Determining Cutting Stock Patterns when Defects are Present

Ronny Aboudi* and Paulo Barcia**

Working Paper n° 289
December, 1996

* University of Miami
** Universidade Nova de Lisboa

The analyses, opinions and findings of this paper present the views of the authors, they are not necessarily those of "Faculdade de Economia".
Determining Cutting Stock Patterns when Defects are Present

Ronny Aboudi(*) and Paulo Barcia(**)

ABSTRACT

This paper investigates the problem of obtaining the best arrangement for a cutting stock patterns in the presence of defective material in the paper, which is a common problem in the industry. This problem is modeled as a multiple subset sum problem and its structure is exploited in order to obtain a practical solution procedure. The solution methodology employs the subset sum problem as a subprocedure. Computational results are reported.

(*) University of Miami, Department of Management Science, POB 248237, Coral Gables, FL, 33124, USA. (e-mail: raboudi@umiami.ir.miami.edu)

(**) Universidade Nova de Lisboa, Department of Economics and Management, Travessa Estevão Pinto, Campolide, P-1070 Lisboa, Portugal. (e-mail: barcia@fe.unl.pt)
1 Introduction

The problem of manufacturing paper rolls and slitting them to appropriate widths is quite complex and varies among paper mills. The basic operation of a paper machine, common to all plants, is to convert a pulp mixture into a roll of a standard width (the width depends on the particular paper machine and can range from 120-280 inches). The basic products consist of either rolls or sheets of smaller widths (ranging in width from 8\(\frac{1}{2}\) to 30 or 40 inches). Obviously, sheets must be slit length-wise (vertically), but sometimes rolls too must be slit length-wise as the demand might consist of rolls ranging in 100-400 feet in length. Some rolls or sheets may be then processed by another machine if they require, for example, to be coated.

The multitude of paper products demanded by clients necessitates paper mills to possess paper machines able to performing various tasks. Some are capable to slit the standard rolls into rolls with smaller widths and lengths, others are capable of producing sheets. The number of knives and the minimum distance between knives is machine dependent. Sometimes the main paper machine is not capable of slitting widths that are too narrow and a secondary machine must process a paper roll (rather than pulp) into the products with the desired dimensions.

The cutting-stock problem, in its simplest form, (see Gilmore and Gomory (1961)) is to determine the most economical way of slitting standard-width rolls into rolls with smaller widths (but with the same length as the standard one) where both the specific widths and quantities are specified a priori. Since the lengths of the rolls are assumed to be the same as the standard roll, these models typically assume that each pattern is used to slit the entire roll.

In reality the problem is more complex, (see Haessler and Sweeney (1991) and Sweeney and Paternoster (1991)). First, it is desirable to achieve the best possible yield but also minimize the number of cutting patterns. The reasoning is two-fold: the first each additional pattern requires a new set-up of the knives that causes additional waste in term of paper and time and the second that each setup might introduce human error in the exact determination of the knives. In some paper mills the assumption that a particular pattern is used for the entire length of a given roll is invalid. At times, when the demands are small, several setups (distinct patterns) will be used on a single roll of length say 10,000 feet. Otherwise there might be unacceptable amount of over or under production.

Ideally, the standard roll produced by the machine from pulp is defect-free and the entire roll can be used to produce final products demanded by the clients. However, it is often the case that some parts of the standard
roll might be defective. The defects usually consist of stains or streaks. Some paper mills have the capability of scanning a standard paper roll and determining the defective area a priori. Since the paper is slit in straight horizontal and vertical lines, it suffices to denote the defective area as a rectangle (even though obviously, the physically damaged area is never an exact rectangle). There are two types of defects, one is "cut-into" and the other is "cut-around". A cut-into defect is such that the area can be processed by a sheeter and the sheets that are stained are thrown away. The cut-around defects, as its name implies, neither horizontal nor vertical slits are allowed in that area. Thus, one has to slit the standard width roll with a pattern that entirely avoids the defective area.

