A new Simplex method updating procedure applied to linear systems arising from the cutting stock problem

Daniela R. Cantane
Department of Bio-statistics, Institute of Bio-Sciences, UNESP, Distrito de Rubião Júnior, S/N CEP: 18618-970, Botucatu, SP, Brazil.
E-mail: dcantane@ibb.unesp.br

Carla L. T. S. Ghidini
Institute of Mathematics, Statistics and Scientific of Computation, UNICAMP, Sérgio Buarque de Holanda, 651, CEP:13081-970, Campinas, SP, Brazil.
E-mail: carla@ime.unicamp.br

Aurelio R. L. Oliveira
Institute of Mathematics, Statistics and Scientific of Computation, UNICAMP, Sérgio Buarque de Holanda, 651, CEP:13081-970, Campinas, SP, Brazil.
E-mail: aurelio@ime.unicamp.br

Christiano Lyra
Electric Engineering and Computation School, UNICAMP, Av. Albert Einsten, 400, CEP:13083-852, Campinas, SP, Brazil.
E-mail: chrlyra@densis.fee.unicamp.br

Abstract

The objective of this work is to apply efficient Simplex-basis LU factorization update techniques to improve the column updating performance of the Simplex method, when used to solve cutting stock problems. Only the factored columns actually modified by the change in the basis are considered, to obtain an efficient LU factorization update. The matrix columns are reordered according to a specific given strategy. Thus, sparse factorizations are obtained for any basis without computational effort, to obtain the order of the columns, because the reordering of the matrix is static, and the basis columns follow this ordering. The proposed method achieves a reduction in the computational time for the solution of the cutting stock problems when compared with the GLPK.

Keywords: sparse matrix, LU factorization update, Simplex method, cutting stock problem.
Main area (Mathematical Programming).

1 Introduction

The cutting stock problem is applicable to various types of industries, including paper, furniture, textiles, and steel. The decisions concerning these problems interact with other decisions that are relevant to production planning, and, thus, determining the optimal solution to the cutting stock problem is important.

The cutting stock problem consists of cutting standard objects that are available in stock into smaller items to meet a known demand, by optimizing a given objective function, e.g., by minimizing the total number of objects in stock during each period or by minimizing the waste of raw material. This problem can be formulated as an integer linear optimization problem. The relevance of this type of approach has been demonstrated by the popularity of the pioneering works of Gilmore and Gomory [5, 6] as the subject of many research studies, in which techniques were proposed to obtain a continuous optimal solution to the relaxed problem.

The aim of this paper is not to study the cutting stock problem in detail or to propose a new method for its solution, which can be found in Poldi and Arenales [12]. The main goal is to present innovations that improve the performance of the proposed methods in the literature. Specifically, we present a new technique of the **LU** factorization update that can be used to solve the linear systems that arise at each iteration of the Simplex method. The Simplex method with column generation, thus, provides a continuous solution to the cutting stock problem.

The efficient solution of large-scale linear systems is important for solving linear optimization problems. These systems can be approached through generic methods, for example, the **LU** factorization of the basis and its update, or through the problem specific structure exploitation, as in the network flow problem in Kennington and Helgason [9].

In this study a static reordering of the basis columns of the cutting stock problem using the Simplex method with column generation is developed together with an efficient update of the entering and leaving columns of the basis. The **LU** factorization is performed only in the columns that are actually modified by the entering and/or leaving columns in the basis.

The newly proposed update presents a completely different approach when compared with the traditional methods of Duff et al., and Maros [3, 11]. In the **LU** factorization update, the operations of the column that leaves the basis are undone; in other words, the operations are performed in the reverse order of the factorization. The operations that factor the basic matrix, which are obtained with the entering column, are then performed. Finally, it is necessary to perform the operations on the columns that are located after the entering column in the basis. Such operations should always consider the sparse pattern of the basis.

The remainder of this paper is organized as follows. In Section 2, we present the cutting stock problem. Section 3 describes the existing **LU** factorization update methods, and Section 4 describes the proposed **LU** factorization update method. Implementation issues and computational experiments are discussed in Sections 5 and 6, respectively. Finally, conclusions are given in Section 7.

