Mathematics_for_Decisions_Project

October 1, 2019

1 COLUMN GENERATION FOR CUTTING-STOCK PROBLEM

Cutting-Stock problem solved through a reinterpretation of the column generation method

Author: *MATTEO GARBELLI - Student ID : VR409285*

University of Verona

Academic year: 2018-2019

1.1 Abstract

In operations research, to solve a problem that is too large or too complex, it useful to decomposed it into easier problems: this is the key idea behind Column Generation (GC) method. CG is a scheme for solving large scale linear programming (LP) problems. Instead of pricing out nonbasic variables by enumeration, in a column generation approach the optimal reduced price is found by solving an optimization problem.In some cases, as in the cutting stock problem, the LP is a relaxation of an integer programming (IP) problem. However, when an LP relaxation is solved by column generation, the solution is not necessarily integer and it is not clear how to obtain an optimal or even a feasible integer solution to the initial IP. In this project an algorithm for the Cutting-Stock problem is presented and implemented. The problem solution gives not only the minimum number of rolls necessary to meet the demand, but also specifies the optimal patterns.

1.2 Chapter One: Overview on linear programming

Any Linear Programming (LP) problem can be written as

$$\min cx$$

such that

$$A'x \geq b'$$

$$x \geq 0$$

where $x \in \mathbb{R}^n$ is a vector of n real variables that represents the solution of the problem, $A' \in \mathbb{R}^{m \times n}$ is a matrix while $b' \in \mathbb{R}^m$ and $c \in \mathbb{R}^n \setminus$

Every LP problem has an associated dual LP problem that is expressed by

$$\max c'y$$

such that

$$Ay \leq b$$

$$y \geq 0$$

where $c = -c'$, $A = -A'$ and $b = -b'$.

The original problem is denoted as primal problem and by strong duality theorem we know that if the LP problem has an optimal solution, then objective optimal value of the dual formulation coincides with the one of primal. Indeed the two formulations for primal and dual LP are strictly connected since the first has m inequality constraints ans n non-negative variables and the second viceversa.

Since we are a going to solve this problem through a variation of the simplex method that is the Column Ceneration (CG) method, the LP problem must be modelled in standard form as

$$\min z = cx$$

such that

$$Ax + I\hat{x} = b$$

$$x \geq 0$$

$$\hat{x} \geq 0$$

Note that prevoius inequalities are written in form of linear equalities through the introduction of slack variables $\hat{x}$ and the objective function is expressed in minimization terms.

1.3 Chapter Two: formulation of the Cutting-Stock problem

The Cutting-Stock problem arises from many physical applications in industry. For example, in a paper mill, there are a number of rolls of paper of fixed width waiting to be cut, yet different firms want different numbers of rolls of various-sized widths.

How should the rolls be cut in order to minimize the wasted leftovers?

We start by formulating the problem as Integer Linear Programming (ILP).

Main parameters and assumptions necessary for constructing the model are:

- a set of items $i \in \mathcal{I} = (1, ..., m)$;
- to each item i a lenght l_i and a demand N_i are assigned;
- W_S is the width of the initial stock that has to be cut into smaller pieces;
- we assume that the cost of cutting a stock per unit width is equal to 1;
- $a_{\mathbf{ip}}$ number of pieces of item $i \in \mathcal{I}$ cut in pattern $p \in \mathcal{P} = (1, ..., n)$

The aim of the optimization problem is to minimize the cost function by finding the optimal patterns that minimize the number of initial stocks used (and so the wasted material) by

$$\min \sum_{\mathbf{p} \in \mathcal{P}} x_{\mathbf{p}}$$

where $(x_1, ..., x_n)$ are the decision variables assigned to each activities corresponding to the number of times a cutting pattern $\mathbf{p}$ is used.