As an example, consider the following roll shown in figure 1:

Figure 1

One possible solution would be to use one pattern up to the defective area (from 0 to $d_1$), then another one for the defective area (from $d_1$ to $d_2$) and yet a third one past the defective area. However, since a new pattern requires a new setup for the knives which produces a waste in terms of paper (40-50 ft.) and also in term of time. An alternative is to use one pattern that cuts into the defective area but continues beyond it, but any stained final product is discarded. This gives rise to the following question: "Given a slitting pattern and a "cut-into" area, how should the sizes be permuted in order to minimize the discarded final products?". This question is addressed in this manuscript. This problem, arising from the complex process of paper manufacturing, is shown to be NP-complete and its modeling and implementation aspects are very interesting. Section 2 provides the assumptions and an appropriate model. In Section 3 the solution techniques are described. The implementation and computational results are given in Section 4.
2 Model for The Minimum Defective Subset Sum

This section will describe the problem and assumptions. It will be shown that the problem is NP-complete. The first step for successfully solving an NP-complete problem is to provide a good model for it. An integer programming model for this problem will be given and some of its limitations will be discussed.

Suppose that the width of the standard size is $c$ and suppose that the defective area lies in a rectangle with the length coordinates $d_1, d_2$, and width coordinates $a, b$. The pattern under consideration is denoted by the sizes (not necessarily distinct) of $a_1, a_2, ..., a_n$. We assume that the sum of the sizes does not exceed $c$. Furthermore, let $\bar{c} = \sum_{i=1}^{n} a_i$. We assume that $c \geq \bar{c} \geq a + (c - b)$. This assumption guarantees that some items will have to intersect the interval $(a, b)$ otherwise the problem is uninteresting as all the items will be placed in the defect-free zone. Note that our computational method does not require this assumption, however, with this assumption some of the proofs to the propositions are more intuitive (even though they can be modified appropriately when this assumption is removed). The problem is to determine the best possible permutation of the vector $(a_1, a_2, ..., a_n)$ so that when the pattern is arranged according to the permutation and the roll is slit length-wise beyond $d_1$ feet, the area that is discarded is minimized. Note that a sheet with any defective area is discarded in its entirety.

At first glance this problem seems to be two-dimensional. However, typically, when the pattern consists of products that are all sheets, the length dimension of each sheet is common for all of the products in a given feasible pattern i.e. the products consist of sheets of sizes $a_i \times \ell$, for $i = 1, 2, ..., n$. Using this fact simplifies the problem to a one-dimensional problem. Since any sheet that has any defect is discarded, the "cost" of a permutation is defined by the sum of all sizes of the orders that have a non-empty intersection with the interval $(a, b)$. The problem then reduces to determining a permutation of the vector of sizes that minimizes the cost. We refer to this problem as the minimum defective subset sum (MDSS).

Proposition 1: The minimum defective subset sum is NP-complete.

Proof. Let $a = 0$. In this case minimizing the intersection with the interval $(a, b)$ is equivalent to maximizing the utilization of the non-defective interval of size $c - b$ which is an instance of the subset sum problem. Thus, any subset sum problem can be reduced to MDSS, therefore, the result is
immediate because the subset sum problem is \( NP \)-complete (see Gary and Johnson (1979)). ■

One can immediately observe that the key element of this problem is determining which items intersect the defective interval \((a, b)\) and the exact permutation of the items does not have a significant role. Thus one can formulate an integer programming problem to minimize the total intersection with that interval, or equivalently, maximize the utilization of the non-defective intervals. Therefore, one can consider the following 0-1 integer programming problem. Define Boolean variables \( x_i^A \in \{0, 1\} \) such that \( x_i^A = 1 \) if item \( i \) is placed entirely in the interval \([0, a]\) and \( x_i^A = 0 \) otherwise. In a similar way define Boolean variables \( x_i^B \in \{0, 1\} \) such that \( x_i^B = 1 \) if item \( i \) is placed entirely in the interval \([b, c]\) and \( x_i^B = 0 \) otherwise. Let \( N = \{1, 2, \ldots, n\} \).

