COMPUTATIONAL & APPLIED MATHEMATICS
Volume 27, N. 1, pp. 61–78, 2008

ISSN 0101-8205
www.scielo.br/cam

Nonlinear cutting stock problem model to minimize the number of different patterns and objects

ANTONIO CARLOS MORETTI[1] and LUIZ LEDUÍNO DE SALLES NETO[2]

[1]Instituto de Matemática, Estatística e Computação Científica – UNICAMP
Cidade Universitária Zeferino Vaz s/n, Barão Geraldo, 13084-790 Campinas, SP, Brazil
[2]Escola de Engenharia Industrial Metalúrgica – UFF
Av. dos Trabalhadores, 420, Vila, 27255-970 Volta Redonda, RJ, Brazil
E-mails: moretti@ime.unicamp.br / leduino@metal.eeimvr.uff.br

Abstract. In this article we solve a nonlinear cutting stock problem which represents a cutting stock problem that considers the minimization of, both, the number of objects used and setup. We use a linearization of the nonlinear objective function to make possible the generation of good columns with the Gilmore and Gomory procedure. Each time a new column is added to the problem, we solve the original nonlinear problem by an Augmented Lagrangian method. This process is repeated until no more profitable columns is generated by Gilmore and Gomory technique. Finally, we apply a simple heuristic to obtain an integral solution for the original nonlinear integer problem.

Mathematical subject classification: 65K05.

Key words: cutting problem, nonlinear programming, column generation.

1 Introduction

The Unidimensional Cutting Stock Problem (1/V/I/R according Dyckhoff [5]) is characterized by cutting stocks in just one dimension. More specifically, we have m items with different sizes with width equal to w_i and we must cut, through its length, a minimum number of master rolls (with width $W > w_i$ for all i) to attend demand d_i for each item i. Each combination of items cut from a master

#728/07. Received: 01/III/07. Accepted: 13/VI/07.

roll is called a cutting pattern. The problem is to determine the frequency of each cutting pattern to attend demand and (for instance) minimize the number of objects cut.

A reasonable goal to be met in a industry is to minimize the number of master rolls used to produce the demanded items. If we consider that there are a sufficient number of objects of same width W available, then the formulation below describes the mathematical model that minimize the total number of objects (i.e., master rolls) used in a cutting plan:

$$(P_1)\begin{cases} \text{Minimize} & c_1 \displaystyle\sum_{j=1}^{n} x_j \\ \text{subject to} & \displaystyle\sum_{j=1}^{n} a_{ij} x_j \geq d_i, \quad i = 1, \ldots, m. \\ & x_j \in \mathbb{N}, \qquad\qquad j = 1, \ldots, n. \end{cases}$$

where

- c_1 is the cost for each master roll used;
- a_{ij} is the number of items i in cutting pattern j;
- x_j is the number of of objects cut according cutting pattern j.

In some cases, the minimization the number of objects used are not the only goal for the manager. In fact, when we have a large demand to attend in a short period of time, the number of machine setup done for cutting the items from the master rolls takes a growing importance, since each time we process a cutting pattern there is a need to adjust the knives in the cutting machine and this adjustment takes time. Adding this setup cost in the previous problem (P_1) we obtain a new formulation which minimizes the number of objects and the number of setup:

$$(\overline{P}_1)\begin{cases} \text{Minimize} & c_1 \displaystyle\sum_{j=1}^{n} x_j + c_2 \sum_{j=1}^{n} \delta(x_j) \\ \text{subject to} & \displaystyle\sum_{j=1}^{n} a_{ij} x_j \geq d_i, \qquad i = 1, \ldots, m. \\ & x_j \in \mathbb{N}, \qquad\qquad j = 1, \ldots, n. \end{cases}$$

where c_2 is the setup cost and $\delta(x_j) = \begin{cases} 1 & \text{if } x_j > 0, \\ 0 & \text{if } x_j = 0. \end{cases}$

Combinatorial problems involving setup costs are known to be very hard to solve. In particular $(\overline{P}_1)$ presents two conflicting objectives: (1) Minimize the total number of processed objects and (2) the total number of setup used.

Solving problem (P_1) is already a hard task to do, since it is a NP-Hard problem. Problem $(\overline{P}_1)$ is a harder problem, since, besides being a nonlinear integer problem, the nonlinear part of the objective function is discontinuous. This fact, does not allow us to solve the problem by using the Gilmore and Gomory strategy [9, 10]. Vanderbeck [22] investigates the problem of minimizing the number of different cutting patterns as an integer nonlinear programming, where the number of objects is fixed. In his approach, Vanderbeck uses Dantzig-Wolfe decomposition [20, 21]. Since the model considered works with a huge number of variables, the method solves only problems with a small number of items. For this reason several papers considering this problem involve the use of heuristic procedures. Below, we describe two of these methods which will be used to compare with our approach.