The problem is subjected to the following constraints:

- demands $N_{\mathbf{i}}$ for each item $\mathbf{i}$ must be fulfilled

$$\sum_{\mathbf{p}\in\mathcal{P}} a_{\mathbf{ip}} x_{\mathbf{p}} \geq N_{\mathbf{i}} \qquad \forall \mathbf{i} \in \mathcal{I}$$

- variables $x_{\mathbf{p}}$ must be non-negative and assume integer values

$$x_{\mathbf{p}} \in \mathbb{Z}_+ \qquad \forall \mathbf{p} \in \mathcal{P}$$

- A final verification is required in order to check that the sum of the lenght of each item l_i does not exceed the initial stock width W_S

$$\sum_{\mathbf{i}\in\mathcal{I}} a_{\mathbf{ip}} l_{\mathbf{i}} \leq W_S \qquad \forall \mathbf{p} \in \mathcal{P}$$

First thing we modify the first inequality to have the standard formulation for the LP problem. Therefore, by introducing m slack variables $x_{n+1}, ..., x_{n+m}$ the Cutting-Stock problem can be solved through finding the n integer $(x_1, ..., x_n)$ that satisfies

$$\min \sum_{i=1,...,n} x_i$$
$$a_{i1}x_1 + ... + a_{in}x_n - x_{n+i} = N_i \quad i = (1, ..., m)$$
$$x_j \geq 0 \quad (j = 1, ..., n + m)$$

Since the beginning we underline the principal issues related to this model that are the size of variables involved, in particular of the decision variable and also the restriction to integers.

Considering the restriction to integers we can remove this assumption and solve the Linear Relaxation of the problem that is in general posed in linear programming setting that can have a solution in general is not integer. Rounding up to the nearest integer value can be a simple solution to deal with this warning but it essential to underline that solving the linear relaxation deals with a polynomial time algorithm (LP is a P-problem) while the original cutting-stock requires an algorithm that lies in the NP-complete class.

1.4 Chapter Three: Column Generation

The principal motivation behind Column Generation (CG) is that many linear programs are too large to consider all the variables and the constraints in an explicit way. Main issue to consider is that most of the variables involved are non basic that will assume a value of zero for the optimal solution and therefore only a subset of variables need to be considered. Column generation exploits this idea to generate such variables which have the potential to improve the objective function.

The problemis split into two simpler problems denoted as the master problem and the subproblems. * master problem is the original problem with only a subset of variables being considered (basic variables); * subproblems deals with the maximization in a feasible pattern dealing with a pricing problem. The objective function of the subproblem is the reduced cost of the new variable with respect to the current dual variables.

The process works as follows: the master problem is solved and from this solution, we are able to obtain dual prices related to each of the constraints in the master problem. Dual prices are then used to compute the reduced costs that measures the amount by which the variable in the

objective function needs to improve before it assumes a positive value. In the Cutting-Stock case the auxiliary problems can be formulated as knapsack problems. After solving the subproblem a variable with negative reduced cost can been identified and this variable is added to the master problem and the iterative process starts until no negative reduced cost variables are identified. When the subproblem returns a solution with non-negative reduced cost, we can conclude that the solution to the master problem is optimal.

1.4.1 Linear Relaxation

To answer we need to exploit the dual structure of the problem.

1. Consider new patterns; at every iteration we require a feaible basic solution for the primal problem and a dual solution satisfying the complementary slackness condtion.
2. If dealing with the new pattern it makes the dual constraint violated that in other words corresponds to the fact that the associated primal variable has negative reduced cost, the current solution is no optimal and it is suggested to let this variable enter the basis, generating a new column in the matrix.
3. The process iterates with the aim to find a pattern (corresponding to a new column) whit negative reduced cost in order to insert it in the matrix as column.

It is necessary to make a warning recalling to what we have said in the begininning: the numbers of patterns is exponentially great and since their combinatorial nature there is not an explicit way to express them.

In order to face off this aspect, we convert this issue into an optimization problem in the following way: since we are looking for a variable with a negative reduced cost, i.e. a violation for the dual constraint, we may require to find the minimum of the reduced costs of all possible variables and check that the minimum is negative that is

$$\min c = 1 - \sum_{i \in \mathcal{I}} y_i^T \bar{x}_i$$

such that $\bar{x}$ is a possible candidate column to enter the matrix.

Note that every column in the matrix represents a cutting pattern and each entry tells how many pieces of fixed lenght l_i are used in that pattern.

Considering also the contraints acting on $\bar{x}_i$ the subproblem can be formulated as

$$\min c = 1 - \sum_{i \in \mathcal{I}} y_i^T \bar{x}_i$$

$$\sum_{i \in \mathcal{I}} L_i \bar{x}_i \leq W$$

$$\bar{x}_i \in \mathbb{Z}_+$$

that can be equivalently posed in maximization form as

$$\max \sum_{i \in \mathcal{I}} y_i^T \bar{x}_i$$

$$\sum_{i \in \mathcal{I}} L_i \bar{x}_i \leq W$$

$$\bar{x}_i \in \mathbb{Z}_+$$

Therefore the coefficents $\bar{x}_i$ with negative reduced costs that are going to enter the matrix can be found through the solution of the above problem that is a known as integer knapsack problem.