2 One-Dimensional Cutting Stock Problem

2.1 Definition and Mathematical Model

The one-dimensional cutting stock problem can be stated as follows. Assume that there is available, in stock, a sufficiently large number of objects (e.g., bars, reels) of length L_k with the quantity e_k, $k = 1, \ldots, K$. Furthermore, we have a set of m smaller ordered items of length l_i, that must be produced in quantities d_i, $i = 1, \ldots, m$. The problem consists of producing the items by cutting the available stock pieces to meet the demand, optimizing an objective function such as minimizing the total number of pieces to be cut, minimizing the waste of raw material, or maximizing the profit.

The mathematical modeling of the cutting stock problem involves two steps: defining the cutting pattern for the stock objects and deciding how often each cutting pattern will be used to meet the demand. The first step can be performed independently of the demand for the items.

The specific method of cutting a stock length into items is termed the cutting pattern. Let N_k be the number of cutting patterns for the stock length type k, and let α_{ijk} be the number of types of items i in the j^{th} cutting pattern for a stock length of type k. A vector $a_{jk} = (\alpha_{1jk}, \alpha_{2jk}, \ldots, \alpha_{mjk})$ represents the j^{th} cutting pattern for a stock of length type k, if it satisfies the following:

$$l_1\alpha_{1jk} + l_2\alpha_{2jk} + \ldots + l_m\alpha_{mjk} \leq L_k$$
$$\alpha_{ijk} \geq 0 \text{ and integer,}$$

where L_k is the length of the stock object type k.

The cutting stock problem with multiple stock lengths in limited quantities was proposed by Gilmore and Gomory [6] and can be formulated as follows:

Data:

- $k = 1, \ldots, K$: number of types (lengths) of the available objects;
- $j = 1, \ldots, N_k$: number of the cutting patterns of the object type k;
- $i = 1, \ldots, m$: number of types of items to be cut;
- m: number of types of items;
- d_i: demand for item type i;
- K: number of types of stock objects;
- e_k: availability of object type k.

Parameters:

- N_k: number of cutting patterns for stock object type k;
- α_{ijk}: number of items of type i cut in the j-th cutting pattern for object type k;
- c_{jk}: the amount of waste of raw material when cutting an object of type k according to the cutting pattern j.

Variables:

- x_{jk}: number of objects of type k there are cut according to cutting pattern j.

$$\text{minimize } \sum_{j=1}^{N_1} c_{j1}x_{j1} + \sum_{j=1}^{N_2} c_{j2}x_{j2} + \ldots + \sum_{j=1}^{N_K} c_{jK}x_{jK} \tag{2.1}$$

$$\text{subject to } \sum_{j=1}^{N_1} \alpha_{ij1}x_{j1} + \sum_{j=1}^{N_2} \alpha_{ij2}x_{j2} + \ldots + \sum_{j=1}^{N_K} \alpha_{ijK}x_{jK} = d_i \tag{2.2}$$

$$\sum_{j=1}^{N_k} x_{jk} \leq e_k \tag{2.3}$$

$$x_{jk} \geq 0, \text{ integer}, i = 1, \ldots, m, j = 1, \ldots, N_k, k = 1, \ldots, K. \tag{2.4}$$

In model (2.1)-(2.4), the objective function minimizes the total waste of material. The waste of raw material when cutting object type k according to the cutting pattern j is computed with the following form:

$$c_{jk} = L_k - \sum_{i=1}^{m} l_i\alpha_{ijk},$$

where l_i is the length of item type i, and L_k is the length of object k. The cost c_{jk} could also be the cost of the object type k (independent of the cutting pattern) and in this case $c_{jk} = c_k$.

The constraints (2.2) guarantee that the demand is met, i.e., that the quantities of the items that are produced are exactly equal to the demand. The constraint set (2.3) guarantees that the number of stock lengths cut of type k does not exceed the availability e_k. Finally, the last constraints (2.4) guarantee that the number of times that each pattern is cut is a non-negative integer.

We can rewrite the constraints (2.3) in the following form:

$$\sum_{j=1}^{N_k} x_{jk} + \mu_k = e_k, \tag{2.5}$$

$$\mu_k \geq 0, \tag{2.6}$$

where μ_k are the slack variables for the stock constraints (2.3).