- **Sequential Heuristic Procedure – SHP**: It was proposed by Haessler [11] and it is based on an exaustive technique of cutting pattern repetitions. In each iteration an aspiration criterion is computed then a search is done to look for cutting patterns that satisfy such parameters until the demand are all attended. The SHP give us a good initial solution and it is used by others method to compare the quality of their solutions. It generates an inexpensive good initial solution to the $(\overline{P}_1)$.

- **Kombi**: This method was developed by Foester and Wascher [7] and it is based on a combination of cutting patterns in order to reduce the number of setups of a given cutting plan. The idea of reducing the number of cutting patterns using a post-optimization procedure was initially mentioned by Hardley [13]. Others methods based on this idea were published by Johnston [15], Allwood and Goulimis [1] and Diegel et al. [3]. All of them have in common the combinations of pair or triples of cutting patterns, but, they differ in the way the combinations are carried out. The method

Kombi can be seen as a generalization of Diegel's method, in which ideas of combining cutting patterns are extended to a consistent system independently from the number of cutting patterns combined. It makes use of the fact that the sum of the cutting pattern frequencies of the resulting cutting patterns has to be identical to the sum of the frequencies belonging to the original cutting patterns in order to keep the material input constant. The results presented show that the setup was reduced by up to 60% in relation to the original cutting plan. Kombi has also been proved superior to SHP.

2 Smoothing a discontinuous cost function

Many practical problems require the minimization of functions that involve discontinuous costs. Martinez [16] propose a smoothing method for the discontinuous cost function and establish sufficient conditions on the approximation that ensure that the smoothed problem really approximate the original problem.

Consider the problem

$$\text{Minimize} \quad f(x) + \sum_{i=1}^{m} H_i[g_i(x)] \quad \text{subject to} \quad x \in \Omega \tag{2.1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is continuous, $g_i : \mathbb{R}^n \to \mathbb{R}$ is continuous for all $i = 1, \ldots, m$ and $\Omega \subseteq \mathbb{R}^n$. Also, $H_i : \mathbb{R} \to \mathbb{R}$, $i = 1, \ldots, m$, are nondecreasing functions such that H_i is continuous except at breakpoints α_{ij}, $j \in I_i$. The set I_i can be finite or infinite but the set of breakpoints is discrete, in the sense that:

$$\inf \left\{ |\alpha_{il} - \alpha_{ij}| \ \text{ such that } \ l, j \in I_i, l \neq j \right\} > 0. \tag{2.2}$$

The side limits $\lim_{t \to \alpha_{ij}^-} H_i(t)$, $\lim_{t \to \alpha_{ij}^+} H_i(t)$, exist for all $j \in I_i$ and

$$\lim_{t \to \alpha_{ij}^-} H_i(t) = H_i(\alpha_{ij}) < \lim_{t \to \alpha_{ij}^+} H_i(t)$$

for all $j \in I_i$, $i = 1, \ldots, m$.

The cost functions H_i will be approximated by a family of continuous nondecreasing functions $H_{ik} : \mathbb{R} \to \mathbb{R}$.. We assume that the approximating functions are such that, for all $\mu > 0$,

$$\lim_{k \to \infty} H_{ik}(t) = H_i(t) \ \text{ uniformly in } \ \mathbb{R} \setminus \bigcup_{j \in I_i} (\alpha_{ij}, \alpha_{ij} + \mu). \tag{2.3}$$

Note that (2.3) implies that

$$\lim_{k\to\infty} H_{ik}(t) = H_i(t) \quad \forall\, t \in \mathbb{R}.$$

For each k, we define the approximated problems as

$$\text{Minimize} \quad f(x) + \sum_{i=1}^{m} H_{ik}[g_i(x)] \quad \text{subject to} \quad x \in \Omega. \tag{2.4}$$

Since (2.4) has a continuous objective function, we can use continuous optimization algorithms to solve it. The following theorem prove that the solution of (2.1) can be approximated by the solution of (2.4).

Theorem 2.1 (Martinez [16]) *Assume that for all* $k = 0, 1, 2, \ldots$, x^k *is a solution of* (2.4) *and that* $x^* \in \Omega$ *is a cluster point of* $\{x^k\}$. *Then,* x^* *is a solution of* (2.1).

