Exact algorithm for minimising the number of setups in the one-dimensional cutting stock problem

François Vanderbeck
Mathématiques Appliquées Bordeaux (MAB), Université Bordeaux 1
351 Cours de la Libération, F-33405 Talence Cedex, France.
Email: fv@math.u-bordeaux.fr
Url: http://www.math.u-bordeaux.fr/~fv
Fax: +33 (05) 57 96 21 23

March 1998 [1] (Revised in February 1999)

Subject Category
Integer programming algorithms: Decomposition, Branch-and-bound, Branch-and-price, Column generation, Cutting plane, Heuristic, Relaxation.
Production/scheduling: Cutting Stock/trim.

[1]Published in the series of *Research Papers in Management Studies*, University of Cambridge, no 10, 1998.

Abstract

The Cutting Stock Problem is that of finding a cutting of stock material to meet demands for small pieces of prescribed dimensions while minimising the amount of waste. As changing over from one cutting pattern to another involves significant setups, an auxiliary problem is to minimise the number of different patterns that are used. The pattern minimisation problem is significantly more complex but it is of great practical importance. In this paper, we propose an integer programming formulation for the problem that involves an exponential number of binary variables and associated columns, each of which corresponds to selecting a fixed number of copies of a specific cutting pattern. The integer program is solved using a column generation approach where the subproblem is a non-linear integer program that can be decomposed into multiple bounded integer knapsack problems. At each node of the branch-and-bound tree, the linear programming relaxation of our formulation is made tighter by adding super-additive inequalities. Branching rules are presented that yield a balanced tree. Incumbent solutions are obtained using a rounding heuristic. The resulting branch-and-price-and-cut procedure is used to produce optimal or approximately optimal solutions for a set of real-life problems.

The Cutting Stock Problem (CSP) is that of finding a feasible cutting of stock material that meets demand for small pieces of prescribed dimensions while minimising the amount of waste. In the standard one-dimensional version of the problem, one has an unlimited supply of identical stock sheets with width $W \geq 0$ and length $L \geq 0$, and a set of order items $i \in \{1, \ldots, n\}$ whose width and demand are given respectively by w_i and $d_i \in I\!N$, where $0 < w_i \leq W$ and the units used to express the integer demands d_i are number of stock sheet lengths. Then, the problem is to specify how stock sheets can be cut to produce the demanded pieces while minimising waste. As demands must be met exactly, the objective is equivalent to minimising the number of stock sheets that are used.

The standard approach to the cutting stock problem is to formulate it in terms of variables associated to the feasible ways of cutting a stock sheet (Gilmore and Gomory, 1961). Let $Q' = \{q = (q_1, \ldots, q_n) \in I\!N^n : \sum_{i=1}^{n} w_i\, q_i \leq W\}$ be the set of feasible cutting patterns, where q_i denotes the number of order pieces cut for item i. Let μ_q be the number of times cutting pattern q is used. Then, the problem takes the form

$$
[CSP] \qquad \begin{array}{rcl} Z = \min & & \displaystyle\sum_{q \in Q'} \mu_q \\ & \text{s.t.} & \\ \displaystyle\sum_{q \in Q'} q_i \, \mu_q & = & d_i \qquad\qquad i = 1, \ldots, n \\ \mu_q & \in & I\!N \qquad\qquad q \in Q'. \end{array} \tag{1}
$$

The Gilmore Gomory formulation does in fact result from the application of the Dantzig-Wolfe decomposition principle to a compact formulation of the cutting stock problem. It gives rise to a very tight LP relaxation: the round-up of the LP solution typically gives the optimal value of the integer solution. This is known as the round-up property (Marcotte, 1985). However, its large number of columns and associated variables must be dealt with using a column generation procedure. One can find exact solutions to the cutting stock problem for instances of practical size – involving say 5 to 30 order items – in just a few seconds of computational time using a branch-and-price algorithm that combines the branch-and-bound method with the use of a column gneration procedure at each node of the branch-and-bound tree (Vanderbeck, 1996).

Given the minimum number of stock sheets required to meet the orders, an auxiliary problem is to minimise the number of different cutting patterns that are used. Indeed, if practitioners agree that their primary objective is to minimise waste, they also recognise that changing over from one cutting pattern to another involves significant setup times (for adjusting of knife positions) and costs (such as those associated with the waste incurred in trial runs). Our purpose therefore is to tackle the problem of minimising the number of different cutting patterns that are used in the solution. This pattern minimisation problem is significantly more complex, as we shall see, but it is of great practical importance.