The constraint matrix of the mathematical model has the following form:

$$\begin{bmatrix}
\alpha_{111} & \dots & \alpha_{1N_11} & \alpha_{112} & \dots & \alpha_{1N_22} & \dots & \alpha_{11K} & \dots & \alpha_{1N_KK} & 0 & 0 & \dots & 0 \\
\alpha_{211} & \dots & \alpha_{2N_11} & \alpha_{212} & \dots & \alpha_{2N_22} & \dots & \alpha_{21K} & \dots & \alpha_{2N_KK} & 0 & 0 & \dots & 0 \\
\vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \dots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\
\alpha_{m11} & \dots & \alpha_{mN_11} & \alpha_{m12} & \dots & \alpha_{mN_22} & \dots & \alpha_{m1K} & \dots & \alpha_{mN_KK} & 0 & 0 & \dots & 0 \\
1 & \dots & 1 & 0 & \dots & 0 & \dots & 0 & \dots & 0 & 1 & 0 & \dots & 0 \\
0 & \dots & 0 & 1 & \dots & 1 & \dots & 0 & \dots & 0 & 0 & 1 & \dots & 0 \\
\vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \dots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & \dots & 0 & 0 & \dots & 0 & \dots & 1 & \dots & 1 & 0 & 0 & \dots & 1
\end{bmatrix}$$

Each of the $\sum_{k=1}^{K} N_k$ first columns of the constraint matrix is an $(m+K)$-dimensional vector that is formed from two parts. The first m components correspond to the cutting patterns and the remaining K components are all null, except for component $m+k$, which is equal to one. The K remaining columns relative to the slack variables are also vectors of dimension $(m+K)$ with 1 in the position $m+k$ and zero in the other positions.

More details of this formulation can be found in Poldi and Arenales [12].

Remarks:

1. The cost c_{jk} could also be the cost of the object type k;
2. The integrality constraints in (2.4) are commonly found in real-world cutting problems, and they make the model (2.1)-(2.4) very difficult to solve exactly, even for medium-size instances. Thus, heuristic approaches are devised to find integer solutions. A class of residual heuristics is commonly used to solve this model. First, an optimal solution to the continuous relaxation is determined and, after other heuristics, used to determine the integer solutions. For details about residual heuristics see Poldi and Arenales [12], Wäscher and Gau [16], Holthaus [8], and Stadtler [14].

2.2 Simplex Method with Column to the One-Dimensional Cutting Stock Problem

Two factors contribute to the solution of the mathematical model (2.1)-(2.4) that become impractical for the one-dimensional cutting stock problem:

- the integrality constraint on the variables (how many times each cutting pattern will be used);
- the large number of variables (one variable for each cutting pattern). Practical problems can be on the order of hundreds of thousands.

This model presents a specific structure that allows us to work implicitly with these variables.

If the integrality constraint of the variables is relaxed, then the problem can be solved by the Simplex method developed by Dantzig in 1947, but the difficulty in relation to the large number of variables still exists. To address this difficulty, Gilmore and Gomory [5] propose a modification to the Simplex method that incorporates the column generation technique, as follows: in each iteration of the Simplex method, one of the steps is to look for a new column for entering in the basis to improve the value of the objective function. Rather than examining a large number of columns (all possible cutting patterns for each type of object) to determine the lower relative cost, a new column is generated by solving an auxiliary problem (subproblem). Whereas the criterion of Dantzig is used to determine the lowest relative cost, the subproblem to be solved for the object of type k is the following:

$$c_{rk} - \pi^T a_{rk} = min\{c_{jk} - \pi^T a_{jk}, j = 1, ..., N_K\}, \tag{2.7}$$

where π is the Simplex multiplier vector of a determined iteration.

Considering that the objective function to be minimized is the total number of cutting objects, when a column $a_k = (\alpha_{1k}\alpha_{2k}\ldots\alpha_{mk})^T$ matches a pattern for cutting the object k, equation (2.7) can be rewritten as follows:

$$g(a_k) = \text{maximize} \sum_{i=1}^{m} \pi_i \alpha_{ik} \tag{2.8}$$

$$\text{subject to} \sum_{i=1}^{m} l_i \alpha_{ik} \leq L_{ik}, \tag{2.9}$$

$$\alpha_{ik} \geq 0, \text{ integer.} \tag{2.10}$$

If the lesser of the relative costs is non-negative, when considering all of the types of objects, then the current solution of the relaxed problem is optimal; otherwise, a new column will enter at the base, and the procedure continues.

