	10

B can be increased by a certain amount, $S = \sum_i a_i x_i - B$

$$y_i = \begin{cases} 1 & \text{if } S \text{ is } \geq i \qquad \forall i = 1, \ldots, 10 \\ 0 & \text{otherwise} \end{cases}$$

For example, if B increases by 6, then $y_1 = y_2 = y_3 = y_4 = y_5 = y_6 = 1$ and then this will increase costs by $1(2) + 1(3) + 1(3) + 1(6) + 1(6) + 1(10) = 30$

The cost function is altered to include this added penalty cost ($\sum p_i y_i$).

The following logic can be used to create the constraints:
If $S - 1 \geq 0$ then $y_1 = 1$
If $S - 2 \geq 0$ then $y_2 = 1$
If $S - 3 \geq 0$ then $y_3 = 1$
And so on...
The constraints become:
$-(y_1 - 1) \leq Mz_1 \;,\; S - 1 \leq M(1 - z_1)$
$-(y_2 - 1) \leq Mz_2 \;,\; S - 2 \leq M(1 - z_2)$
$-(y_3 - 1) \leq Mz_3 \;,\; S - 3 \leq M(1 - z_3)$

The following is the integer linear program:

$$\min \sum_j c_j x_j + \sum_i p_i y_i$$

$$\sum_j a_j x_j - B \leq S$$

$$-(y_i - 1) \leq Mz_i \; \forall i = 1, \ldots, 10$$

$$S - i \leq M(1 - z_i) \; \forall i = 1, \ldots, 10$$

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

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

$$S \geq 0$$

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

3 (c)

For this change, the penalty table might look like this:

i	**penalty p_i for the i-th unit**
1	2
2	1
3	.9
4	.8
5	.7
6	.6
7	.5
8	.4
9	.3
10	.2

Because the only thing that changes in this problem is that the p_i values are different, the whole formulation remains the same.

Based on the above example non-increasing penalty table, if B increases by 6, then $y_1 = y_2 = y_3 = y_4 = y_5 = y_6 = 1$ and then this will increase costs by $1(2) + 1(1) + 1(.9) + 1(.8) + 1(.7) + 1(.6) = 6$

4 (a)

w_j: number of of cores per program
q_j: number of programs desired
x_{ij}: number of programs with w_j number of cores

Let $F = \{P = (a_{P,1}, \ldots, a_{P,n}) : \sum_{i=1}^{n} w_j a_{P,j} \leq W, a_{P,j} \in Z_+ \ \forall 1 \leq j \leq n\}$be the set of all valid patterns.

Then solve:

$$\min \sum_{P \in F} x_P$$

$$\sum_{P \in F} x_p a_{P,j} = b_j \ \forall j = 1, \ldots, n$$

$$x_p \geq 0 \ \forall P \in F$$

In [5]:
```
w = np.array([20,30,32,16,5])
q = np.array([1000,1500,2000,2500,1200])
```

Get all possible patterns:

In [6]:
```
L = min(w)
R = 64
n = len(w)

m = Model()

m.params.PoolSolutions = 400

m.params.PoolSearchMode = 2

y = m.addVars(n,vtype = GRB.INTEGER, name = 'Y')

m.addConstr(np.dot(w,y.values()) <= R)

m.setObjective(0, GRB.MINIMIZE)

m.optimize()

printOptimal(m)
```

```
Changed value of parameter PoolSolutions to 400
   Prev: 10   Min: 1   Max: 2000000000   Default: 10
```

```
Changed value of parameter PoolSearchMode to 2
   Prev: 0  Min: 0  Max: 2  Default: 0
Optimize a model with 1 rows, 5 columns and 5 nonzeros
Variable types: 0 continuous, 5 integer (0 binary)
Coefficient statistics:
  Matrix range     [5e+00, 3e+01]
  Objective range  [0e+00, 0e+00]
  Bounds range     [0e+00, 0e+00]
  RHS range        [6e+01, 6e+01]
Found heuristic solution: objective 0.0000000
Presolve time: 0.00s
Presolved: 1 rows, 5 columns, 5 nonzeros
Variable types: 0 continuous, 5 integer (0 binary)

Root relaxation: objective 0.000000e+00, 0 iterations, 0.00 seconds

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

     0     0          -    0         0.00000    0.00000  0.00%     -
0s
Optimal solution found at node 0 - now completing solution pool...
     0     0          -    0         0.00000    0.00000  0.00%     -
0s
     0     0          -    0         0.00000    0.00000  0.00%     -
0s
     0     2          -    0         0.00000    0.00000  0.00%     -
0s

Explored 187 nodes (0 simplex iterations) in 0.02 seconds
Thread count was 8 (of 8 available processors)

Solution count 94: 0 0 0 ... 0
No other solutions better than 1e+100

Optimal solution found (tolerance 1.00e-04)
Best objective 0.000000000000e+00, best bound 0.000000000000e+00, ga
p 0.0000%


Variable Values:
Y[0] -0.0
Y[1] -0.0
Y[2] -0.0
Y[3] -0.0
Y[4] -0.0

Objective Value: 0.0
```

In [7]:
```
NumSolns = m.solCount

solutions = np.empty([n, NumSolns])

for x in range(NumSolns):
    m.params.outputFlag = 0
    m.params.SolutionNumber = x
    for i in range(n):
        solutions[i,x] = m.Xn[i]
```

There are 94 possible patterns.

