

Fast Iterations for the Combined Cutting-Stock and Lot-Sizing Problems

Gláucia Maria Bressan (USP) galbressan@hotmail.com
 Aurélio Ribeiro Leite de Oliveira (UNICAMP) aurelio@ime.unicamp.br

Abstract

In this work the combined problem, which connects the lot sizing and the cutting stock problems, is considered. The properties of the matrix of constraints are studied and a factorization update of the bases without losing sparsity in the simplex method context, by a static reordering of the basic columns is proposed. Numerical results of an MATLAB implementation simulate simplex iterations and verify the sparsity of the factorizations are presented. These experiments had also proven the robustness of this strategy. The approach that performs the static sparse basis reordering leads to very good computational results for both: speed and robustness.

Key words: linear programming - cutting stock - lot sizing.

1. Introduction

In this work, the linear systems originated from the simplex method applied to the combined cutting stock and lot sizing problems are solved in an efficient way, due to the sparse matrix pattern exploitation. This problem has a block angular matrix structure. The matrix blocks are very sparse indicating that this approach would lead to good computational results.

1.1 The Cutting-Stock Problem

Consider that several bobbins are available for cutting to produce the units who will compose the products. The goal is to determine the amount of bobbins to be used cut and the pattern of the cuts. It is assumed that there is a large stock of bobbins of a certain number of types.

Since there are an exponential number of cutting patterns, leading to a huge matrix of constraints, the simplex method with column generation is adopted. Thus, a subproblem is solved in order to determine the cutting pattern with the best relative cost to enter the simplex basis and it is not necessary to store the whole matrix.

Consider the problem with only bobbins of size L to be cut into pieces of size $l_i, i = 1, \dots, P$. The columns of the matrix of constraints A are given by: $X = \{a = (\alpha_1, \alpha_2, \dots, \alpha_P) \mid l_1\alpha_1 + l_2\alpha_2 + \dots + l_P\alpha_P \leq L, \alpha_i \geq 0 \text{ integer}\}$.

Some of those columns are easily build by the *homogeneous patterns*, that is, the patterns that produce only one type of piece. The P vectors associated to those patterns are given by $X : a_i = (0, \dots, a_{ii}, \dots, 0), i = 1, \dots, P$, where $a_{ii} = \lfloor L/l_i \rfloor$, and $\lfloor x \rfloor$ is the floor of x . Therefore, these columns give rise to a diagonal submatrix (Arenales et al 1997):

$$D = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{PP} \end{bmatrix}.$$

For the unidimensional cutting-stock problem, the subproblems reduce to one knapsack problem to determine the column with smallest relative cost.

2. The Combined Problem

The production process can be divided in three stages: the first one determines the demand for a finite planning horizon, giving the amount of products and the deadline for delivery. The second stage computes the number of pieces to be cut according to the amount of products demanded. Thus, a given piece could be used to assemble different product. Finally, in the third stage the amount of products to be assembled in each period of time is computed in order to minimize cost of stock and loss in the cutting process.

The loss in the cutting process tends to decrease with the rising of the number of products demanded. Therefore, from one side is better to cut more pieces to minimize this loss. On the other hand, stock cost press to delay the cutting process. Thus, the combined problem (Nonas et al 2000), which merges the lot sizing and de cutting stock problems, is a natural development to minimize global costs.

The lot-sizing problem search for solutions that assembles the amount of products demanded for each time interval in the planning horizon and minimizes production and stock costs. This problem can be decomposed in subproblems with only one product and can be solved by dinamic programming (Wagner and Whitin, 1958). The cutting-stock problem is usually solved by the simplex method with column generation (Gilmore and Gomory, 1965).

In practical situations these two problems are approached separately due to the complexity of the combined problem. However, this approach could lead to bad solutions in term of costs, mainly if the bobbins to be cut are expensive.

2.1 Mathematical Formulation

In order to simplify the presentation consider that there is only one type of bobbin of size $L \times W$, in a large number, enough to supply the demand. The model can be defined as follows:

$$\min \sum_{i=1}^M \sum_{t=1}^T (c_{it} x_{it} + h_{it} e_{it}) + \sum_{j=1}^N \sum_{t=1}^T cpy_{jt} + \sum_{p=1}^P \sum_{t=1}^T hp_{pt} ep_{pt}$$

$$\text{s.to: } x_{it} + e_{i,t-1} - e_{it} = d_{it}$$

$$\sum_{j=1}^N a_{pj} y_{jt} + ep_{p,t-1} - ep_{pt} = \sum_{i=1}^M r_{pi} x_{it} \quad \forall t = 1 \dots T$$

$$\sum_{j=1}^N v_j y_{jt} \leq u_t$$

$$x_{it}, e_{it}, y_{jt}, ep_{pt} \geq 0.$$

Indices:

$t=1, \dots, T$ number of time intervals.