We adapt those ideas for the (P_1). Also, we relax (P_1) by eliminating the integrality constraints. We will denote this problem as

$$(P_2) \quad \left\{\text{Minimize} \quad f(x) + \sum_{i=1}^{n} H_i(x_i) \quad \text{subject to} \quad x \in \Omega\right.$$

where:

- $\Omega = \left\{x \in \mathbb{R}^n \text{ such that } Ax = d,\ x \geq 0\right\}$;
- $f(x) = c_1 \cdot \sum_{i=1}^{n} x_i$;
- $H_i(t) = c_2\delta(t)$, $i = 1, \ldots, n$.

Note that $I_i = \{0\}$ for all $i = 1, \ldots, n$ once the only discontinuous point of $H_i(t)$, $i = 1, \ldots, n$ is $t = 0$. Also, we have that

$$\lim_{t\to 0^-} H_i(t) = 0 = H_i(0) < \lim_{t\to 0^+} H_i(t) = c_2.$$

We approximate each one of the functions H_i by the following continuous functions:

$$H_{ik}(t) = \begin{cases} 0 & \text{if } t \leq 0 \\ c_2 k t^2/(1 + kt^2) & \text{if } t > 0 \end{cases}$$

Is easy to see that

$$\lim_{k\to\infty} H_{ik}(t) = H_i(t)$$

for all $t \in \mathbb{R}$ and for all $i = 1, \ldots, n$; and, $H_{ik}(t)$ uniformly converges to $H_i(t)$ if $t \neq 0$.

Therefore, the conditions of Theorem (2.1) can be applied in this case. Let P_k denotes the approximate nonlinear programming problem:

$$(P_k)\begin{cases} \text{Minimize} & c_1 \cdot \sum_{j=1}^{n} x_j + c_2 \cdot \sum_{j=1}^{n} \dfrac{kx_j^2}{1+kx_j^2} & \\ \text{subject to :} & \sum_{j=1}^{n} a_{ij} \cdot x_j \geq d_i & i = 1, \ldots, m \\ & x_j \geq 0 & j = 1, \ldots, n. \end{cases}$$

So, Theorem (2.1) says that if for all $k = 0, 1, 2, \ldots$, the point x_k is the best solution found for the approximate problem P_k and x^* is the cluster point of the sequence $\{x_k\}$ then x^* is the solution of (P_2).

Once we obtain a solution for (P_2), we can use a rounding procedure to obtain a integer solution for $(\overline{P}_1)$. However, there exist a problem to be solved before that: "How to generate good columns (i.e., cutting patterns) for problem (P_2)?". Next section we answer this question.

3 Column generation in a nonlinear problem

The column generation procedure developed by Gilmore and Gomory [9, 10] for linear programming problems, made it possible to solve large scale cutting stock problem. Problems encountered in real life may involve a very large numbers of variables, the trick is to work with only a few cutting patterns (variables) at a time and to generate new profitable cutting patterns only when they are really needed. In [17] we applied the column generation procedure in a nonlinear problem by making use of an auxiliary linear programming problem "closer" to our nonlinear problem. By "closer", we mean that, the solution of (P_2) satisfies the optimality conditions of problem (P_3).

Consider the following linear programming problem:

$$(P_3)\begin{cases} \text{Minimize} & \displaystyle\sum_{j=1}^{P} x_j \\ \text{subject to} & \displaystyle\sum_{j=1}^{P} a_{ij}x_j \geq d_i, \quad i = 1, \ldots, m. \\ & x_j \geq 0, \qquad\qquad j = 1, \ldots, S. \end{cases}$$

where S is the number of different cutting patterns in the solution obtained in P_2.

In Belov and Scheithauer [2], they propose a linearization of the bi-criterion cutting stock objective function in the following way: After solving a sequence of problems P_k we get a (cluster) solution x_j^* for $j = 1, \ldots, n$ and let us call P_4 the following auxiliary linear programming problem:

$$(P_4)\begin{cases} \text{Minimize} & \displaystyle\sum_{j=1}^{n} \left(c_1 + \frac{c_2}{u_j}\right) x_j \\ \text{subject to:} & \displaystyle\sum_{j=1}^{n} a_{ij}x_j \geq d_i, \qquad i = 1, \ldots, m. \\ & x_j \geq 0, \qquad\qquad j = 1, \ldots, n \end{cases}$$

where $u_j = x_j^*$ if $x_j^* > 0$.

We use (P_4) to generate a new column for the original problem (P_2). In the column generation procedure, we need to solve a Knapsack Problem. To generate good profitable columns in the sense of reducing trim loss and setup number, Haessler [12] suggests to solve a bounded knapsack problem where the upper bounds for the variables are fixed according to:

$$b_i = \min\left\{\left\lfloor \frac{d_i}{10} \right\rfloor, \left\lfloor \frac{W}{w_i} \right\rfloor\right\}, \quad i = 1, \ldots, m.$$

In our work, we accepted Haessler's suggestion, but, we compute the upper limits in a different way. This modification is described in Section 4.