The structure of this problem is one that is shared by other applications. Essentially the problem consists in selecting feasible patterns (or scenarios) such that together they satisfy a set of global constraints (or joint constraints). The costs are made of a variable cost per copy of a pattern that is used, plus a fixed cost if a pattern is used at all. Problems where there is a fixed cost associated with the use (or launch) of a pattern are common. For instance, consider the making of a *plate* in the printing industry or a *patron* in the textile industry. Teghem et al. (1995) consider the problem of printing book covers at minimal cost: any four covers can be printed from a single plate (i.e. patterns are sets of 4 ordered covers); there is a fixed cost for producing a plate and a relatively small cost per sheet of paper that is printed. Combinatorial optimisation problems involving set-up costs (fixed costs) are notoriously difficult. Here, moreover, there are typically a huge number of possible patterns (scenarios) to choose from and associated set-up variables. A standard approach is to generate patterns as needed in the course of the optimisation, a technique known as column generation. The issue therefore is how to model and tackle fixed setup costs in the context of a dynamic generation of patterns.

In this paper, we present a branch-and-price-and-cut algorithm for the pattern minimisation problem. We start with a compact quadratic integer programming formulation of the problem. We show how a decomposition approach leads to a linear integer programming reformulation with a relatively strong linear programming (LP) relaxation. This formulation involves a huge number of binary variables, each of which is associated with the decision of selecting a fixed number of copies of a specific cutting pattern. The model is tackled using a column generation algorithm where the subproblem is a bounded knapsack problem involving quadratic terms. It can be linearised by implicitly enumerating all possible multiplicities of a cutting pattern. The lower bound obtained from the LP relaxation of our model is further improved by adding cutting planes (based of super-additive inequalities) to the formulation. The resulting bounds are used in a branch-and-bound procedure to solve our problem. Branching rules are presented that lead to a balanced tree. The approach that is used to solve the column generation subproblem after branching constraints and cuts have been added to the formulation is discussed. Some implementation details are given and computational results are presented for a set of real-life test problems.

1 The Problem

In the Pattern Minimisation Problem (PMP), an upper bound K on the number of stock sheets that can be used is given. The problem is to select cutting patterns for K or less sheets so as to satisfy demands for order items while minimising the number of different cutting patterns that are used. Typically the bound K is set to the minimum number of stock sheets required to satisfy demands d_i for $i = 1, \ldots, n$. It is obtained by solving the standard cutting stock problem. However, it can be set to higher values so as to examine the tradeoff between waste minimisation and setup minimisation. In the problem formulation below, we use index $k = 1, \ldots, K$ to identify cutting patterns, z_k is the number of times pattern k is used, and y_k is a binary variable that takes value 1 if cutting pattern k is used at all and zero otherwise. Variables $x_{i\,k}$ represent the number of strips of item i cut into the stock sheet when cutting pattern k is used, and hence completely define cutting pattern k. With these definitions, a compact formulation for the problem is given by:

$$
[P] \qquad
\begin{array}{rcllr}
Z & = \min & \displaystyle\sum_{k=1}^{K} y_k & & \\
 & \text{s.t.} & & & \\
\displaystyle\sum_{k=1}^{K} z_k\, x_{i\,k} & = & d_i & i = 1, \ldots, n & (2) \\
\displaystyle\sum_{k=1}^{K} z_k & \leq & K & & (3) \\
z_k & \leq & K\, y_k & k = 1, \ldots, K & (4) \\
\displaystyle\sum_{i=1}^{n} w_i\, x_{i\,k} & \leq & W\, y_k & k = 1, \ldots, K & (5) \\
x_{i\,k} & \in & \mathbb{N} & i = 1, \ldots, n \quad k = 1, \ldots, K & (6) \\
y_k & \in & \{0, 1\} & k = 1, \ldots, K & (7) \\
z_k & \in & \mathbb{N} & k = 1, \ldots, K. & (8)
\end{array}
$$

The objective is to minimise the number of different cutting patterns that are used. Constraints (2) ensure that the demands are met exactly. They are non-linear. Constraint (3) enforces the upper bound on the number of stock sheets. Constraints (4) ensure the proper definition of variables y_k. The feasibility of cutting patterns is guaranteed by constraints (5).

The problem presented by Teghem et al. (1995) is closely related to PMP. There, constraints (5) are replaced by $\sum_{i=1}^{n} x_{i\,k} = 4\, y_k$, as any 4 covers can define a plate; constraints (2) are replaced by demand covering constraints; constraint

(3) is absent since the objective is to minimise $\sum_{k=1}^{K} y_k + \kappa \sum_{k=1}^{K} z_k$, where κ is the ratio of variable cost over the fixed cost. Teghem et al. linearised this formulation and noted that the resulting linear mixed integer program could not be solved directly using commercial software. They resorted to using a simulated annealing approach to the problem.