Now solve the LP:

In [9]:
```
m = Model()

x = m.addVars(NumSolns, vtype = GRB.CONTINUOUS, name = 'X')

for i in range(solutions.shape[0]):
    m.addConstr(np.dot(solutions[i,:],x.values()) >= q[i])

m.setObjective(sum(x.values()), GRB.MINIMIZE)

m.optimize()

printOptimal(m)
```

```
Optimize a model with 5 rows, 94 columns and 181 nonzeros
Coefficient statistics:
  Matrix range     [1e+00, 1e+01]
  Objective range  [1e+00, 1e+00]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+03, 2e+03]
Presolve removed 0 rows and 72 columns
Presolve time: 0.00s
Presolved: 5 rows, 22 columns, 46 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    0.0000000e+00   2.775000e+03   0.000000e+00      0s
       5    2.8083333e+03   0.000000e+00   0.000000e+00      0s

Solved in 5 iterations and 0.01 seconds
Optimal objective  2.808333333e+03



Objective Value: 2808.3333333333335
```

Now solved with Column Generation:

In [11]:
```
def solveRestrictedLP(I):

    m = Model()

    m.params.outputFlag = 0

    x = m.addVars(len(I), vtype = GRB.CONTINUOUS, name = 'X')

    for i in range(solutions.shape[0]):
        m.addConstr(np.dot(solutions[i,I],x.values()) >= q[i])

    m.setObjective(sum(x.values()), GRB.MINIMIZE)

    m.optimize()

    y = m.Pi

    return(m.X, m.VBasis, m.objVal, y)
```

In [12]:
```
def solveReducedCostLP(y):
    m = Model()

    m.params.outputFlag = 0

    a = m.addVars(solutions.shape[0], vtype = GRB.INTEGER, name = 'A')

    m.addConstr(np.dot(w,a.values()) <= 64)

    m.setObjective((1 - np.dot(a.values(),y)) , GRB.MINIMIZE)

    m.optimize()

    for i in range(solutions.shape[1]):
        if (list(solutions.T[i]) == list(np.floor(m.X))):
            newpattern = i

    return(m.X, m.objVal, newpattern)
```

In [13]:
```
w = np.array([20,30,32,16,5])
q = np.array([1000,1500,2000,2500,1200])

I = []

for i in range(NumSolns):
    if sum(solutions[:,i]) == 1:
        I.append(i)
```

In [14]:
```
obj = -10000
while obj < 0:
    X, VBasis, objVal, y = solveRestrictedLP(I)
    x, obj, newpattern = solveReducedCostLP(y)
    I.append(newpattern)
    print('-----')
    print(obj)
    print(objVal)
```

```
-----
-11.0
8200.0
-----
-3.0
7100.0
-----
-2.0
5225.0
-----
-1.0
4558.333333333333
-----
-1.0
3808.3333333333335
-----
0.0
2808.333333333333
```

4 (b)

More constraints must be added to account for the RAM and bandwidth requirements.

In [15]:
```
w = np.array([20,30,32,16,5])
q = np.array([1000,1500,2000,2500,1200])
ram = np.array([24,8,16,20,5])
bandwidth = np.array([0,2,1,.05,2.8])
```