4 Solving the Nonlinear Cutting Problem

Below, we describe the main steps of the new algorithm for solving the nonlinear cutting stock problem.

New Algorithm – NANLCP

Step 1: Compute an initial solution (x^*) for ($\overline{P}_1$) using SHP;

Step 2: Obtain a solution for the current (P_2) solving P_k for $k = 10^i$, $i = 1, 2, \ldots, 5$;

Step 3: If the solution obtained in Step 2 is better, for $f(x) = c_1 \cdot \sum_{i=1}^{n} x_i + c_2 \sum_{i=1}^{n} \delta x_i$ than x^*, update it;

Step 4: Get the simplex multiplier π_i, $i = 1, \ldots, m$ by solving (P_4);

Step 5: Solve a Bounded Knapsack Problem with the objective function coefficients given by the simplex multiplier of (P_4):

$$\begin{aligned} \text{Maximize} \quad & Z = \sum_{i=1}^{m} \pi_i y_i \\ \text{subject to} \quad & \sum_{i=1}^{m} w_i y_i \leq W \\ & y_i \leq b_i, \qquad i = 1, \ldots, m \\ & y_i \in \mathbb{N} \qquad i = 1, \ldots, m. \end{aligned}$$

Step 6: If $Z \leq 1$ solve the Knapsack Problem with no limits in the variables with $\overline{Z}$ as the optimal objective function value;

Step 7: If $\overline{Z} \leq 1$ go to 8. Otherwise, add the new column into problem (P_2) and go back to Step (2).

Step 8: Use a rounding procedure to obtain an integer solution. Stop.

To obtain an initial solution for $(\overline{P}_1)$,we implemented the SHP method with some modifications, as described in [17].

The software BOX9903 [14] was used to solve the sequence of nonlinear problems. The BOX9903 solves the optimization problem:

$$\begin{array}{ll} \text{Minimize} & f(x) \\ \text{subject to} & Ax = d \\ & h(x) = 0 \\ & l \leq x \leq u \end{array}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ and $h : \mathbb{R}^n \to \mathbb{R}^m$ are differentiable functions and A is a real matrix $m \times n$. The format of our problem does not include the function $h(x)$ and the limits are defined by $l = 0$ e $u = \infty$. So, in each iteration, BOX9903 solves the problem:

$$\begin{array}{ll} \text{Minimize} & L(x) \\ \text{subject to} & l \leq x \leq u \end{array}$$

where

$$L(x) = c_1 \sum_{i=1}^{n} x_i + c_2 \sum_{i=1}^{n} h_{ik}(x_i) + \lambda^t \cdot (\overline{A}x - d) + (\rho/2) \cdot \left\| \overline{A}x - d \right\|_2^2$$

is called the Augmented Lagrangian. In iteration j we define $\rho = K = 10^j$, where j assumes the values 1, 2, 3, 4 and 5. The Lagrange multiplier is estimated in each iteration j. Since this is a local method, each time a column is added to the problem $(\overline{P}_1)$, we solve a sequence of problems P_k with 20 different starting points: the current solution, the null solution and 18 random generated points.

After solving a sequence of problems $(P_k)'^s$ and obtaining the best solution for the new problem, we need to solve a Bounded Knapsack Problem to find if there is a profitable column (i.e.,a cutting pattern) to be appended to the problem (P_2). To obtain the simplex multipliers, used as coefficients in the objective function of the Knapsack Problem, we work with problem (P_3). In the Bounded Knapsack Problem, we determine the bounds of the components of the current solution (i.e., variables x_j) using the following reasoning: Let the setup number be equal S, $S \leq m$, that means that we have S different cutting patterns. Our interest, when adding a new column, is to reduce not only the trim loss,

but, also the setup number. Assume we want to reduce the current setup number by 20%. Therefore, after adding the new generated column to (P_2) we hope the new setup number (denoted by NSN) to be 0.8*S. Let NB be the number of production rolls processed in the current solution. Suppose that this number remains constant, we should have, on average, $(MBP = NB/NSN)$ processed object per cutting pattern. And, assuming the new generated cutting pattern will belong to the solution with frequency equal to MBP and that the items in this cutting pattern will not be in the others nonzeros cutting patterns, then to guarante a feasible solution, each item in this cutting pattern must be limited by $b_i = \lfloor d_i/MBP \rfloor$. And, if $b_i > W/w_i$ then we fix $b_i = \lfloor W/w_i \rfloor$ so we will obtain only feasible cutting patterns.