Observe that when demands are all 1's, the pattern minimisation problem (PMP) reduces to the standard cutting stock problem (as no cutting pattern shall be selected more than once, $z_k = y_k$ for all k) which itself reduces to the bin packing problem. The latter problem is known to be strongly NP-complete (Garey and Johnson, 1979). So PMP is clearly a hard problem. But it is much harder than the standard cutting stock problem (CSP). McDiarmid (1996) considered the special case where any two items fit on a sheet ($w_i + w_j \leq W$, $\forall i, j$) but no three do ($w_i + w_j + w_k > W$, $\forall i, j, k$). For this special case, he showed that PMP is strongly NP-hard even though CSP is trivial to solve (as exactly $\lceil \frac{\sum_i d_i}{2} \rceil$ stock sheets are required).

An initial lower bound, $\underline{Z}$ on the number of different patterns Z that might be required in an optimum solution can be obtained by solving the associated bin packing problem where all demands have been set equal to 1. On the other hand, a solution of CSP with demands d_i's is an initial incumbent solution to PMP, giving an upper bound $\overline{Z}$. The gap between these bounds is typically quite large: in our computational results, $\underline{Z} = 6.6\%\ K$ while $\overline{Z} = 34.8\%\ K$ on average, where K is the minimum number of stock sheets required. This is primarily because algorithms for the standard cutting stock problem make no attempt at reducing the number of different patterns. Commercial codes however include heuristics that can lead to significant reduction in the number of setups. In fact, the problem instances that we solved involve significant scope for setup reduction: on average our best solution uses 12.6% K different patterns which corresponds to a 63.6% reduction in the number of setups compared to the initial solution of the standard cutting stock problem. Hence, intuitively, it is not difficult to derive a heuristic that improves $\overline{Z}$ (i.e. we would expect that any heuristic could easily achieve 10% or 20% reduction in the number of setups), but the difficulty is to obtain close to optimal solutions and lower bounds to prove this.

2 Decomposition and Reformulation

The above formulation of PMP is a non-linear integer program. Linearising it would lead to a linear integer program with a weak linear programming relaxation. Moreover, the formulation exhibits some symmetry (the indices k are interchangeable) which is bound to lead to difficulties in a branch-and-bound procedure. Hence, we reformulate the problem using the Dantzig-Wolfe decomposition principle adapted to integer programming (Vanderbeck, 1995). Indeed, by dualizing constraints (2-3) in a Lagrangian fashion, the problem decomposes into K identical subproblems that consist in selecting a feasible cutting pattern and fixing the number of times it is used in the solution. The master formulation below arises from reformulating PMP in terms of the solutions to these subproblems.

Let Q be the set of feasible solutions to constraints (4-8) for a fixed k and a relaxation of (2), i.e.

$$
\begin{aligned}
Q = \{q = (q_0, q_1, \ldots, q_n) \in I\!N^{n+1} \quad : \quad & \sum_{i=1}^{n} w_i\, q_i \leq W && (9)\\
& q_0\, q_i \leq d_i \quad \forall i\} \;. && (10)
\end{aligned}
$$

With each point q in Q, we associate a variable λ_q that takes value 1 if the cutting pattern $(q_1, \ldots, q_n)$ of a stock sheet is used q_0 times and zero otherwise. Then, problem PMP can be reformulated as

$$
[M] \qquad
\begin{aligned}
Z^M \;=\; \min \quad & \sum_{q \in Q} \lambda_q && && (11)\\
\text{s.t.} \quad & && && \\
\sum_{q \in Q} q_0\, q_i\, \lambda_q \;&=\; d_i && i = 1, \ldots, n && (12)\\
\sum_{q \in Q} q_0\, \lambda_q \;&\leq\; K && && (13)\\
\lambda_q \;&\in\; \{0, 1\} && q \in Q. && (14)
\end{aligned}
$$

Setting $\lambda_q = 1$ in formulation $[M]$ is equivalent to setting $(x_{1\,k}, \ldots, x_{n\,k}, y_k, z_k) = (q_1, \ldots, q_n, 1, q_0)$ for some k in formulation $[P]$. In the definition of Q, the constraints $q_0\, q_i \leq d_i$ $\forall i$ are implied by (2). They have been added to ensure that the columns of $[M]$ corresponds to *proper* patterns, giving rise to a tighter formulation.

Formulation $[M]$ is an integer linear program with a large number of variables. The non-linearities of $[P]$ are now implicit in the column definitions. Given the enormous number of columns and associated variables in $[M]$, we solve it using an integer programming column generation procedure, also known as a branch-and-price algorithm (Barnhart et al., 1994, Vanderbeck and Wolsey, 1996). In brief, the method consists in embedding a column generation procedure in a branch-and-bound algorithm. At each node of the branch-and-bound tree, a lower bound is obtained by solving the linear programming (LP) relaxation of the current master problem. At the root node, before any branching constraints have been added, the master LP relaxation is given by (11-13) together with the constraints $\lambda_q \geq 0$ for all $q \in Q$.

