Revista Integración
Escuela de Matemáticas
Universidad Industrial de Santander
Vol. 28, No. 1, 2010, pág. 15–35

A Nonlinear Cutting Stock Problem

L. L. Salles Neto[a,*], A. C. Moretti[b]

[a] Universidade Federal de São Paulo - UNIFESP, Departamento de Computação, 12231-280 - São José dos Campos, São Paulo, Brasil.

[b] Universidade Estadual de Campinas - UNICAMP, Instituto de Matemática, Estatística e Computação Científica, 13084-790, Campinas, Sao Paulo, Brazil.

Abstract. In this work we introduce a new method to minimize the number of processed objects and the setup number in a unidimensional cutting stock problem. A nonlinear integer programming problem can be used to represent the problem studied here. The term related to the minimization of the setup number is a nonlinear discontinuous function, we smooth it and generate the cutting patterns using a modified Gilmore-Gomory strategy. Numerical tests on a wide range of test problems are very encouraging and the new method compares favorably with other methods in the literature.
Keywords: Cutting Stock Problem, nonlinear Programming, discontinuous Cost.
MSC2000: 65K05.

Un problema no lineal de archivo de corte

Resumen. En este trabajo presentamos un nuevo método para reducir al mínimo el número de objetos elaborados y el número de patrones de corte en un problema de corte unidimensional. Un problema de programación entera no lineal se puede utilizar para representar el problema estudiado. El término relacionado con la reducción al mínimo del número de patrones de corte es una función discontinua no lineal, la cual suavizamos y genera los patrones de corte utilizando una estrategia de modificación Gilmore-Gomory. Pruebas numéricas en una amplia gama de problemas fueron muy alentadores y el nuevo método se compara favorablemente con otros métodos en la literatura.
Palabras claves: Problema de archivo de corte, programación no lineal, costo discontinuo.

1. Introduction

In 1939 Kantorovich presented the first work in this subject and that was published in 1960 [20], with a formulation to minimize the trim loss in a unidimensional cutting stock problem. Similar problems were studied by other researchers, such as Paull and Walter

*Corresponding author: *E-mail*: luiz.leduino@unifesp.br.
Received: 21 April 2010, Accepted: 31 May 2010.

[27], Metzger [26] and Eilon [9]. However, in all of them, the authors developed strategies to work with small problems [6]. The papers of Gilmore-Gomory [12, 13] produced a great impact in this area because they proposed an efficient method to work with a large scale cutting stock problems through the use of column generation procedures. Since then, the study of cutting stock problems and their variants has played a great importance in the production planning scenario, helping decision makers in industries like paper, glass, chemical and textile production, among others.

The Unidimensional Cutting Stock Problem (1/V/I/R according to Dyckhoff [7]) is characterized by cutting stocks in just one dimension. More specifically, we have m items with different sizes, each one having its width w_i and we must cut, through its length, a minimum number of master rolls (with width $W > w_i$ for all i) to meet demand d_i for each item i for $i = 1, , \ldots, m$. Each combination of items to be cut in a master roll is called a cutting pattern. The problem is to determine the frequency of each cutting pattern to meet demand and (for instance) minimize the waste. A reasonable goal to be met in an industry is to minimize the number of master rolls (objects) used to produce the demanded items. If we consider that there are an infinite number of objects of same width, then the formulation below describes the mathematical model that minimizes the total number of objects used in a cutting plan:

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

where c_1 is the cost for each object; a_{ij} is the number of items i in cutting pattern j; x_j is the number of processed objects with cutting pattern j.

In some cases, minimizing the number of objects used is not the only goal for the manager. In fact, when we have a large demand that needs to be met in a short period of time, the number of machine setups 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 setups:

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

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 [25]. In particular $(\overline{P}_1)$ presents two conflicting objectives: (1) To minimize the total number of processed objects and (2) the total number of setup used. The difficulty in solving this problem is due to the fact that the objective function, besides being nonlinear, is discontinuous. This fact does not allow us to solve the problem by using the Gilmore-Gomory strategy. This is the reason why several papers considering this problem involve the use of heuristic procedures. Below, we describe four of these methods which are compared with our approach.

- **Sequential Heuristic Procedure - SHP**: it was proposed by Haessler [14] and it is based on an exhaustive technique of pattern repetitions. At each iteration an aspiration criterion is computed; then a search is done to look for cutting patterns that satisfy the parameters until the demands are all met. The SHP gives us a good initial solution and it is used by other methods to compare the quality of their solutions. It generates an inexpensive good initial solution to the $(\overline{P}_1)$. In this paper we introduced some modifications in this algorithm.

- **Kombi234**: This method was developed by Foester and Wascher [10] 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 [16]. Some methods based on this idea were published by Johnston [19], Allwood-Goulimis [1] and Diegel et al. [4]. All of them have in common the combinations of pairs or triples of cutting patterns, but they differ in the way the combinations are carried out. The method Kombi234 can be seen as a generalization of Diegel's method, in which the ideas of combining patterns are extended to a consistent system independently from the number of patterns to be combined. It makes use of the fact that the sum of the pattern frequencies of the resulting patterns has

to be identical to the sum of the frequencies belonging to the original patterns in order to keep the material input constant. The results presented by Foester and Wascher show that the setup was reduced by up to 60% in relation to the original cutting plan. Kombi234 was proved to be superior to SHP.