Finally, we used the BRURED method described in [23] to round the solution found in the end of the process. This method does not mess up with the setup number and it is fast. First, we round the nonzero variables up, that is, $x_j^* = \lceil x_j \rceil$. Usually, this procedure may generate an excess of production. This excess can be reduced by checking which variables can be reduced by one unit without making the problem infeasible. So, for each $k \in \{1, 2, \ldots, n\}$, if the variable x_k, after the round up, satisfy the inequality

$$a_{ik}(x_k - 1) + \sum_{i=1; i \neq k}^{m} a_{ij} x_j \geq d_i \ , \ i = 1, \ldots, m$$

then we make $x_k^* = x_k - 1$.

5 Computational experiments

In order to evaluate our approach, we generated several random instances using the one-dimensional cutting stock problem's generator, CUTGEN1, developed by Gau and Wascher [8]. The input parameters for the CUTGEN are

- m = problem size;
- W = standard length;
- v_1 = lower bound for the relative size of order lengths in relation to W, i.e. $w_i \geq v_1 \times W \quad (i = 1 \ldots m)$;

- v_2 = upper bound for the relative size of order lengths in relation to W, i.e. $w_i \leq v_2 \times W \quad (i = 1 \ldots m)$;
- $\overline{d}$ = average demand per order length.

By using these parameters we generate the following data set

- $w_i \in [v_1 \times W, v_2 \times W], \quad 1 = 1, \ldots, m$;
- $d_i, \quad i = 1, \ldots, m$ such that the total demand $D = m \times \overline{d}$.

As in Foester and Wascher's work [7], we generated 18 classes of random problems by combining different values of the generator's parameters:

- v_1 assumed values 0.01 and 0.2;
- v_2 assumed values 0.2 and 0.8;
- the number of items in the original cutting plan, denoted by m was set to 10 (small instances), 20 (mid-size instances) and 40 (large instances);
- problems with low average demand ($\overline{d} = 10$) and high average demand ($\overline{d} = 100$);
- In all the classes $W = 1000$.

Each class contains 100 instances. We generated six classes with small items ($v_1 = 0.01$ and $v_2 = 0.2$), six classes with wide-spread items ($v_1 = 0.01$ and $v_2 = 0.8$) and other six classes with large items ($v_1 = 0.2$ and $v_2 = 0.8$). Table 1 shows the 18 classes and theirs parameters.

The solutions obtained by NANLCP were compared with the solutions of the methods: SHP [11], KOMBI234 [7] and ANLCP [17]. These methods (not including ANLCP) were also used as basis of comparison by Umetami et al. [19] with the heuristic APG. We run CUTGEN1 with the same seed (i.e.,1994) and parameters defined in [7] and [19] to generate all the 1800 instances (i.e., 18 classes, each class with 100 instances).

KOMBI234 uses the solution obtained by the heuristic developed by Stadler [18] as a starting solution. This combination, Stadler + KOMBI234, produced the best results encountered in the literature, at the time Foester and Wascher published their work.

Class	1	2	3	4	5	6	7	8	9
v1	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
v2	0.2	0.2	0.2	0.2	0.2	0.2	0.8	0.8	0.8
$\overline{d}$	10	100	10	100	10	100	10	100	10
Class	10	11	12	13	14	15	16	17	18
v1	0.01	0.01	0.01	0.2	0.2	0.2	0.2	0.2	0.2
v2	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8
$\overline{d}$	100	10	100	10	100	10	100	10	100

Table 1 – Random generated classes and their parameters.

The methods NANLCP and ANLCP and the heuristic SHP were implemented by the authors in Fortran, running under g77 for Linux, in a Athlon XP 1800Mhz, 512MB of RAM. The results for KOMBI234 were obtained from an implementation in MODULA-2 in MSDOS operating system with a IBM 486/66. Therefore, the time was not considered when the comparison is done with KOMBI234.

Below, we present the computational results. We fix c_1, the cost for each object used, equal to 1 and c_2, the setup cost, equal to 100. The value of c_2 can be seen as a penalization parameter for the setup number.

Tables 2, 3 and 4 show the average of the setup number and the average of the number of objects used in the final solution of the 100 instances in all the classes: small items, wide-spread items and large items. The averages of setup and number of objects described in the end of each table suggest that the NANLCP has a better performance than ANLCP, Kombi234 and SHP.

Table 5 presents the variation of setup and number of objects of the method (NANLCP) in relation to SHP. For instance, to compute the variation for the setup number we use the formula

$$100 \times (\text{Setup}_{\text{NANLCP}} - \text{Setup}_{\text{SHP}})/\text{Setup}_{\text{SHP}},$$

therefore, a negative number indicates that NANLCP was better than SHP. The average setup in NANLCP method was better than the average setup for SHP in all the classes, except for classes 5 and 6. And, the average number of objects in NANLCP method was better than the average obtained by SHP in 14 classes out of 18.