1.4.2 Example 1

We start by finding a solution for the following example, requiring 10 as the initial stock lenght and the following orders and lenghts for each item to be cut

Item	Lenght	Orders
a	2	7
b	3	9
c	4	11

Firstly we need to start from a feasible solution that can be constructed by hand. As first guess we choose the patter [3,1,1] i.e. 3 items of lenght 2, 1 of lenght 3 and 1 of lenght 4. We suppose initially to consider 15 stocks. Of course we already know that it will be not the optimal solution since all the deamnd constraints are not only reached but also exceeded. Indeed through with configuration the objective functions $\Lambda_{[3,1,1]} = 15$ while for the constraints we have

$$15 \cdot \Lambda_{[3,1,1]} = 45 > 7$$
$$15 \cdot \Lambda_{[3,1,1]} = 15 > 9$$
$$15 \cdot \Lambda_{[3,1,1]} = 15 > 11$$

We can express the corresponding dual problem by introducing the variables (y_1, y_2, y_3) as

$$\max 7y_1 + 9y_2 + 11y_3$$
$$3y_1 + y_2 + y_3 \leq 1$$

Since the constrains of the primal are expressed as strict inequality the corresponding (y_1, y_2, y_3) will be take the value of 0. So from the equations $3y_1 + y_2 + y_3 = 1$ and $y_1 = y_2 = y_3 = 0$ we can deduce that an orthogonal dual solution does not exist since this first solution of the primal was not basic.

As second attempt we consider the same patter [3,1,1] but this time we select $\Lambda_{[3,1,1]} = 11$ in order to have the order of the item "c" is saturaded but not overtaken. With this choiche the third inequality constrains becomes an equality and the corresponding dual variable that is y_3 equals 0. This comes from the fact that the first dual variables must be zero because the corresponding constraints in the primal are valid and are strict inequalities. So having $y_1 = 0$ and $y_2 = 0$ and at the same time $3y_1 + y_2 + y_3 = 1$ we get the value for y_3.

Now there is a question that naturally arises: if we modify the structure of the pattern by considering another combination of items, is it true that the prevoius solution is still the optimal one? To answer it we need to exploit informations obtained by the dual formulation and reformulate the question: does exist a cutting pattern $[\alpha, \beta, \gamma]$ such that $\alpha y_1 + \beta y_2 + \gamma y_3 > 1$?

For this first step the question is traslated into the condition $\gamma > 1$. So we are going to select a pattern with the third entries greater than 1 and a possible feasible choice that satisfies the

restriction on the initial stock lenght is [2,0,2]. We select the following number of cut for each pattern

$$\Lambda_{[3,1,1]} = 9$$
$$\Lambda_{[2,0,2]} = 1$$

But there is again the possibility to consider a new pattern to going towards the optimality for the objective function. Thruogh this second iteration the second and third inequalities of the primal formulation are perfectly saturated while this does not happen for the first one that is connected to the first variable of the dual and that implicates $y_1 = 0$ and rewriting the dual constrainst we get

$$y_2 + y_3 = 1$$
$$2y_3 = 1$$

that is a 2x2 linear system with solution $(y_1, y_2) = \left(\frac{1}{2}, \frac{1}{2}\right)$ and so the arising question is the following: does it exist a cutting pattern with $\frac{\beta}{2} + \frac{\gamma}{2} > 1$?.

The only feasible pattern that we can add and that satisfies the last inequality is [0,2,1].