- **Hybrid Heuristic**: This method, proposed in Yanasse and Limeira [35], is a hybrid procedure composed of three phases. In the first phase, patterns are generated and the "good" ones are selected and used to reduce the problem; in the second phase, the reduced problem is solved and, in the third phase, a pattern reduction technique is applied. The authors argue that the computational tests performed indicated that the proposed scheme provides alternative solutions to the pattern reduction problem which are not dominated by other solutions obtained by using procedures previously suggested in the literature.

- **ILS**: Umetani et al. [31] presented a local search algorithm that uses two types of local search: (1) the 1-add neighborhood and (2) the shift neighborhood. Linear programming techniques were aggregated to the local search procedures to reduce the number of solutions in each neighborhood and to improve its performance. A sensitivity analysis technique was introduced to solve the large number of associated LP problems quickly. Umetani et al. (2006) compared ILS with KOMBI234, SHP and with an exact branch-and-price method (BP) proposed by Belov and Scheithauer [2], which proposed a method similar to the work of Vanderbeck [33], but, with a small number of variables. In [33], he investigates the problem of minimizing a number of different cutting patterns as a nonlinear integer programming, where a number of objects is fixed and determined after solving the problem by Gilmore-Gomory strategy. In this paper, Vanderbeck uses a Dantzig-Wolfe decomposition, developed in [32], to solve the resulting integer programming problem. Umetani et al. [31] claims that the algorithm ILS obtains better quality solutions than those obtained by the SHP, KOMBI234, BP approaches.

In this work, we solve a nonlinear cutting stock, whose optimal solution is a compromised solution between two conflicting objective functions: (1) minimize the number of objects used in a cutting plan and (2) minimize the number of setup number. A nonlinear formulation for this bi-objective problem was first proposed by Haessler [14], who did not try to solve the resulting nonlinear problem, instead he developed a nice and efficient sequential procedure to find good quality solutions for the problem. His formulation includes a nonlinear term in the objective function to represent the number of setup used in the cutting plan. This term, besides being nonlinear, is not a continuous function. We developed a strategy to solve this nonlinear problem by smoothing this nonlinear function

using a result from [24] and making an adaptation of the method of Gilmore-Gomory column generation to generate new profitable columns to the problem.

The paper is organized as follows. In Section 2 we show how to smooth a discontinuous function by making use of the theory developed in [24]. The adaptation of the Gilmore-Gomory Column Generation Method for a Nonlinear Programming Problem is presented in Section 3. Our approach for the Nonlinear Cutting Stock Problem is discussed in Section 4. The framework of the computational experiences and the results are found in Sections 5 and 6, respectively. And, finally, Section 7 presents the conclusion and perspectives of future works.

2. Smoothing a discontinuous cost function

Many practical problems require the minimization of functions that involves discontinuous costs. Among them we can cite process models with discontinuous investment costs and fixed charges [29], continuous review (S,Q) inventory systems with constant demand and batch arrivals [17], design of flow sheets for systems that satisfy fixed demand of steam [22]. Tools for solving them include Monte Carlo simulation, disjunctive programming approaches, simulated annealing, decomposition methods among others.

Martinez [24] proposes a smoothing method for the discontinuous cost function and establish sufficient conditions on this approximation that ensure that the smoothed problem really approximates the original one. Consider the problem

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

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

$$\inf \left\{ |\alpha_{il} - \alpha_{ij}| \ \text{ such that } \ l, j \in I_i, l \neq j \right\} > 0. \tag{2}$$

The side limits $\lim\limits_{t \to \alpha_{ij}^-} H_i(t)$, $\lim\limits_{t \to \alpha_{ij}^+} H_i(t)$, 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 $\mu > 0$,

$$\lim_{k\to\infty} H_{ik}(t) = H_i(t) \text{ uniformly in } \mathbb{R}\setminus \bigcup_{j\in I_i} (\alpha_{ij}, \alpha_{ij}+\mu). \tag{3}$$

Note that (3) implies that

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

For each k, we define the approximated problems as

$$\text{Minimize} \quad f(x) + \sum_{i=1}^{m} H_{ik}[g_i(x)] \quad \text{subject to} \quad x \in \Omega. \tag{4}$$

Since (4) has a continuous objective function, we can use continuous optimization algorithms to solve it. The following theorem proves that the solution of (1) can be approximated by the solution of (4).

Theorem 2.1. *Assume that for all $k = 0, 1, 2, \ldots$, x^k is a solution of* (4) *and that $x^* \in \Omega$ is a cluster point of $\{x^k\}$. Then, x^* is a solution of* (1).

The proof of this theorem can be found in Martinez (2001).