Class	SHP		Kombi		ANLCP		NANLCP	
	Setup	Objects	Setup	Objects	Setup	Objects	Setup	Objects
1	3.95	14.17	3.40	11.49	3.14	18.44	3.01	14.84
2	5.94	116.47	7.81	110.25	4.66	116.66	4.76	119.62
3	5.00	25.29	5.89	22.13	4.88	25.18	4.91	24.26
4	7.31	225.33	14.26	215.93	7.16	226.72	7.16	223.91
5	6.87	46.89	10.75	42.96	7.02	45.64	7.04	45.96
6	10.81	433.59	25.44	424.71	10.96	432.68	10.84	433.29
Average of the averages	6.65	143.62	11.26	137.91	6.30	144.22	6.29	143.64

Table 2 – Averages for small items.

Class	SHP		Kombi		ANLCP		NANLCP	
	Setup	Objects	Setup	Objects	Setup	Objects	Setup	Objects
7	8.84	55.84	7.90	50.21	5.78	50.84	5.31	53.69
8	9.76	515.76	9.96	499.52	8.22	506.02	6.97	488.85
9	17.19	108.54	15.03	93.67	10.90	106.72	10.92	105.65
10	19.37	1001.59	19.28	932.32	14.56	969.40	12.80	932.67
11	32.20	202.80	28.74	176.97	19.80	220.46	21.12	216.67
12	37.25	1873.05	37.31	1766.20	25.58	1813.60	25.25	1839.63
Average of the average	20.77	626.26	19.70	586.48	14.14	611.17	13.78	606.19

Table 3 – Averages for wide-spread items.

Class	SHP		Kombi		ANLCP		NANLCP	
	Setup	Objects	Setup	Objects	Setup	Objects	Setup	Objects
13	9.38	69.97	8.97	63.27	5.86	66.52	6.31	66.77
14	9.85	643.55	10.32	632.12	7.92	633.98	7.89	639.88
15	18.03	136.03	16.88	119.43	10.28	130.20	11.13	123.93
16	19.63	1253.55	19.91	1191.80	15.00	1193.54	14.44	1169.12
17	34.39	256.01	31.46	224.68	23.32	283.88	21.96	262.07
18	38.23	2381.54	38.28	2342.40	29.80	2410.82	26.03	2247.11
Average of the average	20.77	626.26	19.70	586.48	14.14	611.17	13.78	606.19

Table 4 – Averages for large items.

NANLCP versus SHP – Variation cost in relation to								
Class	Setup Number	Object Number	Class	Setup Number	Object Number	Class	Setup Number	Object Number
1	–23.80	+4.73	7	–39.93	–3.85	13	–32.73	–4.57
2	–19.87	+2.70	8	–28.59	–5.22	14	–19.90	–0.57
3	–1.80	–4.07	9	–36.47	–2.66	15	–38.27	–8.90
4	–2.05	–0.63	10	–33.92	–6.88	16	–26.44	–6.74
5	+2.47	–1.98	11	–34.41	+6.84	17	–36.14	+2.37
6	+0.28	–0.07	12	–32.21	–1.78	18	–31.91	–5.64

Table 5 – Variation in % of NANLCP in relation to SHP.

NANLCP versus Kombi – Variation cost in relation to								
Class	Setup Number	Object Number	Class	Setup Number	Object Number	Class	Setup Number	Object Number
1	–11.47	29.16	7	–32.78	+6.93	13	–29.65	+5.53
2	–39.05	+8.50	8	–30.02	–2.14	14	–23.55	+1.23
3	–16.64	+9.62	9	–27.35	12.79	15	–34.06	+3.77
4	–49.79	+3.70	10	–33.61	+0.04	16	–27.47	– 1.90
5	–34.51	+6.98	11	–26.51	22.43	17	–30.20	16.64
6	–57.39	+2.02	12	–32.32	+4.16	18	–32.00	– 4.07

Table 6 – Variation in % of NANLCP in relation to Kombi.

Table 6 presents the variation of setup and the number of objects of our method (NANLCP) in relation to KOMBI234. In all the classes NANLCP obtained a better average for the setup than KOMBI. But, the average number of objects used by KOMBI was better than NANLCP in the 15 classes.

When comparing the quality of the solutions of each method, we use c_2 equal 5 and 10 in the objective function. Those are "real life" values for c_2 and by doing so, the comparison with the others methods were more honest. In general, NANLCP obtained better objective function values than SHP and KOMBI.