MDSS can be formulated as follows:

\[
V^* = \max \sum_{i \in N} a_i \left( x_i^A + x_i^B \right)
\]

s.t.

\[
x_i^A + x_i^B \leq 1, \forall i \in N \tag{1}
\]

\[
\sum_{i \in N} a_i x_i^A \leq a \tag{2}
\]

\[
\sum_{i \in N} a_i x_i^B \leq c - b \tag{3}
\]

\( x_i^A, x_i^B \in \{0, 1\}, \forall i \in N. \)

The objective function of (MDSS1) clearly maximizes the use of the non-defective intervals. Constraints (1) guarantee the each item is assigned completely to at most one of the non-defective intervals. Constraints (2) require all the items assigned to the interval \([0, a]\) can entirely fit in it. Similarly, constraints (3) guarantee that all the items chosen to be in the interval \([b, c]\) fit in it entirely. It is clear that in the optimal solution when \( x_i^A + x_i^B = 0 \), item \( i \) intersects the interval \((a, b)\).

**Proposition 2** : The LP relaxation of MDSS1 gives \( V_{LP} = a + (c - b) \).

**Proof.** : Since the objective function is that sum of the left hand side of (2) and (3), it is clear that \( V_{LP} \leq a + c - b \). We will now show that there
is a feasible solution of value \( a + c - b \). Let \( k \) be such that \( \sum_{i=1}^{k} a_i < a \), and \( \sum_{i=1}^{k+1} a_i > a \). Similarly, let \( s \) be such that \( \sum_{i=1}^{s} a_i > c - b \) and \( \sum_{i=s}^{n} a_i \leq c - b \).

Let \( f = (a - \sum_{i=1}^{k} a_i)/a_{k+1} \) and \( g = (c - b - \sum_{i=s}^{n} a_i)/a_{s-1} \). Note that by choice of \( k \) and \( s \), \( 0 \leq f, g \leq 1 \). Consider the following solution to the LP relaxation:

\[
\begin{align*}
x^A &= (1, 1, \ldots, 1, f, 0, 0, \ldots, 0), \\
x^B &= (0, 0, \ldots, 0, g, 1, 1, \ldots, 1)
\end{align*}
\]

where \( f \) is in position \( k + 1 \) and \( g \) is in position \( s - 1 \). By construction it is clear that both the knapsack constraints hold at equality. It remains to show that \( x^A_i + x^B_i \leq 1 \), \( \forall i \in N \). Since \( b - a > 0 \) and by choice of \( k \) and \( s \) there is at least one item between the \( k \)th and \( s \)th items thus, \( k + 1 \leq s - 1 \). If the inequality is strict, then we have that \( x^A_i x^B_i = 0 \) for all \( i \), thus the inequality is satisfied as \( 0 \leq x^A_i, x^B_i \leq 1 \), for all \( i \). In the case when \( k + 1 = s - 1 \), we have to show that \( f + g \leq 1 \). In this case we have that \( a_{k+1} = a_{s-1} \geq b - a \). But \( f + g = (a - \sum_{i=1}^{k} a_i)/a_{k+1} + (c - b - \sum_{i=s}^{n} a_i)/a_{s-1} = ((a - \sum_{i=1}^{k} a_i) + (-b + \sum_{i=s-1}^{n} a_i))/a_{k+1} = (a - b + a_{k+1})/a_{k+1} \in [0, 1] \).}

Note that the Lagrangean relaxation problem obtained by relaxing the two knapsack constraints will provide the same bound as \( V_{LP} \) due to the integrality property of the remaining constraints.

By itself Proposition 2 is not that significant. However, one can discern from the structure of the LP solution that when attempting to solve the problem using standard branch-and-bound techniques, the solutions for the LP relaxations will set both the knapsack constraints at equality for all of the nodes of the tree except possibly when all the variables are fixed which happens only at the bottom of the tree. Hence the branch-and-bound will provide no bounds and procedure will simply perform implicit enumeration, resulting in long computational time.

Note that MDSS1 is a multiple knapsack problem where the objective coefficients and the constraints coefficients are the same. In turn, the ratio of profit to consumption is 1 for all the variables, thus, computational routines that are available for the multiple knapsack problems (see Martello and Toth (1990)) will generally require extremely long computational times as all the items will be treated in a symmetric manner. Therefore, although MDSS1 can be theoretically modeled as a well-known combinatorial optimization problem, the solution procedure for this special case are inadequate and necessitate a further exploitation of the structure of the model. Using
the analogy between the knapsack problem and the subset sum problem, an appropriate name for this problem would be the **multiple subset sum problem**.