The column generation procedure that is used to solve this master LP works as follows. A restricted formulation containing only a subset of columns and associated variables is solved optimally. Its dual solution is then used to price out other columns. The most negative reduced cost column is obtained by solving a column generation subproblem,

$$\min\{1 - q_0\,(\sum_{i=1}^{n} \pi_i\, q_i - \sigma) \ : \ q \in Q\} \tag{15}$$

where $(\pi, \sigma) \in I\!R^n \times I\!R_+$ are the dual prices associated respectively with the demand covering constraints (12) and the maximum number of stock sheet constraint (13) of $[M]$. Thus, the subproblem is a non-linear integer program. If its solution defines a column with negative reduced cost, this column is added to the master. Else, the procedure terminates. Early termination of the column generation procedure is implemented using criteria presented in Vanderbeck (1995). The column generation procedure is initialized using an artificial variable and associated column. After adding any cuts or branching constraints to the master, the column reduced costs and, hence, the column generation subproblem take a different form.

Observe that the master formulation $[M]$ is not what would have been obtained by adding fixed setup costs to the Gilmore-Gomory formulation (1) of the standard cutting stock problem. This “natural” extension of the CSP formulation would give rise to an alternative master formulation $[M']$:

$$[M'] \qquad \begin{array}{rcll} Z & = \min & \sum_{q \in Q'} \lambda_q & \\ & \text{s.t.} & & \\ \sum_{q \in Q'} q_i \, \mu_q & = & d_i & i = 1, \ldots, n \\ \sum_{q \in Q'} \mu_q & \leq & K & \\ \mu_q & \leq & K \, \lambda_q & q \in Q'. \\ \lambda_q & \in & \{0, 1\} & q \in Q'. \\ \mu_q & \in & I\!N & q \in Q'. \end{array} \qquad (16)$$

where $Q' = \{q = (q_1, \ldots, q_n) \in I\!N^n : \sum_{i=1}^n w_i \, q_i \leq W\}$ and for which the column generation subproblem is a standard integer knapsack problem.

Formulation $[M']$ is much weaker than formulation $[M]$. Indeed, $[M']$ is the master formulation that would results from the dualization of constraints (2-4) of the compact formulation $[P]$ (if constraints (5) are replaced with $\sum_{i=1}^n w_i x_{ik} \leq W$ in $[P]$). Thus, the Lagrangian theory tells us that the LP relaxation of $[M]$, that results from the dualisation of fewer constraints of $[P]$, must be stronger than that of $[M']$. In fact, the LP relaxation of $[M']$ gives the trivial lower bound $\underline{Z} = 1$ (set μ_q equal to the CSP solution and $\lambda_q = \frac{\mu_q}{K}$ for all $q \in Q'$). This remark emphasizes the importance of capturing the fixed cost in the column generation subproblem as we did in formulation $[M]$. However, our approach results in a much larger number of columns (for each feasible cutting pattern, there is a column associated with each possible multiplicity). It also means that we shall have to deal with a non-linear subproblem.

Our second remark is that the LP relaxation of $[M]$ will not provide the sort of tight bounds that the column generation approach provides for the standard cutting stock problem. Here, we do not have the round-up property. In fact, it is not difficult to see that the LP can "cheat" by using columns with a large multiplicity q_0 at a fractional level. However, we expect that the bound provided by the LP relaxation of formulation $[M]$ will be typically better than that of the relaxation of $[P]$ obtained by relaxing integrality. The improvement comes from the convexification of the intersection of the knapsack polytope defined by a constraint (5) with the corresponding constraint (4) for each k. Moreover, constraint (10) implies a tighter modelling of the upper bounds on pattern multiplicities: instead of using the upper bound K as in (4), formulation $[M]$ implicitly models the bound

$$q_0 \leq \min_{i \in \{1,\ldots,n\}} \{ \left\lfloor \frac{d_i}{q_i} \right\rfloor \} . \tag{17}$$

The bounds provided by the LP relaxation of $[M]$ are further improved by adding cutting planes to the master formulation as explained below.

3 The Subproblem

At the root node, before adding any cuts or branching constraints to the master, the column generation subproblem is given by (15). If we leave the constant aside, this integer program with quadratic terms takes the form:

$$v = \max \ q_0 \, (\sum_{i=1}^{n} \pi_i \, q_i - \sigma) \tag{18}$$

$$\text{s.t.} \ \ \sum_{i=1}^{n} w_i \, q_i \leq W$$

$$q_0 \, q_i \leq d_i \quad \forall i$$

$$q_0 \in I\!N, \ q_i \in I\!N \quad \forall i \, .$$

For a fixed q_0, this problem reduces to a bounded integer knapsack problem. Note that q_0 is bounded: a trivial upper bound on the maximum multiplicity of any cutting pattern is

$$q_0^{\max} = \min\{K - \underline{Z} + 1 \, , \ \max_i d_i \, \} \, .$$

So, a brute force approach to solving the subproblem would be to enumerate on $q_0 = 1, \ldots, q_0^{\max}$ and, for each value of q_0, to solve a bounded integer knapsack problem of the form

$$max\{\sum_{i=1}^{n} \pi_i \, x_i \ : \ \sum_i w_i \, x_i \leq W \, , \ x_i \leq u_i(q_0) \ \forall i, \ \text{and} \ x_i \in I\!N \ \forall i \, \} \, , \tag{19}$$

where the item upper bounds are

$$u_i(q_0) = \min\{ \left\lfloor \frac{W}{w_i} \right\rfloor , \ \left\lfloor \frac{d_i}{q_0} \right\rfloor \} \, . \tag{20}$$

This however requires solving a pseudo-polynomial number of knapsack problems.