In the literature, Diegel et al. [4] were the only to mention about practical values for c_1 and c_2. According to Diegel, a exact relation between c_1 and c_2 depends on the data we have on hands. But, they say, that c_2 is never much bigger than c_1. However, if the main goal is to minimize the setup number then c_2 must be bigger than c_1. Therefore, the relation between those two cost depends on several factors as: demand, deadlines, labor costs, etc.

The results obtained by NANLCP when $c_1 = 1$ e $c_2 = 5$ are presented in Table 7. The averages for the objectives function values for NANLCP were better than those obtained by SHP in all 18 classes an better than KOMBI in 15 classes.

The computational results confirm the good performance of NANLCP method. The version NANLCP, as shown in Table 8, obtained average costs better than SHP in 16 classes and better than KOMBI in all the classes.

Objective function values – NANLCP versus								
Class	SHP	KOMBI	Class	SHP	KOMBI	Class	SHP	KOMBI
1	–11.88	+ 4.91	7	–19.79	–10.56	13	–15.87	–9.06
2	–1.88	– 3.94	8	–7.24	–4.66	14	–1.94	–0.64
3	–2.94	–5.37	9	–17.61	–5.08	15	–20.60	–11.90
4	– 0.83	–9.58	10	–9.26	–3.12	16	–8.17	–3.87
5	–0.10	–16.08	11	–11.42	+ 0.50	17	–13.11	–2.65
6	–0.03	–11.67	12	–4.54	+ 0.67	18	–7.60	–6.18

Table 7 – Variation in % of the objective function value, with $c_1 = 1$ e $c_2 = 5$, of NANLCP in relation to SHP and KOMBI.

Objective Function Values – NANLCP versus								
Class	SHP	KOMBI	Class	SHP	KOMBI	Class	SHP	KOMBI
1	–16.27	–1.21	7	–25.96	–17.35	13	–20.70	–15.10
2	–4.92	–11.22	8	–8.94	–6.77	14	–3.14	–2.25
3	–2.56	–9.47	9	–23.39	–11.94	15	–25.64	–18.39
4	–0.98	–17.58	10	–11.26	–5.73	16	–9.40	–5.56
5	+0.67	–22.66	11	–18.47	–7.86	17	–19.71	–10.68
6	+0.00	–20.24	12	–6.83	–2.20	18	–9.28	–7.99

Table 8 – Variation in % of the objective function value, with $c_1 = 1$ e $c_2 = 10$, of NANLCP in relation to SHP and KOMBI.

6 Conclusions and future work

Based on the computational results presented, we may affirm that the NANLCP has an average performance better that the ANLCP [17] and, therefore, it is a good method to use when the objective is to minimize the cost of the processed

object number and setup. Specifically in relation to Setup Number, NANLCP has a better performance than SHP and KOMBI in almost all the classes.

It is important to say that another advantage of NANLCP is the possibility of working with explicit values of c_1 and c_2 in the objective function. In fact, for real life problems these costs depend on many factors as demand, deadline, labor costs, etc. We do not know any other method that treat the problem of minimizing the setup and the number of processed object in the same way NANLCP does.

Computational time (in seconds)				
Class	SHP	KOMBI	ANLCP	NANLCP
1	0.01	0.14	0.80	0.83
2	0.08	1.14	1.17	1.21
3	0.17	1.74	0.47	0.94
4	0.21	16.00	0.94	1.22
5	0.27	38.03	0.58	0.89
6	0.31	379.17	0.93	1.02
7	0.01	0.07	16.49	13.44
8	0.02	0.20	8.96	16.51
9	0.04	3.37	69.11	75.81
10	0.06	3.25	77.21	142.02
11	0.22	36.26	185.53	168.67
12	0.32	76.31	318.54	420.53
13	0.01	0.08	4.44	5.12
14	0.02	0.13	1.95	4.44
15	0.03	1.81	29.36	61.68
16	0.04	2.60	26.30	78.34
17	0.16	50.93	248.62	250.04
18	0.24	70.94	443.66	390.75

Table 9 – Average time (in seconds) for each method.

However, NANLCP method has one disadvantage in relation to SHP and KOMBI: the computational time. Although, we can not compare the computational time with KOMBI, since it was not implemented by the authors and, therefore, it was not run in the same computational environment is easy to see that the NANLCP method has a higher computational time. This happens because once a new column is added to the problem, we need to solve (P_k) with

20 initial solutions to (hopefully) get rid of local minimum. For classes with a large numbers of items, as Classes (11, 12, 17, 18) the computational time for NANLCP was high.