### 3 Solution Techniques

It was shown in Section 2 that the LP relaxation of MDSS1 does not provide a good bound and the standard branch-and-bound will result in extreme long computational time. In order to solve integer programming effectively one needs to have good mechanism for obtaining good bounds and good feasible solutions. This section describes how to obtain stronger bounds for the problem and provides two effective heuristics for finding feasible solutions.

#### 3.1 Obtaining Bounds

One possible method for obtaining good bounds is by using surrogate relaxation. This theory been investigated in Karwan and Radin (1979) (see also Parker and Radin (1988)) and it was shown these relaxations provide better bounds than the LP relaxation, and also superior to those obtained by Lagrangean relaxation.

The relaxation of MDSS1 is obtained by using a surrogate of the two last constraints, that is, for $\theta \in [0, 1]$ we have:

$$
V(\theta) = \max \sum_{i \in N} a_i (x_i^A + x_i^B) \\
\text{s.t.} \\
x_i^A + x_i^B \leq 1, \forall i \in N \\
\sum_{i \in N} \left[ \theta a_i x_i^A + (1 - \theta) a_i x_i^B \right] \leq \theta a + (1 - \theta) (c - b) \\
x_i^A, x_i^B \in \{0, 1\}, \forall i \in N,
$$

which is a multiple choice knapsack problem. A strong bound could then be obtained by using the surrogate dual $\min \{ V(\theta) : \theta \in [0, 1] \}$ which is simple to solve because the problem has only one multiplier.

Fortunately, no numerical procedure need be used here as an optimal multiplier for this relaxation can be computed analytically. This is stated in the following proposition:
Proposition 3: $\theta = \frac{1}{2}$ is an optimal solution to $\min \{ V(\theta) : \theta \in [0,1] \}$.

Proof: Note that since the $a_i$'s sum to $c$, $V(\theta) \leq c$ for all $\theta$. First consider $0 \leq \theta < \frac{1}{2}$. When $\theta = 0$, $V(0) = c$ as all the $x_i^A$ variables can be assigned to 1. When $0 < \theta < \frac{1}{2}$, one would never assign $x_i^B = 1$ because $x_i^A = 1$ provides the same profit with less consumption of the resource. Thus the problem reduces to:

$$V(\theta) = \max \sum_{i \in N} a_i x_i^A$$

s.t.

$$\sum_{i \in N} a_i x_i^A \leq a + \frac{(1 - \theta)}{\theta} (c - b)$$

$$x_i^A \in \{0, 1\}, \forall i \in N,$$

and clearly $\inf_{0 < \theta < \frac{1}{2}} V(\theta) = V(\frac{1}{2})$ because $\theta = \frac{1}{2}$ gives the minimum right hand side of the constraint for $\theta \in (0, \frac{1}{2}]$. But we also have that $V(0) \geq V(\frac{1}{2})$. The proof for $\theta > \frac{1}{2}$ is mutatis mutandis. □

The special case when $\theta = \frac{1}{2}$ is relatively simple to compute because it can be modeled as a subset sum problem (CP) in the variables $z_i = x_i^A + x_i^B$:

$$V(\frac{1}{2}) = \max \sum_{i \in N} a_i z_i$$

s.t.

$$\sum_{i \in N} a_i z_i \leq a + (c - b)$$

$$z_i \in \{0, 1\}, \forall i \in N.$$  

It is clear that CP will provide upper bounds to MPSS.L that are at most $a + (c - b)$.

The problem CP has the physical interpretation that a cylinder was created by joining the endpoints and the items are placed optimally around the cylinder. It is clear arranging the items around a cylinder rather than a flat surface of the same length is easier and hence the "cylinder" solution provides upper bounds for the original problem.
3.2 Heuristics

In order solve problems to optimality and reduce the search using branch-and-bound good heuristics are paramount. We present two heuristics that utilize the "cylinder" solution and attempt to unwrap the solution so that it will fit on the flat surface. The first type fits the items to one interval at a time and the second is a greedy type that consider both intervals simultaneously.