We adapt these ideas for the (P_1) and relaxing the integrality constraints we obtain the problem (P_2)

$$(P_2)\begin{cases} \text{Minimize} \quad f(x) + \sum_{i=1}^{n} H_i(x) \quad \text{subject to } \ x \in \Omega, \end{cases}$$

where:

- $\Omega = \{x \in \mathbb{R}^n \quad \text{such that} \quad Ax \geq d, \ x \geq 0\}$, where the columns of the matrix $A \in \mathbb{R}^{m\times n}$ correspond to cutting patterns, d is a vector containing the demands of the items and x is a vector where the j^{th}-component contains the number of time the j^{th}-patterns in used in the cutting plan.
- $f(x) = c_1 \cdot \sum_{i=1}^{n} x_i$;
- $H_i(t) = c_2\delta(t)$, $i = 1, \ldots, n$.

Note that $I_i = \{0\}$ for all $i = 1, \ldots, n$; hence 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 function H_i by the following continuous functions:

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

It 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 $P_2^{(k)}$ be the approximation of problem P_2 for different values of k:

$$(P_2^{(k)}) \begin{cases} \text{Minimize} & c_1 \cdot \sum_{j=1}^{n} x_j + c_2 \cdot \sum_{j=1}^{n} \dfrac{k x_j^2}{1+kx_j^2} \\ \text{subject to :} & \sum_{j=1}^{n} a_{ij} \cdot x_j \geq d_i,\ i = 1, \ldots, m, \\ & x_j \geq 0,\ j = 1, \ldots, n. \end{cases}$$

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 $P_2^{(k)}$ and x^* is the cluster point of the sequence $\{x^k\}$ then x^* is the solution of (P_2).

Therefore, for a given set of cutting pattern (i.e., columns) x^* is the solution of (P_2). After obtaining a solution for (P_2), we use Gilmore-Gomory method to verify if there a new profitable column. If so, we append this new column to (P_2) and solve $P_2^{(k)}$ for $k = 0, 1, 2, \ldots$ again. This process keeps going until there is not a profitable columns to append to (P_2).

The Gilmore-Gomory method was developed for linear programming problems, but, problem (P_2) is nonlinear. To overcome this difficulty we create an approximate problem which is linear. This is better explained in the next section.

3. Column generation in a nonlinear problem

The column generation procedure for linear programming problems developed by Gilmore-Gomory [12, 13], made it possible to solve large-scale cutting stock problem. Problems encountered in real life may involve a very large number of variables, instead working with all of them creating a huge problem, Gilmore-Gomory method creates an initial linear programming problem with only a few number of columns (i.e.,cutting patterns) and it keeps appending new columns to the original problem only when these columns are profitable. That is, each iteration of the Gilmore-Gomory method solves a linear programming problem, the first iteration starts with a problem with a few numbers of columns. After solving this first problem, the method generates a new column

obtaining by solving a particular knapsack problem. The objective function value of the knapsack problem is the reduced cost of the generated column. If the new column is profitable then it is appended to the previous problem and the process continues until no more profitable columns are generated. See Gilmore-Gomory [12, 13] for more details of the method.

In this section we show how we applied the column generation procedure in a nonlinear problem. This was done by creating an auxiliary linear programming problem that is related to our nonlinear problem (P_2). By related we mean that the solution of (P_2) is a solution for the linear problem.

Consider the following linear programming problem:

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

where NCP denotes the number of different cutting patterns in the solution obtained. We use (P_3) to generate a new column for the original problem (P_2), this was done by solving a particular knapsack Problem. Haessler suggests in his paper to solve a bounded knapsack problem to generate profitable columns in the sense of reducing trim loss and setup number. In his work, Haessler suggests to fix the upper bounds in the knapsack problem in the following way:

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

We also solve a Bounded Knapsack Problem, but, we compute the upper limits in a different way. This is explained in the next section.

4. Solving the nonlinear cutting problem

Below, we describe the main steps of the algorithm for solving the nonlinear cutting stock problem, having in mind that our original problem is the one denoted by $(\overline{P}_1)$, but we work with the approximated problem (P_2). To solve problem (P_2), for a given set of columns, we used a sequence of problems (P_2^k). After solving the problem we append profitable columns. If there are no more profitable columns, the algorithm stops. If the last solution obtained is integer we are done, if not we applied a heuristic to obtain an integer solution.

Algorithm - ANLCP

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

Step 2: Obtain a solution for the current (P_2) solving P_2^k for $k = 10^i$, $i = 1, 2, \ldots 5$ using x^* as an initial point;

Step 3: If the solution obtained in (2) is better than x^*, update it;

Step 4: Get the simplex multiplier π_i, $i = 1, \ldots, m$ by solving (P_3);

Step 5: Solve a Bounded Knapsack Problem with the objective function coefficients given by the simplex multiplier of (P_3):

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

Step 6: If $Z \leq 1$ Solve the Knapsack Problem with no limits in the variables;

Step 7: If $Z \leq 1$ = STOP. Otherwise, add the new column y into problem (P_2) and go back to Step (2).

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

To obtain an initial solution, we implemented the SHP method with some modifications

- Instead of fixing the parameter *MAXTL*, the maximum allowable trim loss, we fix the value of *MAXTL* equal to $MAXTL = 0.01W$, in the beginning of the process and for each generated cutting pattern we increase its value by 0.25%, that is, $MAXTL = MAXTL + 0.0025W$ until it reaches its upper limit defined by 3% of W.