$p=1, \dots, P$ number of different pieces to be cut.

$j=1, \dots, N$ number of cutting patterns.

$i=1, \dots, M$ number of products assembled by the pieces.

Parameters:

c_{it} : Product i manufacturing cost in time interval t .

h_{it} : Product i storage cost in time interval t .

hp_{pt} : Type p piece storage cost in time interval t .

d_{it} : Product i demand in time interval t .

r_{pi} : Number of type p piece to assemble product i .

v_j : Time needed to cut a bobbin for the cutting pattern j .

a_{pj} : Type p pieces in the cutting pattern j .

u_t : Maximum time of machine operation.

cp : Bobbin cost.

Variables:

x_{it} : Product i amount assembled in time interval t .

e_{it} : Product i amount stored in time interval t .

ep_{pt} : Type p pieces amount stored in time interval t .

y_{jt} : Amount of bobbins cut according to pattern j in time interval t .

The model is simplified since setup time (Trigeiro et al 1989) is not considered and the integrality constraints are relaxed, with is not an important issue for a large amount of bobbins, that is, a large demand of products. Since this work is concerned with the LU factorization of a simplex basis, the integrality of variables is of no importance either

The model obtained is a linear programming problem, however, it is still a difficulty one since there is a large amount of cutting patterns.

The model can be rewritten in the following matrix notation:

$$\min \sum_{t=1}^T (c_t x_t + h_t e_t) + \sum_{t=1}^T c_p y_t + \sum_{t=1}^T h_p e_p$$

$$\text{s.to: } x_t + e_{t-1} - e_t = d_t$$

$$-R x_t + A y_t + e_{p,t-1} - e_{p,t} = 0 \quad \forall t = 1 \dots T$$

$$v^T y_t \leq u_t$$

$$x_t, e_t, y_t, e_{p,t} \geq 0$$

Where $e_{p,0}, e_0$ are known, $c_t = (c_{1t}, c_{2t}, \dots, c_{Mt})$; and the remaining parameters and variables $h_t, e_t, hp_t, ep_t, d_t, x_t$ and y_t are defined in a similar way, $v^T = (v_1 v_2 \dots v_N)$, A is a $P \times N$ matrix where each column represents a cutting pattern and

$$R = \begin{bmatrix} r_{11} & r_{12} & \dots & r_{1M} \\ r_{21} & r_{22} & \dots & r_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ r_{P1} & r_{P2} & \dots & r_{PM} \end{bmatrix}.$$

3. The Static Reordering

The matrix of constraints can be reordered to present block diagonal pattern. This ordering consists in permuting the columns related to variables e_t and x_t . This reordering is meant to reduce de amount of fill-in during the LU factorization of the simplex basis. Furthermore, the slack variables $f_t = u_t - v^T y_t$ for $t=1...T$ are added as shown in Table 1. The slack variables could represent artificial variables in a phase one procedure of the simplex if necessary.

Since this reordering is static, it is possible to obtain sparse factorization of any simplex basis with no computational effort to the column permutation: the column ordering in the basis follows the column ordering for the constraint matrix. Therefore, the static reordering has no cost for either, initialization or updating.

The cutting pattern columns are not reordered a priori since they are generated as need during the simplex iterations and are not even known. It is a trivial task, however, to find the correct position of those columns when entering the basis in the appropriate time interval. In practice these columns are located for each time interval according with the respective number of nonzero entries.

The matrix R , which represents the demand of pieces, has also an certain degree of sparsity since each of its columns represents a product that does not necessarily is assembled by all of the pieces types. Therefore, the variables x_{it} can be reordered a priori according to the sparse pattern of R considering the number of nonzero of R columns. This reordering is equivalent to enumerate the demanded products in a convenient way.

e_1	e_2	...	e_T	x_1	x_2	...	x_T	y_1	...	y_T	ep_1	...	ep_T	f_1	...	f_T	d_T
-I	0	...	0	I	0	...	0	0	...	0	0	...	0	0	...	0	$d_1 - e_0$
I	-I	...	0	0	I	...	0	0	...	0	0	...	0	0	...	0	d_2
	