In fact, only a subset of multiplicity values q_0 needs to be considered. Indeed, a solution $x^* \in \mathbb{N}^n$ of the knapsack problem (19) for $q_0 = 1$, will remain optimal for all q_0 values up to the multiplicity

$$m^* = \min_i \{ \left\lfloor \frac{d_i}{x_i^*} \right\rfloor \} . \tag{21}$$

Therefore, the next q_0 that needs to be considered is $q_0 = m^* + 1$ and this remark can be applied recursively. Thus, the procedure that we use to solve the column generation subproblem at the root node is:

```
Let $q_0 = 1$ and $c = 1$.
While ($q_0 \leq q_0^{\max}$) do
    Compute $u_i(q_0)$ for $i = 1, \ldots, n$ according to (20).
    Let $v^* = c/q_0^{\max} + \sigma$ be a lower bound on (18).
    Solve the bounded integer knapsack problem (19) with initial incumbent $v^*$.
    Let $v^*$ and $x^*$ be respectively its optimum value and solution.
    Compute the maximum multiplicity $m^*$ of $x^*$ according to (21).
    If $m^* \; (v^* - \sigma) > c$, let $c = m^* \; (v^* - \sigma)$ and record $q = (m^*, x^*)$.
    Let $q_0 = m^* + 1$.
end.
```

On completion of the procedure, the solution to subproblem (15) is given by q whose reduced cost is $1 - c$.

4 Cutting Planes

As we noted in Section 2, the master LP bound for the pattern minimisation problem is not as tight as it is for the standard cutting stock problem. In our computational results the gap between the root node LP bound and our best feasible solution is 33.5% on average. Hence, we consider using a cutting plane procedure to strenghten the LP formulation.

A recent trend in combinatorial optimisation has been to revisit the use of general purpose cuts such as Chvátal-Gomory (C-G) inequalities or Gomory fractional cuts but to consider only subclasses of inequalities that are strong. For instance, Caprara, Fischetti and Letchford (1997) consider maximally violated *mod-k inequalities*, a subclass of C-G inequalities, and show that they include well-known problem-specific facet-defining inequalities for the travelling salesman problem. Marchand (1998) shows how another subclass of C-G inequalities for the continuous knapsack problem dominates specific classes of inequalities for network flow models such as flow cover or cut-set inequalities. Here, we consider a specific class of super-additive inequalities. Nemhauser and Wolsey (1988) showed that every inequality constructed by the C-G rounding procedure, and hence all maximal valid inequalities for an integer polyhedron, can be obtained from a super-additive nondecreasing function.

General purpose cuts have been previously used for the cutting stock problem. However, they were not applied in the context of a dynamic generation of columns. Goulimis (1990) used Gomory fractional cuts for solving the 2-sided cutting stock problem (where production is restricted to be within an interval instead of having to meet demands exactly). Goulimis considered small instances for which all cutting patterns can be generated a priori and he solved the resulting Gilmore-Gomory formulation of the CSP using a branch-and-cut procedure. It is well-known that Gomory fractional cuts can be derived as C-G inequalities and that, inversely, rank 1 C-G cuts can be obtained as Gomory fractional inequalities (Nemhauser and Wolsey, 1988). Scheithauer and Terno (1997) suggest using C-G inequalities for the standard cutting stock problem. By adding cuts derived from a single row of the Gilmore-Gomory formulation of CSP, i.e. cuts of the form

$$\sum_{q \in Q'} \lfloor \nu \, q_i \rfloor \, \mu_q \leq \lfloor \nu \, d_i \rfloor \ ,$$

where $0 < \nu < 1$, they have been able to close the LP gap for their test instances that did not satisfy the integer rounding property.

Combining general purpose cuts and column generation raises some difficulties since the modifications to the column reduced costs that are caused by adding cuts must be properly modelled in the column generation subproblem; modifying the subproblem may destroy its special structure and make it intractable. To our knowledge, the literature does not contain any report on using general purpose cuts in a column generation context where the column generation subproblem is a non-binary integer program as is the case for the cutting stock problem. Here, we use super-additive inequalities in a combined cut and column generation scheme. In order to limit the extent of the modifications to the column generation subproblem, we restrict our attention to a specific class of super-additive inequalities derived from a single row of the master formulation $[M]$. This class of inequalities is shown to dominate all rank 1 C-G cuts that could be generated from a single row of $[M]$. Based on these cuts, we implement a cutting plane procedure that combines cut and column generation to strengthen the LP relaxation of $[M]$ at each node of the branch-and-bound tree.