- We did the same for the parameter *MINU*, the minimum number of production rolls to be processed according to the pattern. In this case, we start with $MINU = 0.5NR$ and for each generated pattern we add $0.05NR$ to it until the maximum of $0.9NR$ (where NR is an estimative of the number of stock rolls used, see (Haessler 1975). The reason for that is simple, higher values of *MINU* generate smaller *setup* and bigger *trim loss*. Therefore, starting with a small value for *MINU* increases the search space in the first iterations allowing the method to look for cutting pattern which will produce small *trim loss*. And, as we increase the parameter value in each iteration the procedure will look for cutting patterns that will produce small *setup* without spoiling the *trim loss* value.

- In the SHP algorithm the search is conducted by generating patterns in lexicographically decreasing order that satisfies the current aspiration level. In our implementation, we allocate, in the current pattern, the biggest possible value of the item with the higher residual demand. We do the same with the item which has the second biggest value and so on, without violating the length of the stock.

  If the generated pattern does not satisfy the aspiration criterion, we decrease by one unit the frequency of item with higher residual demand and we return to the search. By doing this, the final cutting plan has a small number of different cutting pattern (i.e., setup number).

The parameter *MINR* (the minimum number of rolls permitted in the pattern) is fixed according to Haessler. The parameter *MAXR*, the number of knives, was not used in our implementation. For more details about SHP see [14].

The software `BOX9903` [21] was used to solve the sequence of nonlinear problems. This routine solves the following 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 $l = 0$ and $u = \infty$. 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. For iteration j we define $\rho = K = 10^j$, where j assume the values $1, 2, 3, 4$ and 5, that is, we realize 5 iterations of the method. The Lagrange multiplier is estimated in each iteration j. Keep on mind that $h_{ik}(x) = kx_i^2/(1+kx_i^2)$.

Since this is a local method, each time a column is added to the problem we solve the sequence of problems P_k with 20 different initial points: the current solution, the null solution and 18 random generated points.

After solving the sequence of problems and obtaining the best solution for the new problem, we solve a Bounded Knapsack Problem to find if there is a profitable column

(cutting pattern) to append to the problem (P_2). To do so, we obtain the simplex multiplier from problem (P_3) and use them as the coefficients of the knapsack problem objective function. Since we are working with a bounded knapsack problem we need to define the limits of the components of the current solution. Let the setup number be equal NCP, $NCP \leq m$, that means that we have NCP different cutting patterns. When we add a column we are interested in reducing not only the trim loss, but also the setup number. Assuming we want to reduce the current setup number by 20%, therefore, after adding the new generated column to (P_2) we hope to obtain a new setup number that is 20% smaller than the current setup number, i.e., $NewSetup = 0.8 * NCP$. Let NB be the number of production rolls processed in the current solution. Supposing that this number remains constant, we should have, on average, $(MBP = NB/NewSetup)$ processed object per cutting pattern. And, assuming the new generated pattern will belong to the solution with frequency equal to MBP and the items in this pattern will not belong to the nonzeros patterns, then to guarantee a feasible solution, each item in this pattern must be limited by $b_i = d_i/MBP$. Since we need an integer upper bound, we make $b_i = \lfloor b_i \rfloor$. And, if $b_i > W/w_i$ then we fix $b_i = \lfloor W/w_i \rfloor$ so we will obtain only feasible patterns.

Finally, to round the solution in the end of the process we used the BRURED method described in [34]; this method does not modify the setup number and it is efficient. 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. That is, for each $k \in \{1, 2, \ldots, n\}$, if the variable x_k, after the round up, satisfies the inequality

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

If this is the case then we fix $x_k^* = x_k - 1$.

5. Computational experiments

In order to evaluate our approach, random instances for the one dimensional cutting problem were generated by CUTGEN1, developed by Gau and Wascher [11]. As in the paper [10], we generated 18 problem classes by combining different values of the CUTGEN1's parameters

- v_1 values 0.01 and 0.2;
- v_2 values 0.2 and 0.8;

- the number of patterns in the original cutting plan, denoted by m set to 10 (small instances), 20 (mid-size instances) and 40 (large instances).

- problem classes have low average demand ($\bar{d} = 10$) and high average demand ($\bar{d}$= 100)

We obtained 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 $v_2 = 0.8$). For each class, we generated and solved 100 instances. Table 1 shows the parameters used for each class.

Class	v_1	v_2	m	d
1	0.01	0.2	10	10
2	0.01	0.2	10	100
3	0.01	0.2	20	10
4	0.01	0.2	20	100
5	0.01	0.2	40	10
6	0.01	0.2	40	100
7	0.01	0.8	10	10
8	0.01	0.8	10	100
9	0.01	0.8	20	10
10	0.01	0.8	20	100
11	0.01	0.8	40	10
12	0.01	0.8	40	100
13	0.2	0.8	10	10
14	0.2	0.8	10	100
15	0.2	0.8	20	10
16	0.2	0.8	20	100
17	0.2	0.8	40	10
18	0.2	0.8	40	100

Table 1. Random Generated Classes and their parameters.

We compare our results with the results of SHP [14], KOMBI234 [10], Hybrid Heuristic [35] and ILS [31]. The comparison with ILS was done with the results obtained without an upper bound in the number of objects. KOMBI234 uses the solution obtained by the heuristic developed by Stadler [28] as initial solution. According to Foester and Wascher, the combination *Stadler + KOMBI234* produced the best results encountered in the literature at the time they published their work.