0	0	...	0	0	0	...	0	0	...	0	0	...	0	0	...	0	d_{t-1}
0	0	...	0	0	0	...	0	0	...	0	0	...	0	0	...	0	d_t
	
0	0	...	-I	0	0	...	I	0	...	0	0	...	0	0	...	0	d_T
0		...	0	-R	0	...	0	A	...	0	-I	...	0	0	...	0	0
0		...	0	0	-R	...	0	0	...	0	I	...	0	0	...	0	0
			0
0		...	0	0	0	...	-R	0	...	A	0	...	-I	0	...	0	0
0		...	0	0	0	...	0	v^T	...	0	0	...	0	-1	...	0	u_1
0		...	0	0	0	...	0	0	...	0	0	...	0	0	...	0	u_2
	
0		...	0	0	0	...	0	0	...	v^T	0	...	0	0	...	-1	u_T

Table 1: Matrix of Constraints Static Reordering for the Combined Problem

The advantage of this reordering comes from the fact that the columns of the basis will also exhibit a block diagonal pattern leading to sparse LU factorizations. Another and more important advantage is that the factorization update after a simplex iteration turns out to be inexpensive and robust, as it will be shown in the Numerical Experiments Section.

3.1 Starting Basis

A starting basis can be obtained considering that there are no pieces and products in stock and using the homogeneous cutting patterns. Such a basis has the pattern shown in Table 2.

x_1	x_2	...	x_T	y_1	y_2	...	y_T	f
$I_{M \times M}$	0	...	0	0	0	...	0	0
0	$I_{M \times M}$...	0	0	0	...	0	0
			
0	0	...	$I_{M \times M}$	0	0	...	0	0
$-R_{P \times M}$	0	...	0	$D_{P \times P}$	0	...	0	0
0	$-R_{P \times M}$...	0	0	$D_{P \times P}$...	0	0
			
0	0	...	$-R_{P \times M}$	0	0	...	$D_{P \times P}$	0
0	0	...	0	v_B^T	0	...	0	
0	0	...	0	0	v_B^T	...	0	-1
0	0	...	0	0	0	...	v_B^T	

Table2: Starting Basis

Matrix D is a diagonal submatrix of A with $P \times P$ dimension and $v_{B_{i \times P}}$ represents a vector whose columns correspond to the columns of D . Notice that this basis is a triangular matrix and, therefore, it is not necessary to factorize it.

3.2 Column Identification

Given the entering and leaving columns, it is easy to identify the type of variable that these columns represent. The position of the first k_1 and last k_2 nonzero entry of the columns can be used for this purpose. The following algorithm shows how it can be done:

```

if  $K_1 \leq TM$ 
    then if  $K_2 > TM$ 
        then  $c \in$  product variable
        else  $c \in$  product stock variable
    else if  $K_1 > T(P+M)$ 
        then  $c \in$  slack variable
    else if  $K_2 > T(P+M)$ 
        then  $c \in$  cutting pattern
        else  $c \in$  piece stock variable.
    
```

4. Numerical Experiments

The goal of these experiments is to verify the potential use of the static reordering. The code is implemented using MATLAB 5.3, in pentium 4 Intel CPU with 1.8GHz and 512MB RAM.

The following parameters are adopted for the test problem: the number of time intervals is set to 6. There are 2 products and 5 types of pieces. The number of cutting patterns is set to 60. For those parameters, the matrix of constraints has dimension 48×420 , where 360 of the columns correspond to cutting patters.

The simplex iterations are simulated choosing the leaving and the entering columns randomly. After choosing the leaving column, the entering one is randomly chosen provided it leads to a nonsingular basis, otherwise another column is also randomly chosen until a nonsingular basis is obtained.

Three sparse versions of the delayed LU factorization are used in the code and the number of flops need for each one compared. The first one called $lu(B)$ its equivalent to the internal MATLAB command $lu()$ and computes the LU factorization of the basis from scratch without using information from the previous factorization. The second version, called $min(e,l)$ the factorization is performed from either the position of the entering or the leaving variable, whichever with the smaller index. Since the previous columns remained unchanged, the factorization of them can be reused for consecutive iterations.

The third version named $dec.sparse(B)$ is somewhat more subtle. Only the columns whose factorization is really affected by the change of columns are involved in the factorization update.

For this version, if $l < e$, the columns factorization affected by the leaving columns are considered. Those columns with index greater than l and smaller than e , but whose nonsparse pattern does not coincide with the leaving column are not affected and its factorization remains unchanged. For the columns affected the operations related to the factorization from the leaving column are undone in the reverse order always considering sparsity.