In [16]:
```
n = len(w)

m = Model()

m.params.PoolSolutions = 400

m.params.PoolSearchMode = 2

y = m.addVars(n,vtype = GRB.INTEGER, name = 'Y')

m.addConstr(np.dot(w,y.values()) <= 64)
m.addConstr(np.dot(ram,y.values()) <= 32)
m.addConstr(np.dot(bandwidth,y.values()) <= 4)

m.setObjective(0, GRB.MINIMIZE)

m.optimize()

printOptimal(m)
```

```
Changed value of parameter PoolSolutions to 400
   Prev: 10  Min: 1  Max: 2000000000  Default: 10
Changed value of parameter PoolSearchMode to 2
   Prev: 0  Min: 0  Max: 2  Default: 0
Optimize a model with 3 rows, 5 columns and 14 nonzeros
Variable types: 0 continuous, 5 integer (0 binary)
Coefficient statistics:
  Matrix range     [5e-02, 3e+01]
  Objective range  [0e+00, 0e+00]
  Bounds range     [0e+00, 0e+00]
  RHS range        [4e+00, 6e+01]
Found heuristic solution: objective 0.0000000
Presolve time: 0.00s
Presolved: 3 rows, 5 columns, 14 nonzeros
Variable types: 0 continuous, 5 integer (3 binary)

Root relaxation: objective 0.000000e+00, 0 iterations, 0.00 seconds

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

     0     0          -    0          0.00000    0.00000  0.00%     -
0s
Optimal solution found at node 0 - now completing solution pool...
     0     0          -    0          0.00000    0.00000  0.00%     -
0s
     0     0          -    0          0.00000    0.00000  0.00%     -
```

```
0s
     0     2           -     0          0.00000    0.00000   0.00%     -
0s

Explored 27 nodes (0 simplex iterations) in 0.01 seconds
Thread count was 8 (of 8 available processors)

Solution count 14: 0 0 0 ... 0
No other solutions better than 1e+100

Optimal solution found (tolerance 1.00e-04)
Best objective 0.000000000000e+00, best bound 0.000000000000e+00, ga
p 0.0000%


Variable Values:
Y[0] -0.0
Y[1] -0.0
Y[2] -0.0
Y[3] -0.0
Y[4] -0.0

Objective Value: 0.0
```

In [17]:

```
NumSolns = m.solCount

solutions = np.empty([n, NumSolns])

for x in range(NumSolns):
    m.params.outputFlag = 0
    m.params.SolutionNumber = x
    for i in range(n):
        solutions[i,x] = m.Xn[i]
```

In [18]:

```
m = Model()

x = m.addVars(NumSolns, vtype = GRB.CONTINUOUS, name = 'X')

for i in range(solutions.shape[0]):
    m.addConstr(np.dot(solutions[i,:],x.values()) >= q[i])

m.setObjective(sum(x.values()), GRB.MINIMIZE)

m.optimize()

printOptimal(m)
```

```
Optimize a model with 5 rows, 14 columns and 19 nonzeros
Coefficient statistics:
  Matrix range     [1e+00, 2e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+03, 2e+03]
Presolve removed 0 rows and 3 columns
Presolve time: 0.00s
Presolved: 5 rows, 11 columns, 17 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    0.0000000e+00   6.450000e+03   0.000000e+00      0s
       6    4.5000000e+03   0.000000e+00   0.000000e+00      0s

Solved in 6 iterations and 0.01 seconds
Optimal objective  4.500000000e+03


Variable Values:
X[0] 0.0
X[1] 0.0
X[2] 800.0
X[3] 0.0
X[4] 0.0
X[5] 1000.0
X[6] 1200.0
X[7] 0.0
X[8] 0.0
X[9] 500.0
X[10] 0.0
X[11] 0.0
X[12] 1000.0
X[13] 0.0

Objective Value: 4500.0
```

In [19]:
```
def solveRestrictedLP(I):

    m = Model()

    m.params.outputFlag = 0

    x = m.addVars(len(I), vtype = GRB.CONTINUOUS, name = 'X')

    for i in range(solutions.shape[0]):
        m.addConstr(np.dot(solutions[i,I],x.values()) >= q[i])

    m.setObjective(sum(x.values()), GRB.MINIMIZE)

    m.optimize()

    y = m.Pi

    return(m.X, m.VBasis, m.objVal, y)
```

In [20]:
```
def solveReducedCostLP(y):
    m = Model()

    m.params.outputFlag = 0

    a = m.addVars(solutions.shape[0], vtype = GRB.INTEGER, name = 'A')

    m.addConstr(np.dot(w,a.values()) <= 64)
    m.addConstr(np.dot(ram,a.values()) <= 32)
    m.addConstr(np.dot(bandwidth,a.values()) <= 4)

    m.setObjective((1 - np.dot(a.values(),y)) , GRB.MINIMIZE)

    m.optimize()

    for i in range(solutions.shape[1]):
        if (list(solutions.T[i]) == list(np.floor(m.X))):
            newpattern = i

    return(m.X, m.objVal, newpattern)
```

In [21]:
```
w = np.array([20,30,32,16,5])
q = np.array([1000,1500,2000,2500,1200])

I = []

for i in range(NumSolns):
    if sum(solutions[:,i]) == 1:
        I.append(i)
```

In [22]:
```
obj = -10000
while obj < 0:
    X, VBasis, objVal, y = solveRestrictedLP(I)
    x, obj, newpattern = solveReducedCostLP(y)
    I.append(newpattern)
    print('-----')
    print(obj)
    print(objVal)
```

```
-----
-1.0
8200.0
-----
-1.0
7200.0
-----
-1.0
6950.0
-----
-1.0
5750.0
-----
-0.5
5350.0
-----
-0.5
5100.0
-----
0.0
4500.0
```