Implementing better strategies to obtain global solutions when solving non-linear problems, can make NANLCP better. Also, a better strategy to obtain a integer solution from a fractional solution is welcome since the BRURED method used here is very naive, although, it is fast.

Acknowledgements. The authors appreciate the constructive comments of the referee, resulting in an improved presentation.

REFERENCES

[1] J.M. Allwood and C.N. Goulimins, *Reducing the number of patterns in one-dimensional cutting stock problems*. Control Section Report EE/CON/IC/88/10, Industrial Systems Group, Department of Electrical Engineering, Imperial College, London, (1988).

[2] G. Belov and G. Scheithauer, *The number of setups (different patterns) in one-dimensional stock cutting*. Technical Report, Dresden University (2003).

[3] A. Diegel, M. Chetty, S. Van Schalkwyck and S. Naidoo, *Setup combining in the trim loss problem – 3-to-2 & 2-to-1*. Working paper, Business Administration, University of Natal, Durban, First Draft (1993).

[4] A. Diegel, E. Montocchio, E. Walters, S. Schalkwyk and S. Naidoo, *Setup minimising conditions in the trim loss problem*. European J. Oper. Res., **95** (1996), 631–640.

[5] H. Dyckhoff, *A typology of cutting and packing problems*. European J. Oper. Res., **44** (1990), 145–159.

[6] H. Dyckhoff and U. Finke, *Cutting and Packing in Production and Distribution: A Typology and Bibliography*. Springer-Verlag, Berlin Heidelberg, 1992.

[7] H. Foester and G. Wascher, *Pattern Reduction in One-dimensional Cutting-Stock Problems*. International Journal of Prod. Res., **38** (2000), 1657–1676.

[8] T. Gau and G. Wascher, *CUTGEN1: A Problem Generator for the Standard One-dimensional Cutting Stock Problem*. European J. Oper. Res., **84** (1995), 572–579.

[9] P.C. Gilmore and R.E. Gomory, *A Linear Programming Approach to the Cutting Stock Problem*. Oper. Res., **9** (1961), 849–859.

[10] P.C. Gilmore and R.E. Gomory, *A Linear Programming Approach to the Cutting Stock Problem*. Oper. Res., **11** (1963), 863–888.

[11] R. Haessler, *Controlling Cutting Pattern Changes in One-Dimensional Trim Problems*. Oper. Res., **23** (1975), 483–493.

[12] R. Haessler, *A Note on Computational Modifications to the Gilmore-Gomory Cutting Stock Algorithm*. Oper. Res., **28** (1980), 1001–1005.

[13] C.J. Hardley, *Optimal cutting of zinc-coated steel strip*. Oper. Res., **4** (1976), 92–100.

[14] N. Krejic and J.M. Martinez, *Validation of an Augmented Lagrangian algorithm with a Gauss-Newton Hessian approximation using a set of hard-spheres problems*. Comput. Optim. Appl., **16** (2000), 247–263.

[15] R.E. Johnston, *Rounding algorithms for cutting stock problems*. Asia-Pacic. J. Oper. Res., **3** (1986), 166–171.

[16] J.M. Martinez, *Minimization of discontinuous cost functions by smoothing*. Acta Applicandae Mathematical, **71** (2001), 245–260.

[17] A.C. Moretti and L.L. Salles Neto, *Modelo não-linear para minimizar o número de objetos processados e o setup num problema de corte unidimensional*. Anais do XXXVII Simpósio Brasileiro de Pesquisa Operacional, Gramado, (2005) 1677–1688.

[18] H. Stadler, *A one-dimensional cutting stock problem in the aluminium industry and its solution*. European J. Oper. Res., **44** (1990), 209–223.

[19] S. Umetami, M. Yagiura and T. Ibaraki, *One Dimensional Cutting Stock Problem to Minimize the Number of Different Patterns*. European J. of Oper. Res., **146** (2003), 388–402.

[20] F. Vanderbeck, *On Dantizg-Wolfe decomposition in integer programming and ways to perform branching in a branch-and-price algorithm*. Research Papers in Management Studies, University of Cambridge, 1994–1995, no. 29.

[21] F. Vanderbeck, *Computational Study of a Column Generation Algorithm for Bin Packing and Cutting Stock Problems*. Mathematical Programming, **86**(3) (1999), pp. 565–594.

[22] F. Vanderbeck, *Exact Algorithm for Minimizing the Number of Setups in the one-dimensional cutting stock problem*. Operations Research, **48** (2000), pp. 915–926.

[23] G. Wascher and T. Gau, *Heuristics for the Integer One-dimensional Cutting Stock Problem: a computational study*. OR Spektrum, **18** (1996), 131–144.