To find the best values for each pattern we solve the linear relaxation expressed by

$$\min \Lambda_{[3,1,1]} + \Lambda_{[2,0,2]} + \Lambda_{[0,2,1]}$$

such that

$$\begin{aligned} 3\Lambda_{[3,1,1]} + 2\Lambda_{[2,0,2]} + \quad\quad &\geq 7 \\ \Lambda_{[3,1,1]} + \quad\quad + 2\Lambda_{[0,2,1]} &\geq 9 \\ \Lambda_{[3,1,1]} + 2\Lambda_{[2,0,2]} + \Lambda_{[0,2,1]} &\geq 11 \end{aligned}$$

We also deal with its dual statement

$$\max 7y_1 + 9y_2 + 11y_3$$
$$\begin{aligned} 3y_1 + y_2 + y_3 &\leq 1 \\ 2y_1 + \quad + 2y_3 &\leq 1 \\ + 2y_2 + y_3 &\leq 1 \end{aligned}$$

We solve the first one minimization problem through SageMath by the command MixedIntegerLinearProgram:

```
In [12]: p=MixedIntegerLinearProgram( maximization=False )
         p.set_objective( p[1]+p[2]+p[3] )
         p.add_constraint( 3*p[1]+2*p[2], min = 7 )
         p.add_constraint( p[1]+2*p[3], min = 9 )
         p.add_constraint( p[1]+2*p[2]+p[3], min = 11 )
         p.solve()
         p.show()
         print('Objective Value: {}'.format(p.solve()))
         print ("The optimal values are x_1 = "+str(p.get_values(p[1]))+", x_2 = "
                +str(p.get_values(p[2]))+", x_3 = "+str(p.get_values(p[3])))
```

```
Minimization:
  x_0 + x_1 + x_2

Constraints:
  7.0 <= 3.0 x_0 + 2.0 x_1
  9.0 <= x_0 + 2.0 x_2
  11.0 <= x_0 + 2.0 x_1 + x_2
Variables:
  x_0 is a continuous variable (min=-oo, max=+oo)
  x_1 is a continuous variable (min=-oo, max=+oo)
  x_2 is a continuous variable (min=-oo, max=+oo)
Objective Value: 7.800000000000001
The optimal values are x_1 = 0.19999999999999987, x_2 = 3.2, x_3 = 4.4
```

So optimal values are $\Lambda_{[3,1,1]} = \frac{1}{5}$, $\Lambda_{[2,0,2]} = \frac{16}{5}$, $\Lambda_{[0,2,1]} = \frac{22}{5}$ that give an optimal value of $\frac{39}{5}$. For these values the constraints are perfectly satisfied in terms equality and so also the corresponding dual constraints become equality. Through the 3x3 linear system obtained we can compute a value for (y_1, y_2, y_3) in order to investigate if other possible patterns have to be considered

```
In [6]: A = Matrix([[3,1,1],[2,0,2],[0,2,1]])
        b = vector([1, 1, 1])
        x = A.solve_right(b)
        print (x)

(1/10, 3/10, 2/5)
```

From the linear system we obtain a solution that can be translated into the question: does it exist an admissimible cutting patern with

$$\frac{1}{10}\alpha + \frac{3}{10}\beta + \frac{2}{5}\gamma > 1$$

For example for [1,3,0], the previous condition becomes 1=1....

```
In [21]: p=MixedIntegerLinearProgram( maximization=False )
         p.set_objective( p[1]+p[2]+p[3]+p[4] )
         p.add_constraint( 3*p[1]+2*p[2]+p[4], min = 7 )
         p.add_constraint( p[1]+2*p[3]+3*p[4], min = 9 )
         p.add_constraint( p[1]+2*p[2]+p[3], min = 11 )
         p.solve()
         p.show()
         print('Objective Value: {}'.format(p.solve()))
         print ("The optimal values are x_1 = "+str(p.get_values(p[1]))+", x_2 = "
                +str(p.get_values(p[2]))+", x_3 = "+str(p.get_values(p[3]))+
                ", x_4 = "+str(p.get_values(p[4])))

Minimization:
  x_0 + x_1 + x_2 + x_3
```

```
Constraints:
  7.0 <= 3.0 x_0 + 2.0 x_1 + x_3
  9.0 <= x_0 + 2.0 x_2 + 3.0 x_3
  11.0 <= x_0 + 2.0 x_1 + x_2
Variables:
  x_0 is a continuous variable (min=-oo, max=+oo)
  x_1 is a continuous variable (min=-oo, max=+oo)
  x_2 is a continuous variable (min=-oo, max=+oo)
  x_3 is a continuous variable (min=-oo, max=+oo)
Objective Value: 7.800000000000001
The optimal values are x_1 = 0.19999999999999987, x_2 = 3.2, x_3 = 4.4, x_4 = 0.0
```

A possibile solution for the integer formulation can be formulated $1\text{x}[3,1,1]$, $3\text{x}[2,0,2]$ and $4\text{x}[0,2,1]$ that imply the use of 8 initial bars.