The auxiliary problem (2.8)-(2.10), which consists of cutting a single type of object in stock, is a knapsack problem, and an integer programming problem; therefore, implicit enumeration methods or dynamic programming can be used to solve it.

Remark: In the model described in (2.1)-(2.4), the homogeneous solution, which is usually used as an initial basic solution, may be not feasible, and in this case, phase I of the Simplex method must be applied.

Algorithm:

1. {**Phase I**}
Determine an initial basic matrix B.
Do: Stop = False and $it = 1$.

2. {**Phase II**}
While Stop = False, do:

2.1 Determine the actual basic solution: $By_B = d$.

2.2 Determine the dual solution: $B^T\pi = c_B$.

2.3 Solve the K knapsack problems in (2.8)-(2.10) (one for each type of object).
Find r such that $g(a_r) = min \ \{g(a_k), k = 1, ..., K\}$ and achieve the new column a_r.

2.4 {Optimality test:}
If $(1 - g(a_r)) \geq 0$ then Stop = True (the actual solution is optimal).
Otherwise, determine the basic coordinates of the Simplex direction:
$Bz = -a_r$.

2.5 {Determining the step size:}
Find l such that:

$$-\frac{y_l}{z_l} = min \ \ \{-\frac{y_{B_i}}{z_i}, z_i \geq 0, 1 = 1, ..., m\}$$

2.6 {Updating:}
Update the basis B, replacing column l by a_r, which is archived in 2.3.
Do: $it = it + 1$.

An integer solution to the original cutting stock problem can be determined from the optimal solution of the relaxed problem using heuristic procedures (e.g., constructive heuristics and residual heuristics) that were developed by several researchers in the area, such as Poldi and Arenales [12], Wäscher and Gau [16], Hinxman [7], and Stadtler [14], among others.

3 LU Factorization Update Methods

Much of the computational effort needed for the Simplex method with column generation in each iteration consists of solving the following linear systems:

- $By_B = d$ (primal basic solution);
- $B^T\pi = c_B$ (dual solution);
- $Bz = -a_r$ (Simplex direction).

Note that the three linear systems have the same coefficient matrix B or its transpose B^T. Thus, part of the computation that is performed for finding the solution of one system can be used for the solution of the other systems.

A method that is often used in software packages for linear optimization, to solve linear systems, is **LU** factorization, which is roughly equivalent to the Gauss elimination strategy, especially when applied to large and sparse linear systems.

Furthermore, in efficient implementations of the **LU** factorization method, it is important that good procedures be used to update from one iteration to another the **LU** factorization of the matrix B, because only one column of B is changed in each iteration. It is regarding this point that the present study make its greatest contribution. In the next section, we present the proposed method, in details, for performing the **LU** factorization update.

The **LU** factorization update technique with partial pivoting was proposed by Bartels and Golub [1]. Two variants of this algorithm, studied in Reid [13], aimed to balance the sparsity and numeric stability in the factorization. The latter variant is an improvement over the former.

In the literature several methods have been proposed to update the triangular factors of a modified matrix. Forrest-Tomlin's analysis [4] addresses large scale optimization problems and concludes that the method of Brayton et al. [2], appears to be the most convenient for practical implementations.

Maros [11] describes in detail the **LU** factorization update and its application to the basis update in the Simplex method proposed by Markowitz [10]. Moreover, Maros discusses the efficient implementation that was proposed by Suhl and Suhl [15], which is a good combination of the symbolic and numerical phases of the pivoting and presents a compromise between the sparsity and the numerical stability. These update techniques are described in detail in Duff, Erisman and Reid [3].

In section 4, we present the update method that is proposed in this study. This updating approach never requires refactoring, which is required by the updating strategies described above.

4 The Proposed LU Factorization Update Method

The **LU** factorization update method proposed in this study performs operations only in the columns that were actually modified by the change in the basis, and it uses the structure of the sparse matrix in best manner possible.

It is possible that changes are not made in columns that are located after the entering column (e) at the basis, and/or the leaving column (l), because of the sparse structure of the columns that are involved. In other words, if a matrix entry is in the row k and in the original entering column e, it is equal to zero ($\mathbf{B}(k, e) = 0$). Additionally, if the matrix entry is in row k and in the leaving column l of the ($\mathbf{B}(k, l) = 0$), then column k does not change in the basis updating procedure.