We run CUTGEN1 with the same seed (i.e.,1994) and parameters that were defined in [10], [30, 31] and [35] to generate all the 1800 (i.e., 18 classes, each class with 100 instance problems) instance problems.

The method ANLCP and the heuristic SHP were implemented by the authors in Fortran, running under g77 for Linux, with a processor Athlon XP 1800Mhz, 512MB of RAM. The results of KOMBI234, ILS and Hybrid Heuristic were taken from [10], [31] and Yanasse and Limeira [35], respectively. The results for KOMBI234 were obtained from an implementation in MODULA-2 in MSDOS operating system with an IBM 486/66. ILS was coded in C language and run on an IBM compatible personal computer (Pentium IV 2 GHz, 1 GB memory) under Linux. Hybrid Heuristic (HH) were performed in C++ in a microcomputer Intel Celeron, 266 MHz, 128MB RAM and a workstation Sun Ultra 30, 296MHz 384MB RAM.

In our computational experiments we are looking for solutions with small setup number. This is an advantage of the ANLCP. By using as a penalization parameter instead the real life value of the setup cost, we can control the setup number. Next section, we presented the results obtained by ANLCP using $c_2 = 100$, that is, c_2 used as a penalization parameter since real life values would lie in the interval [1,10]. However, when we compare the solutions obtained by ANLCP with the solutions obtained by the other methods, we use the values $c_2 = 1, 5, 10$ in the objective function. That is, although we obtain solutions by using $c_2 = 100$, when we compute the objective functions values to compare with the other solutions we use the real life values for c_2.

6. Computational results

Tables 2, 3 and 4 show the average for the setup number and for the number of objects used in the final solution for the 100 instances in all the classes: small items, widespread items and large items.

Class	SHP		Kombi		HH		ILS		ANLCP	
	Setup	Objects	Setup	Objects	Setup	Objects	Setup	Objects	Setup	Objects
1	3.95	14.17	3.40	11.49	3.31	11.56	1.67	15.15	3.14	18.44
2	5.94	116.47	7.81	110.25	6.95	110.4	1.67	149.78	4.66	116.66
3	5.00	25.29	5.89	22.13	4.96	22.17	2.57	28.01	4.88	25.18
4	7.31	225.33	14.26	215.93	10.32	215.98	2.57	278.57	7.16	226.72
5	6.87	46.89	10.75	42.96	7.63	42.99	4.28	55.12	7.02	45.64
6	10.81	433.59	25.44	424.71	13.31	424.89	4.28	546.64	10.96	432.68
Average of the averages	6.65	143.62	11.26	137.91	7.75	138.00	2.84	178.88	6.30	144.22

Table 2. Averages for small items.

Table 5 presents the variation of setup and number of objects of the method ANLCP in relation to SHP. To compute the variation for the setup number we use the formula $100 \times (Setup_{ANLCP} - Setup_{SHP})/Setup_{SHP}$. The average setup in ANLCP method was

Class	SHP		Kombi		HH		ILS		ANLCP	
	Setup	Objects	Setup	Objects	Setup	Objects	Setup	Objects	Setup	Objects
7	8.84	55.84	7.90	50.21	7.66	51.69	5.01	54.14	5.78	50.84
8	9.76	515.76	9.96	499.52	9.62	502.23	5.01	541.50	8.22	506.02
9	17.19	108.54	15.03	93.67	13.64	99.49	9.27	101.21	10.90	106.72
10	19.37	1001.59	19.28	932.32	18.21	948.41	9.27	1008.05	14.56	969.40
11	32.20	202.80	28.74	176.97	24.60	195.67	16.95	193.17	19.80	220.46
12	37.25	1873.05	37.31	1766.20	33.23	1847.42	16.95	1920.39	25.58	1813.60
Average of the averages	20.77	626.26	19.70	586.48	17.83	607.49	10.41	636.41	14.14	611.17

Table 3. Averages for wide-spread items.

Class	SHP		Kombi		HH		ILS		ANLCP	
	Setup	Objects	Setup	Objects	Setup	Objects	Setup	Objects	Setup	Objects
13	9.38	69.97	8.97	63.27	8.93	64.20	6.26	67.61	5.86	66.52
14	9.85	643.55	10.32	632.12	10.51	633.26	6.26	675.50	7.92	633.98
15	18.03	136.03	16.88	119.43	16.28	123.90	11.76	125.86	10.28	130.20
16	19.63	1253.55	19.91	1191.80	19.89	1197.66	11.76	1256.92	15.00	1193.54
17	34.39	256.01	31.46	224.68	29.76	244.02	21.50	239.64	23.32	283.88
18	38.23	2381.54	38.28	2342.40	37.90	2268.30	21.50	2391.53	29.80	2410.82
Average of the averages	21.59	790.11	20.97	762.28	20.55	755.22	13.17	792.84	15.36	786.49

Table 4. Averages for large items.

better than the average setup for SHP in all the classes, except for classes 5 and 6. And, the average number of objects in ANLCP method was better than the average obtained by SHP in 12 classes out of 18.