1.4.3 Example 2

Inital stock lenght W=30

Item	Lenght	Orders
a	4	40
b	7	30
c	8	25
d	11	20

Formulation of the primal minimization problem where Λ_i represent the number of cuts for each pattern i.

$$\min \sum \Lambda_i$$

As first guess consider pattern [1,1,1,1] with multiplicity $\Lambda_{[1,1,1,1]} = 40$ enough to have the first order filled.

Constraints:

$$40 \cdot \Lambda_{[1,1,1,1]} = 40$$
$$40 \cdot \Lambda_{[1,1,1,1]} \geq 30$$
$$40 \cdot \Lambda_{[1,1,1,1]} \geq 25$$
$$40 \cdot \Lambda_{[1,1,1,1]} \geq 20$$

Also its dual formulation is necessary

$$\max 40y_1 + 30y_2 + 25y_3 + 20y_4$$

$$y_1 + y_2 + y_3 + y_4 \leq 1$$

Since second, third and fourth inequalities are true in strict sense while the first inequality is achieved in terms of equality, the correspondig variable y_1 is equal to one by the dual constraint

written in terms of equality. According to this we can write the condition for a new hypotetic pattern $[\alpha, \beta, \gamma, \delta]$ requiring $\alpha > 1$. A possible pattern is for example [3,1,0,1] and with this choice the solution can be selected as

$$\Lambda_{[1,1,1,1]} = 25$$

$$\Lambda_{[3,1,0,1]} = 5$$

And the process starts iterating: the only inequality that holds strictly is the fourth that implies $y_4 = 0$.

So the related knapsack problem to solve can be written as

$$\max 40y_1 + 30y_2 + 25y_3$$

subject to

$$y_1 + y_2 + y_3 \leq 1$$

$$3y_1 + y_2 \leq 1$$

```
In [24]: p=MixedIntegerLinearProgram( maximization=True )
         p.set_objective( 40*p[1]+30*p[2]+25*p[3])
         p.add_constraint( p[1]+p[2]+p[3]<= 1 )
         p.add_constraint( 3*p[1]+p[2] <= 1 )
         p.solve()
         p.show()
         print('Objective Value: {}'.format(p.solve()))
         print ("The optimal values are x_1 = "+str(p.get_values(p[1]))+", x_2 = "
                +str(p.get_values(p[2]))+", x_3 = "+str(p.get_values(p[3])))
```

```
Maximization:
  40.0 x_0 + 30.0 x_1 + 25.0 x_2

Constraints:
  x_0 + x_1 + x_2 <= 1.0
  3.0 x_0 + x_1 <= 1.0
Variables:
  x_0 is a continuous variable (min=-oo, max=+oo)
  x_1 is a continuous variable (min=-oo, max=+oo)
  x_2 is a continuous variable (min=-oo, max=+oo)
Objective Value: 30.0
The optimal values are x_1 = 0.3333333333333333, x_2 = 0.0, x_3 = 0.6666666666666667
```

That has solution

$$(y_1, y_2, y_3) = \left(\frac{1}{3}, 0, \frac{2}{3}\right)$$

and the conditon for the next cutting pattern becomes:

$$\frac{1}{3}\alpha + \frac{2}{3}\gamma > 1$$

But also $(0,1,0,0)$ and $(\frac{1}{5},\frac{2}{5},\frac{2}{5},0)$ with the related conditions

$$\beta > 1$$

$$\frac{1}{5}\alpha + \frac{2}{5}\beta + \frac{2}{5}\gamma > 1$$

that suggest to choose [2,2,1,0] to advance with the algorithm.