To achieve greater efficiency and to reduce fill-in, the columns of the basis are ordered according to a specific strategy. In the initial basis, the stock columns are permuted to the first positions and the cutting patterns to the last positions of the matrix. When entering a column in the basis, the procedure verifies whether it is a stock or a cutting pattern column. The search for the position in which the column will be inserted at the basis is performed only for the positions of the stock columns or of the cutting pattern. The column will be inserted in the right position at the basis according to the number of nonzero entries. The set of cutting and stock columns is updated and the procedure is repeated.

In the proposed **LU** factorization update, the operations caused by the leaving column l are undone, in the reverse order of the factorization, from the last column to the $s + 1$ column, considering the **LU** sparse patterns.

The next step is to determine which columns from the entering column ($e + 1$ column) to the last remain unchanged, which is accomplished by entering column e at the basis. The operation will be performed only in the modified columns, after the column that enters in the basis e is updated.

Algorithm:

1. {Reordering:}
Order (static approach) the base columns according to the number of nonzero entries: permute the stock columns to the first positions and the cutting patterns to the latest positions of the matrix.

2. {LU Factorization:}
Perform a complete factorization of the base.

3. Verify the entering column and the leaving column of the base.
Search the position in which the column will be inserted at the basis only in the positions of the stock columns or of the cutting pattern.

4. {Undo the LU Factorization:}
Perform operations in the reverse order of factorization from the last column to column $s + 1$, considering the LU sparse patterns.

5. {Update the LU Factorization:}
Insert the column in the right position at the basis according to the number of nonzero entries (according to Step 3).
Perform the LU factorization in the entering column e at the base.
Determine which columns from the entering column (column $e + 1$) to the last column remain unchanged by entering column e at the basis.
Perform the operations only in the modified columns.

5 Implementation Issues

The objective of the implementation developed here is to evaluate the computational performance of the **LU** factorization update implemented in the GLPK ("Gnu Linear Programming Kit"). The GLPK uses the revised Simplex method with two phases and the Bartels-Golub update [1].

In this proposed approach, a static reordering approach is used that leads to sparse factorizations without computational effort, to obtain the order of the columns, as described in the previous section. The entering column e in the basis follows the static ordering, and an **LU** factorization update of the basis (which considers its sparsity) is performed.

An implementation of the Simplex method with the column generation technique was performed because the large number of columns that correspond to the cutting pattern, provide a very large constraint matrix. A branch-and-bound method was used to solve the constrained knapsack problem (this method is a modification of Gilmore and Gomory that takes into account the upper bounds) to generate the entering columns at the base.

Consider the mathematical model $(2.1) - (2.6)$, without (2.3) (the pattern form). It is assumed that $L_1 \geq L_k, k = 2, \ldots, K$, i.e., the stock length type 1 is the largest component. Additionally, it is assumed that $l_i \leq L_1, i = 1, \ldots, m$; otherwise, the problem would be infeasible. Therefore, we can build m columns that are associated with simple cutting patterns that, have $m + K$ components,

$$a_{j1} = (0, \ldots, b_{jj}, 0, \ldots, 1, 0, \ldots, 0)^T,$$

where $b_{jj} = min\{L_1/l_j, d_j\}, j = 1, \ldots, m$.

Any basis has $m+K$ columns; thus, K more columns are needed, which could include columns that are associated with slack variables. When $x_{jk} = 0$, the m's first equations are sufficient to determine the values of $x_{j1}, j = 1, \ldots, m$.

Then, the feasibility of the solution must be analyzed. If $\sum_{j=1}^{m} x_{j1} \leq e_1$, then the solution is feasible, and the initial basic matrix is B, with dimension $(m + K)$. Next, if $\sum_{j=1}^{m} x_{j1} > e_1$, then the solution is infeasible and a phase I of the Simplex method is necessary.

The column generation Simplex method solves only the linear relaxation of the cutting stock problem. To improve the performance of the Simplex method with columns generation, using the sparse structure of the matrix, the news techniques of the **LU** factorization update are used to solve the linear systems. Such techniques were not considered in Poldi and Arenales [12].

The implementation of this approach uses GLPK functions to perform the **LU** factorization of the base and the updating columns that are entered at the base in the Simplex method. The implementation of the **LU** factorization update proposed was integrated into this implementation. Therefore, it was possible to perform computational experiments using the GLPK functions and the proposed **LU** factorization update.