If $e < l$, ie, if the entering column has the smaller index, the factorization step related to the entering column is performed for columns between e and l considering the sparse pattern. Finally, for columns with index greater then $max(e,l)$, both steps are performed.

Table 3 shows a comparison for the number of flops to obtain the LU factorization update among the three versions. The number of flops shown is the minimum, average and maximum after 500 iterations. Those results do not change in a significant way from 500 to 10000 iterations.

flops	$lu(B)$	$min(e,l)$	$dec.sparse(B)$
min	1241	37	7
max	1477	391	71
average	1352.9	185.1	33.7

Table3: Flops comparison for LU update

The LU factorization update reduces about 97% the number of flops in comparison to computing a factorization from scratch. It also obtains good performance in comparison with a more naive update.

4.1 Robustness of the Updating Scheme

The following experiments show that the factorization update proposed is robust. In these tests, the whole factorization is undone for all simplex iterations, thus, obtaining the original basis with some amount of numerical error. The next basis is obtained from this one together with the entering variable. This is a worst case scenario since at each iteration, the basis is factored and then the factorization is undone and the computed basis is used for the next simplex iteration.

In order to measure the error introduced for these operations, the norm of the computed basis is compared with the norm from the original basis obtained from the columns of the constraints matrix. The product $L*U$ is also computed, given yet another approximation for the basis.

The average and maximum error obtained for iterations 100, 1000, 2000 and 10000 are shown in Table 4. The first two rows contain the error for the undo operation and the last two for the $L*U$ product.

error	100	1000	2000	10000
max(undo)	6.8 e-13	1.3 e-12	1.8 e-12	4.0 e-12
average(undo)	1.6 e-13	6.1 e-13	5.8 e-13	7.0 e-13
max(L*U)	1.1298 e-12	2.2397 e-12	2.8235 e-12	3.6005 e-12
average(L*U)	4.2917 e-13	1.1003 e-12	9.9500 e-13	1.1265 e-12

Table 4: Error estimation for the updated LU factorization

It can be concluded that the factorization update proposed method is very robust since after 10000 iterations the accumulated error is in the worst case in the order of 10^{-12} . This result is still more significant considering that the error for the $L*U$ product is in average a little bit larger than the error for the undo operation. These results mean that it is not necessary at all to refactorize the basis from scratch at any time as it is usually done due to the robustness of the proposed approach.

5. Conclusions

In this work a static reordering of the columns of the constraint matrix is proposed, leading to simplex block diagonal basis with sparse factorizations. The static reordering allows the implementation of a fast factorization update that showed to be very robust.

The static reordering and factorization update present the following advantages:

- The static reordering leads to sparse bases factorizations and to inexpensive factorization updates.
- The reordering has no initialization or updating costs since there is no need to reorder the columns in the factorization.
- This reordering can be easily integrated to other implementations. The algorithm that identifies the type of columns plays an important role in this integration. One option in consideration is the software GLPK - *Gnu Linear Programming Kit*, which is an open code.
- Table 4 presents stable results and it is safe to conclude that no periodical factorization is needed as it is usually suggested for updating schemes (Bartels, 1969).

- The previous conclusion, together with a triangular starting basis means that no complete LU factorization is ever performed for this approach. Only partial updates are need for all iterations.

Acknowledgments

This research was partially sponsored by the Foundation for the Support of Research of the State of São Paulo, (FAPESP) and the Brazilian Council for the Development of Science and Technology (CNPq).

References

- ARENALES, M.N, MORÁBITO, R. and YANASSE (1997) - H. *O problema de Corte e Empacotamento e Aplicações Industriais*. XX Congresso Nacional de Matemática Aplicada e Computacional, Gramado RS, 8 a 12 de setembro, in Portuguese.
- BARTELS R.H. (1969) - *A Stabilization of the Simplex Method*, Numerical Mathematics, 16:414-434.
- GILMORE, P. and GOMORY, R. (1965) - *Multistage cutting stock problems of two and more dimensions*. Operation Research, 14:1045-1074.
- NONAS, S.L. and THORSTENSON (2000) - A. *A combined cutting-stock and lot-sizing problem*. Operation Research, 120(2):327-342.
- TRIGEIRO, W. THOMAS, L.J. and MCCLAIN (1989) - J.O. *Capacited lot sizing with setup times*, Management Science, 35(3):353-366.
- WAGNER, H.M. and WHITIN, T.M. (1958) - *Dynamic version of the Economic Lot size Model*. Management Science, 5(1):89-96.