Table 6 presents the variation of setup and the number of objects of our method (ANLCP) in relation to KOMBI234. In all the classes ANLCP obtained a better average for the setup than KOMBI. But, the average number of objects used by KOMBI was better than ANLCP in all the 18 classes. Still, in the 7 classes the difference was smaller than 3%.

Table 7 presents the variation of setup and the number of objects of our method (ANLCP) in relation to HH. In all the classes ANLCP obtained a better average for the setup than HH. In two classes ANLCP obtained better averages than HH for the number of the number of objects used in the cutting plan.

Table 8 presents the variation of setup and the number of objects of our method (ANLCP) in relation to ILS. ANLCP obtained better averages than ILS for the number of setup in two classes. For the number of objects, ANLCP was better than ILS in twelve classes.

Class	Setup	Number of objects
1	−20.51	30.13
2	−21.55	0.16
3	−2.40	−0.43
4	−2.05	0.62
5	2.18	−2.67
6	1.39	−0.21
7	−34.62	−8.95
8	−15.78	−1.89
9	−36.59	−1.68
10	−24.83	−3.21
11	−38.51	8.71
12	−31.33	−3.17
13	−37.53	−4.93
14	−19.59	−1.49
15	−42.98	−4.29
16	−23.59	−4.79
17	−32.19	10.89
18	−22.05	1.23

Table 5. Variation in % of ANLCP in relation to SHP.

Class	Setup	Number of objects
1	−7.65	60.49
2	−40.33	5.81
3	−17.15	13.78
4	−49.79	5.00
5	−34.70	6.24
6	−56.92	1.88
7	−26.84	1.25
8	−17.47	1.30
9	−27.48	13.93
10	−24.48	3.98
11	−31.11	24.57
12	−31.44	2.68
13	−34.67	5.14
14	−23.26	0.29
15	−39.10	9.02
16	−24.66	0.15
17	−25.87	26.35
18	−22.15	2.92

Table 6. Variation in % of ANLCP in relation to Kombi.

We observed by the computational experiences that no method dominated any other method in all the classes. That is, no method obtained the smallest number of objects and setup number in all the classes. More specifically, ANLCP dominated SHP in 10 out of 18 classes and it is not dominated in any class. There is not a dominance relation between ANLCP and KOMBI234 in any one of the classes. ANLCP dominated HH in three classes and it is not dominated in any class. Finally, ANLCP dominated ILS in one class and it is dominated in 5 classes.

We consider an advantage of ANLCP over the other methods the fact that with ANLCP we can work with the costs of setup in the objective function. Therefore, we can control the value of (as a penalization parameter) such that the cutting patterns generated take in account number of objects and setup number and not only one of them.

To compare the quality solution of each method in terms of the objective function values, we use the values of c_2 equal to 1,5 and 10. Those are real life values for c_2 and by doing so, the comparisons with the others methods are more fair. In general, ANLCP method obtained better objective function values that SHP, KOMBI234, HH and ILS. In the literature, Diegel et al. [5] were the only to mention about practical values for c_1 and c_2. According to Diegel et al.[5], an exact relation between c_1 and c_2 depends on

Class	Setup	Number of objects
1	−5.14	59.52
2	−32.95	5.67
3	−1.61	13.58
4	−30.62	4.97
5	−7.99	6.16
6	−17.66	1.83
7	−24.54	−1.64
8	−14.55	0.75
9	−20.09	7.27
10	−20.04	2.21
11	−19.51	12.67
12	−23.02	−1.83
13	−34.38	3.61
14	−24.64	0.11
15	−36.86	5.08
16	−24.59	−0.34
17	−21.64	16.33
18	−21.37	6.28

Table 7. Variation in % of ANLCP in relation to HH.

Class	Setup	Number of objects
1	88.02	21.72
2	179.04	−22.11
3	89.88	−10.10
4	178.60	−18.61
5	64.02	−17.20
6	156.07	−20.85
7	15.37	−6.10
8	64.07	−6.55
9	17.58	5.44
10	57.07	−3.83
11	16.81	14.13
12	50.91	−5.56
13	−6.39	−1.61
14	26.52	−6.15
15	−12.59	3.45
16	27.55	−5.04
17	8.47	18.46
18	38.60	0.81

Table 8. Variation in % of ANLCP in relation to ILS.

the data we have on hands. But they say that c_2 is never much bigger than c_1. Also, they say that $c_1 = c_2 = 1$ are appropriate for large problems. 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 costs depends on several factors as: demand, deadlines, labor costs, etc.

For a better understanding of the behavior of each method, Table 9 shows the difference in percentage of the average costs obtained by ANLCP when compared with SHP, KOMBI234, HH and ILS for $c_1 = c_2 = 1$.

Observe that ANLCP obtained the best averages in 14 classes when compared with SHP, in 6 classes when compared with Kombi234, in 5 classes when compared with HH and in 12 classes when compared with ILS, even when $c_1 = c_2 = 1$.

The results obtained by ANLCP when $c_1 = 1$ and $c_2 = 5$ are presented in Table 10. The averages for the objective function values for ANLCP were better than those obtained by SHP in all 18 classes, better than KOMBI234 in 12 classes, better than HH in 12 classes and better than ILS in 10 classes.