6 Computational Experiments

The cutting problems used in the numerical experiments were generated randomly. The random generator of instances was developed by Poldi and Arenales [12]. In the first experiment (Table 1), 12 classes were generated with 40 instances in each class, resulting in a total of 480 instances. In the second experiment (Table 2), 30 classes were generated with 20 instances in each class, resulting in a total of 600 instances. The parameters used in the problems generation were the following.

- *Number of stock objects:* $K = 3, 5, 7, 50, 60, 100$ and 200;
- *Stock length objects:* The values of the $L_k, k = 1, \ldots, K$ were randomly generated and varied between 1 and 100 length units;

- *Available stock length:* The values of the $e_k, k = 1, \ldots, K$ were randomly generated from 1 to $100\frac{m}{2}$;
- *Number of item types:* Different values were used $m = 5, 10, 20, 100, 200$ and 300;
- *The length of the items to be produced:* The length of items l_i was randomly generated between $v_1 L$ and $v_2 L$, where L was the average value among the L_k, $k = 1, \ldots, K$. The parameters v_1 and v_2 were fixed at $v_1 = 0.01$ or 0.1 and $v_2 = 0.2$ or 0.8. Combining these values, we generated classes with problems that were *smalls* ($v_1 = 0.01$ and $v_2 = 0.2$), *mediums* ($v_1 = 0.01$ and $v_2 = 0.8$) and *larges* ($v_1 = 0.1$ and $v_2 = 0.8$).
- *Demand:* We used two different generators for the values of the demand of the items: the first generator produced values close to 10, ($d_i \in [7, 12], i = 1, \ldots, m$) and the second generator produced values close to 100, ($d_i \in [70, 120], i = 1, \ldots, m$).

With the objective of comparing the computational time with the GLPK solver, the Simplex method simulation and the proposed **LU** factorization update were implemented in the C programming language.

In Table 1, the examples were generated according to Poldi and Arenales [12]. The experiments were run on an Intel Core 2 Quad (2.33GHz and 2.96GB RAM). We verified that the proposed update obtained a reduction in the computational time in comparison with the GLPK update. Better results were obtained in the $C_3; C_1; C_6$, and C_{11} classes, with a reduction in the computational time of $50\%; 28\%; 23\%$, and 15%, respectively.

Table 2 shows the results of the large examples that were tested. In this case, the experiments were run on an Intel Core i7 (2.93GHz, 16.00GB RAM and 64Bits Operational System). The proposed update obtained a reduction in the computational time, and better results were found in the $C_{13}; C_{23}$, and C_3 classes, with a reduction in the computational time of $43\%; 40\%$, and 23%, respectively.

7 Conclusions

In this study, a static reordering of the matrix columns for linear programming problems is proposed, leading to a Simplex base with sparse LU factorizations and inexpensive factorization updates.

This method uses a specific reordering according to a specific strategy, to reduce fill-ins. The reordering has no initialization or updating costs because there is no need to reorder the columns in the factorization.

Table 1 shows the proposed method for obtaining a reduction in the computation time of 50% for class C_3 in comparison with the GLPK update. The proposed **LU** factorization update reduced up to 43% of the computation time in comparison with the GLPK in Table 2.

The updating approach never requires refactoring; thus, it is faster than other updating strategies. A further result is the following. If the starting basis is the identity matrix, then it is not necessary to perform any **LU** factorization at all.

Acknowledgements

This research was sponsored by the Foundation for the Support of Research of the State of São Paulo (FAPESP), the Brazilian Council for the Development of Science and Technology (CNPq) and Foundation for Development of UNESP (FUNDUNESP). Thanks also go to Kelly Christina Poldi for providing the cutting stock problem instances generator.

References

[1] R. Bartels and G. Golub, *The Simplex method of linear programming using the LU decomposition*, Communications of the Association for Computing Machinery, 12 (1969), pp. 266–268.

[2] R. Braytom, F. Gustavson, and R. Willoughby, *Some Results on Sparse Matrices*, IBM Research Center, Yorktown Heights, N.Y., 1969.

[3] I. Duff, A. Erisman, and J. Reid, *Direct Methods for Sparse Matrices*, Clarendon Press, Oxford, 1986.

[4] J. Forrest and J. Tomlin, *Updating triangular factors of the basis to maintain sparsity in the product form Simplex method*, Mathematical Programming, 2 (1972), pp. 263–278.