From the inequalities (13) of the pattern minimisation problem, we can derive valid inequalities of the form:

$$\sum_{q \in Q} \left(\left\lceil \frac{\gamma\, q_0}{K} \right\rceil - 1 \right) \lambda_q \leq \gamma - 1 \tag{22}$$

for $\gamma \in \{2, \ldots, K\}$. Similarly, from inequality (12) we derive inequalities:

$$\sum_{q \in Q : q_i > 0} \left(\left\lceil \frac{\gamma\, q_0\, q_i}{d_i} \right\rceil - 1 \right) \lambda_q \leq \gamma - 1 \tag{23}$$

for $\gamma \in \{2, 3, \ldots, d_i\}$ and for $i = 1, \ldots, n$. The proposition below shows that these inequalities are valid and that they dominate any rank 1 C-G inequalities that could be derived by applying the rounding procedure to the associated single row of M.

Proposition 1 *Let $S = \{x \in I\!N^n : \sum_i a_i\, x_i \leq b\}$ where $a \in I\!N^n$, $b \in I\!N$ and $a_i \leq b$ for all i. Then inequalities*

$$\sum_i F^\gamma(a_i)\, x_i \leq F^\gamma(b) \ \ \text{where } F^\gamma(z) = \max\{0, \left\lceil \frac{\gamma\, z}{b} \right\rceil - 1\} \tag{24}$$

and $\gamma \in \{2, 3, \ldots, b\}$, are valid for S. Moreover, they are equivalent to or dominate any rank 1 C-G inequalities of the form

$$\sum_i \lfloor \nu\, a_i \rfloor\, x_i \leq \lfloor \nu\, b \rfloor\ , \tag{25}$$

where $\frac{1}{b} < \nu < 1$.

Proof:
It is easy to check that $F^{\gamma}(z)$ is a non-decreasing super-additive function on $I\!R^{+}$ and that $F^{\gamma}(0) = 0$ for $\gamma \in \{2, 3, \ldots, b\}$. Hence, the validity of inequalities (24) follows from Proposition 4.1, page 229, in Nemhauser and Wolsey (1988). We now show that inequalities (24) dominate inequalities (25). Observe that for all $\nu \in (\frac{1}{b}, 1)$ such that $\lfloor \nu\, b \rfloor = F^{\gamma}(b) = \gamma - 1$, $\nu\, b < \gamma$ and, hence, $\nu\, a_i < \frac{\gamma\, a_i}{b}$ for all i, which in turn implies that $\lfloor \nu\, a_i \rfloor \leq \max\{0, \left\lceil \frac{\gamma\, a_i}{b} \right\rceil - 1\}$ for all i. ■
Note that $\nu < \frac{1}{b}$ yields an inequality (25) that is trivial; while $\nu \geq 1$ yields an inequality that is dominated by the one for $\nu - \lfloor \nu \rfloor$ (since the latter plus $\lfloor \nu \rfloor$ times $\sum_i a_i\, x_i \leq b$ implies the former). Observe that the above result also holds for the standard cutting stock problem which is a special case of PMP where $q_0 = q_0^{\max} = 1$ for all columns q.

The separation algorithm for inequalities (22-23) is a simple enumeration procedure (for each γ and i, we test for the violation of the associated inequality). Cuts (22) are added in priority because they are more likely to force the LP-bound up (since the LP cheats by using high multiplicity pattern at a fractional level). Then, cuts (23) are added. The most violated cut is selected so as to avoid adding in the master cuts that are dominated for the current LP solution. The complexity of the separation procedure is $O(m\ (\sum_i d_i + K))$, where m is the number of rows in the current master and hence an upper bound on the number of non-zero variables in the master LP solution. This complexity is pseudo-polynomial, however, in practice $\sum_i d_i$ and K are reasonably small. In the computational results reported in Section 8, the use of cuts (22-23) has lead to an average 17.3% increase in the master LP bound, reducing the optimality gap at the root node from 33.5% to 13.8% on average.

5 Branching

If the master LP bound does not allow to prove optimality of the current incumbent solution (even after adding cuts), one must resort to branching. In Vanderbeck and Wolsey (1996) and Vanderbeck (1995), we present ways of implementing branching in an IP column generation procedure. Essentially, it is not appropriate to fix (or bound) fractional master variables to their integer value as it leads to difficulties in accounting for this in the column generation subproblem and to an unbalanced branch-and-bound tree. Instead, one can proceed by fixing (or bounding) the number of columns that are used from amongst those sharing a similar property that can be easily identified in the column generation subproblem.