3.2.1 Heuristic 1: Nested Subset Sum

Consider the optimal solution for the "cylinder" problem. The items not chosen by this solution are deleted from consideration. The heuristic then attempts to optimally assign a subset of the chosen items to the interval $[0,a]$. The heuristic then attempts to assign the remaining item ("cylinder" items not assigned to the interval $[0,a]$) to the interval $[b,c]$. This procedure is called NSSA.

Let $z^*$ denote the optimal solution to the "cylinder" problem. Let $N^* = \{ i : z_i^* = 1 \}$. First assign these variables optimally to $[0,a]$ by solving a subset problem over the interval $[0,a]$ restricted to the chosen items obtained by the optimal solution to CP:

$$\max \sum_{i \in N^*} a_i w_i$$

$$\text{s.t. } \sum_{i \in N^*} a_i w_i \leq a$$

$$w_i \in \{0,1\} \text{ for } i \in N^*.$$

Let $w_i^*$ be an optimal solution for the restricted subset sum problem over $[0,a]$ and let $N_0 = \{ i \in N^* : w_i^* = 0 \}$. Now one can solve another restricted subset sum problem over the interval $[b,c]$ by solving:

$$\max \sum_{i \in N_0} a_i t_i$$

$$\text{s.t. } \sum_{i \in N_0} a_i t_i \leq c - b$$

$$t_i \in \{0,1\} \text{ for } i \in N_0.$$

Let $t_i^*$ denote an optimal solution for restricted subset sum over the interval $[b,c]$. 


Let \( y_i^A = 1 \) if \( w_i^* = 1 \) and \( y_i^A = 0 \) otherwise, and similarly, \( y_i^B = 1 \) if \( t_i^* = 1 \) and \( y_i^B = 0 \) otherwise. The vector \( y \) is a heuristic solution for the original problem.

Another but similar heuristic (NSSB) exchanges the order of the intervals, that is, first assigns the variables to the interval \([b, c]\) and then to \([0, a]\).

### 3.2.2 Heuristic 2: Greedy Heuristic

This heuristic also starts with the "cylinder" solution but assigns the items to both intervals simultaneously. The heuristic is a greedy type where the items chosen by the "cylinder" problem are sorted descendingly by size. The first unassigned item from the list that can fit in one of the two intervals (any item that does not fit is discarded) is assigned to the larger of the two intervals \([0, a]\) or \([b, c]\). The length of the chosen interval is adjusted by the size of the chosen item and the process is then repeated.

### 3.3 Branch-and-Bound

The branch-and-bound method used to solve the problem to optimality exploits the special structure of the problem. It can be easily noticed that a single item may be placed either in interval \([0, a]\) (left), in the interval \([b, c]\) (right) or have a non-empty intersection with \((a, b)\) (center). This scheme corresponds to \((z_i^A = 1, z_i^B = 0), (z_i^A = 0, z_i^B = 1), (z_i^A = 0, z_i^B = 0)\), respectively, which is a complete partition of the solution space as we have the constraint \(z_i^A + z_i^B \leq 1\) for all \(i \in N\).

Thus, each node of the tree has three branches. At each node of the tree the CP is solved in order to obtain an upper bound for that node. The three heuristics described in Section 3.2 are executed in order to obtain feasible solutions. The incumbent solution is updated as in any standard branch-and-bound approach. All the standard fathoming criteria can also be applying, that is, one is able to fathom by value and by infeasibility (all solutions in this branch-and-bound are integral). The items are first sorted by size descendingly which determines the order of the branching variables. The rationale for this choice is that large items are more difficult to fit and they should be processed first. The strategy employed for this branch-and-bound is depth-first search. The computational results are given in Section 4.

### 4 Implementation and Computational Results

As stated in Section 2, solving MDSS1 directly would result in extremely long computational times. The problem was solved by branch-and-bound where
the surrogate relaxation $V(\frac{1}{2})$ provides an upper bound and heuristics provide feasible solutions. Solving $V(\frac{1}{2})$ and two of the heuristics require solutions for subset sum problems. These were solved using the MTSL routine by Martello and Toth (1990). Note the for the bound the optimal value is needed, whereas when applying the heuristics, an optimal solution is not necessary.