The possible cutting-pattern are [1,1,1,1],[2,2,1,0] and [3,1,0,1] and we solve the LP minimization problem to compute how many times they occur.

$$\min \Lambda_{[1,1,1,1]} + \Lambda_{[2,2,1,0]} + \Lambda_{[3,1,0,1]}$$

subject to

$$\begin{aligned}
\Lambda_{[1,1,1,1]} + 2\Lambda_{[2,2,1,0]} + 3\Lambda_{[3,1,0,1]} &\geq 40 \\
\Lambda_{[1,1,1,1]} + 2\Lambda_{[2,2,1,0]} + \Lambda_{[3,1,0,1]} &\geq 30 \\
\Lambda_{[1,1,1,1]} + \Lambda_{[2,2,1,0]} + &\geq 25 \\
\Lambda_{[1,1,1,1]} + \quad + \Lambda_{[3,1,0,1]} &\geq 20
\end{aligned}$$

```
In [28]: p=MixedIntegerLinearProgram( maximization=False )
         p.set_objective( p[1]+p[2]+p[3])
         p.add_constraint( p[1]+2*p[2]+3*p[3]>=40 )
         p.add_constraint( p[1]+2*p[2]+p[3]>=30 )
         p.add_constraint( p[1]+p[2]>=25 )
         p.add_constraint( p[1]+p[3]>= 20 )
         p.solve()
         p.show()
         print('Objective Value: {}'.format(p.solve()))
         print ("The optimal values are x_1 = "+str(p.get_values(p[1]))+", x_2 = "
                +str(p.get_values(p[2]))+", x_3 = "+str(p.get_values(p[3])))

Minimization:
  x_0 + x_1 + x_2

Constraints:
  - x_0 - 2.0 x_1 - 3.0 x_2 <= -40.0
  - x_0 - 2.0 x_1 - x_2 <= -30.0
  - x_0 - x_1 <= -25.0
  - x_0 - x_2 <= -20.0
Variables:
  x_0 is a continuous variable (min=-oo, max=+oo)
  x_1 is a continuous variable (min=-oo, max=+oo)
  x_2 is a continuous variable (min=-oo, max=+oo)
Objective Value: 27.5
The optimal values are x_1 = 17.5, x_2 = 7.5, x_3 = 2.5
```

The optimal values for each pattern are

$$\left(\Lambda_{[1,1,1,1]}, \Lambda_{[2,2,1,0]}, \Lambda_{[3,1,0,1]}\right) = \left(\frac{35}{2}, \frac{15}{2}, \frac{5}{2}\right)$$

. Evaluating these values in the constraints of the primal problem it occurs that the only inequality strictly holding is the second one $\Lambda_{[1,1,1,1]} + 2\Lambda_{[2,2,1,0]} + \Lambda_{[3,1,0,1]} \geq 30$ i.e. $35 > 30$. This implies that the corrispettive dual variable y_2 has to be equal to 0. Therefore we write the dual LP problem that is expressed in maximization terms:

$$\max 40y_1 + 30y_2 + 25y_3 + 20y_4$$

$$\begin{aligned} y_1 + y_2 + y_3 + y_4 &\leq 1 \\ 2y_1 + 2y_2 + y_3 + \quad &\leq 1 \\ 3y_1 + y_2 + \quad + y_4 &\leq 1 \end{aligned}$$

```
In [37]: A = Matrix([[1,1,1],[2,1,0],[3,0,1]])
         b = vector([1, 1, 1])
         x = A.solve_right(b)
         print (x)
```

```
(1/4, 1/2, 1/4)
```

So a feasible solution optimizing the dual problem is

$$\left(y_1, y_2, y_3, y_4\right) = \left(\frac{1}{4}, 0, \frac{1}{2}, \frac{1}{4}\right)$$

that implies

$$\frac{1}{4}\alpha + \frac{1}{2}\gamma + \frac{1}{4}\delta \geq 1$$

We add to the previous pattern $[1,1,1,1]$,$[2,2,1,0]$,$[3,1,0,1]$ the one $[5,0,1,0]$ chosen to enforce the last condition and we solve the related LP relaxation.

```
In [38]: p=MixedIntegerLinearProgram( maximization=False )
         p.set_objective( p[1]+p[2]+p[3]+p[4])
         p.add_constraint( p[1]+2*p[2]+3*p[3]+5*p[4], min = 40 )
         p.add_constraint( p[1]+2*p[2]+p[3], min = 30 )
         p.add_constraint( p[1]+p[2]+p[4], min = 25 )
         p.add_constraint( p[1]+p[3], min = 20 )
         p.solve()
         print('Objective Value: {}'.format(p.solve()))
         print ("The optimal values are x_1 = "+str(p.get_values(p[1]))+", x_2 = "
                +str(p.get_values(p[2]))+", x_3 = "+str(p.get_values(p[3]))+
                ", x_4 = "+str(p.get_values(p[4])))
```

```
Objective Value: 26.42857142857143
The optimal values are x_1 = 18.57142857142857, x_2 = 5.0000000000000036, x_3 = 1.428571428571
```

```
In [20]: print(18.57142857142857+2*5+3*1.42857+5*1.42857)
         print(18.57142857142857+2*5+1.42857)
         print(18.57142857142857+5+1.42857)
         print(18.57142857142857+1.42857)
```

```
39.9999885714286
29.9999985714286
24.9999985714286
19.9999985714286
```