In short, an appropriate branching scheme consists in enforcing

$$\sum_{q \in \hat{Q}} \lambda_q \in I\!N$$

for carefully selected column subsets $\hat{Q} \subseteq Q$; i.e. if the master LP solution λ is fractional, a subset $\hat{Q} \subseteq Q$ is chosen for which $\sum_{q \in \hat{Q}} \lambda_q = \alpha \notin I\!N$ and two new branch-and-bound nodes are defined by imposing respectively

$$\sum_{q \in \hat{Q}} \lambda_q \leq \lfloor \alpha \rfloor \quad \text{or} \quad \sum_{q \in \hat{Q}} \lambda_q \geq \lceil \alpha \rceil \,. \tag{26}$$

The column subset $\hat{Q} \subseteq Q$ on which to branch should be chosen so that

a. it is easy to detect whether a solution to the column generation subproblem corresponds to a column $q \in \hat{Q}$ and hence whether it must include the dual price associated with the branching constraint in its reduced cost;

b. the partition $\hat{Q}$, $Q \setminus \hat{Q}$ of the column set Q is relatively balanced, giving rise to a balanced branch-and-bound tree.

So, a good practice is to branch on the most loosely defined (larger) subsets $\hat{Q} \subseteq Q$ that enable one to prune the current fractional solution. However, as we get deeper into the branch-and-bound tree, it will become necessary to consider ever smaller (more specific) subsets $\hat{Q} \subseteq Q$. In the worst-case a subset $\hat{Q}$ will contain (isolate) a single fractional column.

In accordance with the logic outlined above, we branch on the first column subset $\hat{Q}$ in the following list that yields a fractional sum $\sum_{q \in \hat{Q}} \lambda_q$ (the notation $q_i \leftrightarrow b$ used below stands for b is the binary vector associated with the logarithmic decomposition of q_i, i.e. $q_i = \sum_{k=0}^{\lfloor \log_2 q_i^{\max} \rfloor} 2^k \, b_k$ where $q_i^{\max} = u_i(1)$ as defined in (20)):

i. $\hat{Q} = \{q \in Q : q_0 \leftrightarrow b,\ b_l = 1\}$ for $l = \lfloor \log_2 q_0^{\max} \rfloor, \ldots, 0$;

ii. $\hat{Q} = \{q \in Q : q_0 \leftrightarrow b,\ b_l = 1\ \forall l \in S\}$ for $|S| = 2, \ldots, \lfloor \log_2 q_0^{\max} \rfloor$;

iii. $\hat{Q} = \{q \in Q : q_i > 0\}$ for $i = 1, \ldots, n$;

iv. $\hat{Q} = \{q \in Q : q_i \leftrightarrow b,\ b_l = 1\}$ for $i = 1, \ldots, n$ and $l = \lfloor \log_2 q_i^{\max} \rfloor, \ldots, 0$;

v. $\hat{Q} = \{q \in Q : q_i \geq k\}$ for $i = 1, \ldots, n$ and $k = 2, \ldots, q_i^{\max}$;

vi. $\hat{Q} = \{q \in Q : q_0 = m$ and $q_i > 0\}$ for $i = 1, \ldots, n$ and $m = 2, \ldots, q_0^{\max}$;

vii. $\hat{Q} = \{q \in Q : q_0 \leftrightarrow b^0, b_l^0 = 1$ and $q_i \leftrightarrow b^i, b_k^i = 1\}$ for $l = \lfloor \log_2 q_0^{\max} \rfloor, \ldots, 0$, $i = 1, \ldots, n$ and $k = \lfloor \log_2 q_i^{\max} \rfloor, \ldots, 0$.

These branching rules respectively amount to fixing to an integer value the number of columns with a given multiplicity (i and ii), the number of columns that include item i (iii), the number of columns that include a specific number of copies of item i (iv and v), the number of columns that include item i and have multiplicity m (vi), and the number of columns that satisfy to specific restrictions on both their multiplicity and their number of duplicates of i (vii). This selection of branching rules and branching priorities is also based on the intuition built through extensive computational experiments. We only report here what worked best in practice. Branching rules 1 and 2 are not only the most global constraints, they are also those that have the biggest impact on the LP bound because of the way the LP cheats by using patterns with high multiplicity q_0 at a fractional level. The items $i = 1, \ldots, n$ are indexed in order of non-increasing weights (i.e. $w_1 \geq w_2 \geq \ldots \geq w_n$) so that we branch first on the largest items.

In theory, these rules alone might not suffice to eliminate all fractional solutions, and more specific rules might be needed. However, in our computational tests, we have not encountered any fractional solution that could not be cut off by adding one of the above disjunctive branching constraints.