The computational results confirm the good performance of ANLCP method. The

ANLCP, as shown is Table 11, obtained average costs better than SHP in 16 classes, better than KOMBI234 in 17 classes, better than HH in 15 classes and better than ILS in 9 classes when $c_1 = 1$ and $c_2 = 10$.

7. Conclusions and future work

Based on the computational results presented, we may say that the ANLCP method is, in fact, competitive in relation to SHP, KOMBI234, HH and ILS approaches. When the total cost ($c_1\times$ (*number of objects*) + $c_2\times$ (*setup*)) is used as a basis of comparison, ANLCP obtain averages 3% better than the ones obtained by SHP, KOMBI234, HH and ILS in several classes and usually in all of them when $c_1 = 1$ and $c_2 = 5$ or 10. The ANLCP method is competitive even when $c_1 = c_2 = 1$.

Considering only the *setup number*, ANLCP has a better performance than SHP, KOMBI234 and HH in almost all the classes.

It is important to say that another advantage of ANLCP method is the possibility of working with explicitly values of c_1 and c_2 in the objective function. In fact, in real life problems those costs depend on many factors as demand, deadline, labor costs, etc.

We do not know any other method that treats the problem of minimizing the setup and the number of processed object in the same way ANLCP method does.

However, ANLCP method has a disadvantage in relation to SHP, KOMBI234, HH and ILS : the computational time. Although we cannot compare the computational time with KOMBI234, ILS and HH, since they were not implemented by the authors, but it is easy to see that the ANLCP method has a higher elapsed time when compared with the other methods. This happens because once a new column is added to the problem, we need to solve (P_2^k) with 20 initial points. For classes with a large number of items, as the Classes (11,12,17,18), the computational time for ANLCP was high.

Testing better strategies for solving nonlinear problems to obtain global solutions can make ANLCP better. Also, a better strategy to round the fractional solutions is welcome since the BRURED method used for rounding the solution is very simple; however it is quick and it keeps the setup number found by ANLCP.

Acknowledgments

The authors have been partially supported by M.E.C. (Spain), Project MTM2007-063432. The second author is also supported by CNPq (Brazil), Project 307907/2007-4.

Class	SHP	Kombi	HH	ILS
1	19.09	44.93	45.12	28.30
2	−0.89	2.76	3.38	−19.89
3	−0.76	7.28	10.80	−1.70
4	0.53	1.60	3.35	−16.81
5	−2.05	−1.95	4.03	−11.35
6	−0.17	−1.45	1.24	−19.47
7	−12.46	−2.56	−4.60	−4.28
8	−2.15	0.93	0.47	−5.90
9	−6.45	8.21	3.97	6.46
10	−3.62	3.40	1.79	−3.28
11	2.24	16.80	9.08	14.34
12	−3.72	1.98	−2.21	−5.07
13	−8.78	0.19	−1.03	−2.02
14	−1.76	−0.08	−0.29	−5.85
15	−8.81	3.06	0.21	2.08
16	−5.08	−0.26	−0.74	−4.74
17	5.79	19.93	12.21	17.64
18	0.86	2.52	5.83	1.14

Table 9. Variation in % of the total cost, with $c_1 = c_2 = 1$, of ANLCP in relation to the SHP, Kombi, HH and ILS.

Class	SHP	Kombi	HH	ILS
1	0.65	19.83	21.45	45.28
2	−4.25	−6.26	−3.58	−11.49
3	−1.41	−3.88	5.56	21.34
4	0.24	−8.60	−1.89	−9.92
5	−0.62	−16.51	−0.49	5.51
6	−0.03	−11.67	−0.81	−14.18
7	−20.29	−11.11	−11.39	0.69
8	−3.09	−0.40	−0.58	−3.43
9	−17.11	−4.50	−3.86	9.26
10	−5.12	1.31	0.26	−1.16
11	−12.19	−0.38	0.25	14.95
12	−5.72	−0.58	−3.58	−3.17
13	−18.01	−11.38	−11.97	−3.12
14	−2.77	−1.48	−1.78	−4.70
15	−19.71	−10.91	−11.54	−1.66
16	−6.15	−1.77	−2.20	−3.59
17	−6.42	4.84	1.95	15.37
18	−0.50	1.03	4.15	2.43

Table 10. Variation in % of the total cost, with $c_1 = 1$ and $c_2 = 5$, of ANLCP in relation to the SHP, Kombi, HH and ILS.

Class	SHP	Kombi	HH	ILS
1	−7.14	9.56	11.60	56.48
2	−7.17	−13.32	−9.25	−1.93
3	−1.74	−8.70	3.08	37.74
4	−0.04	−16.79	−6.54	−1.96
5	0.22	−23.01	−2.89	18.30
6	0.11	−20.15	−2.82	−8.00
7	−24.68	−15.92	−15.32	4.22
8	−4.10	−1.82	−1.71	−0.57
9	−23.08	−11.58	−8.55	11.25
10	−6.72	−0.90	−1.37	1.29
11	−20.26	−9.89	−5.26	15.38
12	−7.84	−3.27	−5.06	−0.98
13	−23.60	−18.21	−18.49	−3.91
14	−3.89	−3.01	−3.41	−3.38
15	−26.34	−19.16	−18.73	−4.30
16	−7.33	−3.40	−3.80	−2.25
17	−13.81	−4.12	−4.53	13.73
18	−1.99	−0.60	2.32	3.92

Table 11. Variation in % of the total cost, with $c_1 = 1$ and $c_2 = 10$, of ANLCP in relation to the SHP, Kombi, HH and ILS.