At each node of the branch-and-bound tree, after the necessary adjustments to the subproblem resulting from fixing variables, (changes in RHS coefficients), the problem $V(\frac{1}{2})$ and the heuristics NSSA, NSSB and the greedy heuristics are solved. The MTSL routine employs dynamic programming (DP). In all the nodes of the branch-and-bound tree $V(\frac{1}{2})$ is solved to optimality by the MTSL routine whereas the number of DP states in the MTSL routine is limited to 100,000 when it was used as a subroutine in the heuristics. The rationale for this is that it is necessary to obtain a true upper bound at each node of the branch-and-bound tree but for in the heuristic phase we are only interested in good feasible solutions. The branch-and-bound tree was explored using a depth-first search strategy. Computational experiments revealed that problems with relatively small value of $n$ (of about 30) were the most difficult to solve. Therefore, the allowable number of nodes in the branch-and-bound tree was chosen to be inversely proportional to $n$. We allowed for 500,000, 400,000, 300,000, 200,000, 100,000 nodes for $n = 10, 20, 30, 40, 50$ respectively and 80,000, 60,000, 40,000, 20,000, 10,000 nodes for $n = 60, 70, 80, 90, 100$ respectively.

The test problems were randomly generated but the distributions were chosen so that some particularly difficult instances would be included in the sample. Let $U[s, t]$ denote the uniform discrete distribution over the interval $[s, t]$. The algorithm was tested on three sets of problems. The method of choosing value of $a_i$'s and the length of the defective intervals is summarized in Table 1.

<table>
<thead>
<tr>
<th>Set1</th>
<th>$a_i$</th>
<th>$b - a$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Probability</td>
<td>$\frac{1}{8}$</td>
<td>$\frac{5}{8}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Set2</th>
<th>$b - a$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Probability</td>
<td>$\frac{1}{8}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Set3</th>
<th>$b - a$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distribution</td>
<td>$5 \times U[3, 5]$</td>
</tr>
<tr>
<td>Probability</td>
<td>$\frac{1}{8}$</td>
</tr>
</tbody>
</table>
The value of $c$ was chosen as the sum of $a_i$'s, as these instances have less flexibility and hence are more difficult to solve. One of the endpoints of the interval $(a, b)$ was chosen from a uniform distribution in $[1, c]$. The chosen value was assigned the role of $a$ unless that when the sum of it and the value chosen for the length of the interval exceeded $c$, it was then assigned the role of $b$.

For Set3, the length of the interval was chosen from $U[1, [q]]$, where $q = [C/4]/[n/5]$. If the result was a multiple of 5 a random amount with distribution $U[1, 4]$ was added to the result. The last data set guaranteed that all the coefficients are multiples of 5 and the length of the interval is not.

For both Set1 and Set2, the length of the interval was chosen so that the probability of fitting some of the sizes exactly in the interval $[a, b]$ is small, thus ensuring to generate difficult instances of the problem. Ten problems for each value of $n = 10, 20, \ldots, 100$ were generated. The results are reported in Table 2.

<table>
<thead>
<tr>
<th></th>
<th>Set 1</th>
<th>Set 2</th>
<th>Set 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solved to Optimality</td>
<td>93</td>
<td>94</td>
<td>85</td>
</tr>
<tr>
<td>Solved at the root</td>
<td>43</td>
<td>38</td>
<td>52</td>
</tr>
<tr>
<td>Mean Gap (Unsolved)</td>
<td>7.38%</td>
<td>5.88%</td>
<td>16.17%</td>
</tr>
<tr>
<td>Total B&amp;B Nodes(*)</td>
<td>3.15</td>
<td>1.83</td>
<td>3.93</td>
</tr>
<tr>
<td>Total Time (**)</td>
<td>1562</td>
<td>1249</td>
<td>2094</td>
</tr>
</tbody>
</table>

(*) In millions
(**) In seconds on a 486 DX2 personal computer

Our computational results demonstrate that most of the problems were solved optimality in a relatively short time. This paper is another example of important interplay between modeling and solution methodology. Elegant models (such as MDSS1) may be difficult to solve by standard commercial codes and further exploitation of their structure is needed for an effective solution procedure.

5 References.