[5] P. Gilmore and R. Gomory, *A linear programming approach to the cutting stock problem*, Operations Research, 9 (1961), pp. 848–859.

[6] ——, *A linear programming approach to the cutting stock problem - part ii*, Operations Research, 11 (1963), pp. 863–888.

[7] A. Hinxman, *The trim-loss and assortment problems: a survey*, European Journal of Operational Research, 5 (1980), pp. 8–18.

[8] O. Holthaus, *Decomposition approaches for solving the integer one-dimensional cutting stock problem with multiple stock lengths*, European Journal of Operational Research, 44 (2002), pp. 295–312.

[9] J. L. Kennington and R. V. Helgason, *Algorithms for Network Programming*, Wiley, New York, 1980.

[10] H. Markowitz, *The elimination form of the inverse and its applications to linear programming*, Management Science, 3 (1957), pp. 255–269.

[11] I. Maros, *Computational Techniques of the Simplex Method*, Kluwer Academic Publishers, 2003.

[12] K. Poldi and M. Arenales, *Heuristics for the one-dimensional cutting stock problem with limited multiple stock lengths*, Computers and Operations Research, 36 (2009), pp. 2074–2081.

[13] J. Reid, *A sparsity-exploiting variant of the Bartels-Golub decomposition for linear programming bases*, Mathematical Programming, 24 (1982), pp. 55–69.

[14] H. A. Stadtler, *A one-dimensional cutting stock problem in the aluminiun industry and its solution*, European Journal of Operational Research, 44 (1990), pp. 209–230.

[15] U. Suhl and L. Suhl, *A fast LU update for linear programming*, Annals of Operations Research, 43 (1993), pp. 33–47.

[16] G. Wscher and T. Gau, *Heuristics for the integer one-dimensional cutting stock problem: a computational study*, OR Spektrum, 18 (1996), pp. 131–144.

Classes	Data				Computational time (in seconds)	
	K	m	Items	Demand	Proposed update	GLPK update
C1	3	5	small	small	0.080	0.111
C2	3	5	medium	small	0.015	0.016
C3	3	5	large	small	0.016	0.032
C4	3	20	small	small	15.566	15.837
C5	5	10	small	small	155.533	158.126
C6	5	10	medium	small	0.157	0.203
C7	5	10	large	small	0.109	0.125
C8	5	20	large	small	0.126	0.141
C9	7	10	small	small	1.361	1.423
C10	7	10	medium	small	0.250	0.267
C11	7	10	large	small	0.172	0.202
C12	7	20	large	small	1.967	1.985

Table 1: Computational time of the Poldi and Arenales [12] examples.

Classes	Data				Computational time (in seconds)	
	K	m	Items	Demand	Proposed update	GLPK update
C1	50	100	medium	small	135.87	164.52
C2	50	100	medium	large	13.57	16.99
C3	50	100	large	small	14.12	18.22
C4	50	100	large	large	9.92	12.40
C5	50	150	medium	small	397.22	415.72
C6	50	150	medium	large	91.24	94.35
C7	50	150	large	small	83.92	85.67
C8	50	150	large	large	45.95	46.96
C9	50	200	medium	large	223.32	231.19
C10	50	200	large	small	253.10	256.27
C11	50	200	large	large	113.82	121.69
C12	50	300	medium	large	907.30	1100.21
C13	50	300	large	small	594.25	1040.21
C14	50	300	large	large	452.48	455.55
C15	100	150	medium	small	3168.53	3520.48
C16	100	150	medium	large	127.31	136.73
C17	100	150	large	small	129.68	133.83
C18	100	150	large	large	75.64	78.67
C19	100	200	medium	small	12000.00	13138.97
C20	100	200	medium	large	346,48	360,26
C21	100	200	large	small	300.57	316.14
C22	100	200	large	large	202.22	212.04
C23	100	300	large	small	1564.04	2589.34
C24	100	300	large	large	782.90	934.98
C25	150	200	medium	large	551.29	597.00
C26	150	200	large	large	284.38	294.6
C27	150	300	medium	large	2999.50	3556.90
C28	150	300	large	large	1110.01	1144.83
C29	200	300	large	large	1286.00	1539.25
C30	250	300	large	large	1817.38	1867.38

Table 2: Computational time of the large examples.