Class	SHP T(s)	Kombi* T(s)	HH* T(s)	ILS* T(s)	ANLCP T(s)
1	0.01	0.14	0.23	0.10	0.80
2	0.08	1.14	0.48	0. 22	1.17
3	0.17	1.74	0.12	0.72	0.47
4	0.21	16.00	2.75	2.69	0.94
5	0.27	38.03	3.43	7.55	0.58
6	0.31	379.17	7.81	23.18	0.93
7	0.01	0.07	0.11	0.21	16.49
8	0.02	0.2	0.60	0.27	8.96
9	0.04	3.37	0.49	1.96	69.11
10	0.06	3.25	3.36	2.19	77.21
11	0.22	36.26	7.17	19.16	185.53
12	0.32	76.31	44.62	23.87	318.54
13	0.01	0.08	0.13	0.26	4.44
14	0.02	0.13	0.25	0.31	1.95
15	0.03	1.81	0.97	2.01	29.36
16	0.04	2.6	2.46	2.21	26.30
17	0.16	50.93	15.46	22.01	248.62
18	0.24	70.94	50.61	26.84	443.66

Table 12. Average time (in seconds) of each method (* KOMBI, HH and ILS were not implemented and tested in the same computational environment).

References

[1] Allowod J.M., and Goulimins C.N., "Reducing the number of patterns in one-dimensional cutting stock problems," *Control Section Report* No EE/CON/IC/88/10, Industrial Systems Group, Department of Electical Engineering, Imperial College, London, 1988.

[2] Belov G., Scheithauer G., "The Number of Setups (Different Patterns) in One-Dimensional Stock Cutting," *Technical Report MATH-NM*-15-2003, Dresden University, 2003.

[3] Chvatal V., "Linear Programming," Freeman, San Francisco, 1983.

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

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

[6] Dowsland K. and Dowsland W., Packing Problems, *European J. Oper. Res.*, 56 (1992), 2-14.

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

[8] Dyckhoff H., and Finke U., "Cutting and Packing in Production and Distribution: A Typology and Bibliography," Springer-Verlag Co, Heidelberg, 1992.

[9] Eilon S., Optimizing the shearing of steel bars, *J. Mech. Eng.* Sci. 2 (1960), 129-142.

[10] Foester H., and Wascher G., Pattern Reduction in One-dimensional Cutting-Stock Problems, *Internat. J. Prod. Res.*, 38 (2000), 1657-1676.

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

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

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

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

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

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

[17] Heuts R., and Deklein J., An (S-Q) inventory model with stochastic and interrelated lead times, *Naval Res. Logist.*, 42 (1995), 839-859.

[18] Hinxman A., The trim loss and assortment problems: a survey, *European J. Oper. Res.*, 5 (1980), 8-18.

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

[20] Kantorovich L.V., Mathematical Methods of Organizing and Planning Production, *Management Sci.*, 6 (1960), 366-422.

[21] Krejic M., Martinez J.M. et al., 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.

[22] Maia L., Valério de Carvalho L.A., Quassim R., Synthesis of utility systems by simulated annealing, *Comp. Chemical Eng.*, 19 (1995), 481-488.

[23] Martello S., Toth P., *Knapsack Problems: Algotithms and Computer Implementations*, John Wiley & Sons, New York, 1990.

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

[25] McDiamird C., Pattern minimisation in cutting stock problems, *Discrete Appl. Math.*, 98 (1999), 121-130.

[26] Metzger R.W., *Stock Slitting. Elementary Mathematical Programming*, Wiley, 1958.

[27] Paull A.E., and Walter J.R., "The trim problem: an application of linear programming to the manufacture of news-print paper, *Presented at Annual Meeting of Econometric Society*, Montreal, 10-13, 1954.

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

[29] Turkay M., and Grossmann I.E., Disjunctive Programming Techniques for the Optimization of Process Systems with Discontinuous Investment Costs-Multiple Size Regions, *Ind. Eng. Chem. Res.*, 35 (1996), 2611-2623.

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

[31] Umetani S., and Yagiura M., One Dimensional Cutting Stock Problem with a Given Number of Setups: A Hybrid Approach of Metaheuristics and Linear Programming, *Journal of Mathematical Modelling and Algorithms*, 5 (2006), 43-64.

[32] Vanderbeck F., Computational study of a column generation algorithm for bin packing and cutting stock problems, *Math. Program.*, 86 (1999), 565-594.

[33] Vanderbeck F., Exact Algorithm for Minimising the Number of Setups in the One-Dimensional Cutting Stock Problem, *Oper. Res.*, 48 (2000), 915-926.

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

[35] Yanasse H.I., and Limeira M., A hybrid heuristic to reduce the number of different patterns in cutting stock problems, *Comput. Oper. Res.*, 33 (2006), 2744-2756