The dual problem has the following structure

$$\max 40y_1 + 30y_2 + 25y_3 + 20y_4$$

$$\begin{aligned} y_1 + y_2 + y_3 + y_4 &\leq 1 \\ 2y_1 + 2y_2 + y_3 + \quad &\leq 1 \\ 3y_1 + y_2 + \quad + y_4 &\leq 1 \\ 5y_1 + \quad + y_3 + \quad &\leq 1 \end{aligned}$$

```
In [25]: A = Matrix([[1,1,1,1],[2,2,1,0],[3,1,0,1],[5,0,1,0]])
         b = vector([1, 1,1, 1])
         x = A.solve_right(b)
         print (x)
```

```
(1/7, 3/14, 2/7, 5/14)
```

That is traduces into

$$\frac{1}{7}\alpha + \frac{3}{14}\beta + \frac{2}{7}\gamma + \frac{5}{14}\delta$$

that suggests to add the pattern $[0, 2, 2, 0]$ to the current solution.

```
In [41]: p=MixedIntegerLinearProgram( maximization=False )
         p.set_objective( p[1]+p[2]+p[3]+p[4]+p[5])
         p.add_constraint( p[1]+2*p[2]+3*p[3]+5*p[4], min = 40 )
         p.add_constraint( p[1]+2*p[2]+p[3]+2*p[5], min = 30 )
         p.add_constraint( p[1]+p[2]+p[4]+2*p[5], min = 25 )
         p.add_constraint( p[1]+p[3], min = 20 )
         p.solve()
         print('Objective Value: {}'.format(p.solve()))
         print ("The optimal values are x_1 = "+str(p.get_values(p[1]))+", x_2 = "
                +str(p.get_values(p[2]))+", x_3 = "+str(p.get_values(p[3]))+", x_4 = "
                +str(p.get_values(p[4]))+", x_5 = "+str(p.get_values(p[5])))
```

```
Objective Value: 26.428571428571427
The optimal values are x_1 = 0.0, x_2 = -13.571428571428571, x_3 = 20.0, x_4 = 1.4285714285714
```

```
In [42]: p=MixedIntegerLinearProgram( maximization=False )
         w = p.new_variable(integer=False, nonnegative=True)
         p.set_objective( w[1]+w[2]+w[3]+w[4]+w[5])
         p.add_constraint( w[1]+2*w[2]+3*w[3]+5*w[4], min = 40 )
         p.add_constraint( w[1]+2*w[2]+w[3]+2*w[5], min = 30 )
         p.add_constraint( w[1]+w[2]+w[4]+2*w[5], min = 25 )
         p.add_constraint( w[1]+w[3], min = 20 )
         p.solve()
         print('Objective Value: {}'.format(p.solve()))
         print ("The optimal values are x_1 = "+str(p.get_values(w[1]))+", x_2 = "
                +str(p.get_values(w[2]))+", x_3 = "+str(p.get_values(w[3]))+
                ", x_4 = "+str(p.get_values(w[4]))+", x_5 = "+str(p.get_values(w[5])))

Objective Value: 26.42857142857143
The optimal values are x_1 = 13.57142857142857, x_2 = 0.0, x_3 = 6.428571428571429, x_4 = 1.428
```

```
In [44]: A = Matrix([[1,1,1,1],[2,2,1,0],[3,1,0,1],[5,0,1,0],[0,2,2,0]])
         b = vector([1,1,1,1, 1])
         x = A.solve_right(b)
         print (x)

(1/7, 3/14, 2/7, 5/14)
```

Since the objective function of the primal is not improved and also the condition retrieved from the dual does not change from the previous iteration, we are sure that the greedy algorithm can be stopped and that we are dealing with the optimal solution.