6 The Modified Subproblem

Adding cutting planes (22-23) or branching constraints (26) to the master LP formulation yields modifications to the column reduced costs and hence to the subproblem: the dual price associated with the new constraint must appear in the objective of the subproblem multiplied by a coefficient equal to the column coefficient in the master constraint. The extent of these modifications depends on how easy it is to formulate the new objective coefficient in the subproblem. Inequalities (22) and branching constraints (i-ii) of Section 5 have a contribution to the column reduced cost that is a function of q_0. Branching constraints (iii-v) see their associated dual price multiplied by a function of q_i in the subproblem objective. While inequalities (23) as well as branching constraints (vi-vii) have a contribution that is a function of both q_0 and q_i. Therefore, a generic form of the modified subproblem is:

$$v = \max\ G(q_0) + \sum_{i=1}^{n} V^i(q_0,\, q_i) + \sum_{i=1}^{n} F^i(q_i) \tag{27}$$

$$\begin{aligned} \text{s.t.}\ \ & \sum_{i=1}^{n} w_i\, q_i \le W \\ & q_0\, q_i \le d_i \quad \forall i \\ & q_0,\, q_i \in I\!N \quad \forall i\,. \end{aligned}$$

In the original subproblem (18), $G(q_0) = -\sigma\, q_0$, $V^i(q_0,\, q_i) = \pi_i\, q_0\, q_i$, and $F^i(q_i) = 0$ for all i. Adding a branching constraint of the form (iii), for instance, results in defining $F^i(q_i) = \nu\, \delta(q_i)$ for the corresponding item i, where ν is the dual variable associated with the branching constraint and $\delta(z) = 1$ if $z > 0$. Similarly, if a single cut of the form (23) for a specific i and γ is added to the master and ν is the associated dual variable, function $V^i(q_0,\, q_i)$ takes the form $\pi_i\, q_0\, q_i + \nu\ \max\{0,\ \lceil \frac{\gamma\, q_0\, q_i}{d_i} \rceil - 1\}$. When several additional constraints are present in the master, each of their contributions is added to the appropriate function in the objective of the subproblem. Observe that we did not introduce any constraints that would have a contribution to the reduced cost that is a function of more than one item $i \in \{1, \ldots, n\}$ as that would create even more complex modifications to the subproblem. Also note that we could have avoided the need for terms $F^i(q_i)$ in the subproblem objective by defining branching constraints (iii-v) as $\sum_{q \in \hat{Q}} q_0\ \lambda_q \le \lfloor \beta \rfloor$ or $\ge \lceil \beta \rceil$ instead of $\sum_{q \in \hat{Q}} \lambda_q \le \lfloor \alpha \rfloor$ or $\ge \lceil \alpha \rceil$. However, the latter disjunctive constraints are typically stronger (i.e more restrictive).

If we fix the multiplicity, say $q_0 = m$, problem (27) reduces to a bounded integer knapsack problem:

$$v^m = \max \sum_{i=1}^{n} V^i(m,\, q_i) + \sum_{i=1}^{n} F^i(q_i) \tag{28}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} w_i\, q_i \leq W$$

$$q_i \leq u_i(m) \quad \forall i$$

$$q_i \in I\!N \quad \forall i\,.$$

where $u_i(m)$ is defined by (20). However, the objective function is typically non-linear. By computing the cost $c_{i\,p}$ associated with each value p that can be assumed by entry q_i, we derive a multiple choice knapsack problem:

$$\max \sum_{i=1}^{n} \sum_{p=0}^{u_i(m)} c_{i\,p}\, x_{i\,p} \tag{29}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} \sum_{p=0}^{u_i(m)} w_i\, p\, x_{i\,p} \leq W$$

$$\sum_{p=0}^{u_i(m)} x_{i\,p} = 1 \quad \forall i$$

$$x_{i\,p} \in \{0,1\} \quad \forall i\,, p\,.$$

Formulation (29) involves a pseudo-polynomial number of variables. However, for our test problems, $\sum_{i=1}^{n} u_i(1)$ remains reasonably small. Moreover, as we increase m, the number of variables reduces significantly.

The core knapsack subproblem (28) takes a simpler form if we do not use branching constraints (iii) and (vi) and we consider a weaker version of cuts (23). Branching constraints (iv) and (vii) involve a logarithmic decomposition of the q_is. By transforming problem (28) into its 0-1 form, we can formulate these contributions in linear terms. The transformation is defined by the change of variable $q_i = \sum_{k=0}^{\lfloor \log_2 q_i^{\max} \rfloor} 2^k\, b_k^i$ where $b_k^i \in \{0,1\}$ for $k = 0, \ldots, \lfloor \log_2 q_i^{\max} \rfloor$ and $i = 1, \ldots, n$, where $q_i^{\max} = u_i(1)$. In the resulting 0-1 form of the problem (which is a bounded mutiple-class binary knapsack problem: see Vanderbeck, 1998), the contribution of a branching constraint of type (iv) for a given l, for instance, would be $F^i(q_i) = \nu\, b^i_l$, where ν is the associated dual price. The contribution of cuts (23) can also be expressed linearly in the 0-1 form of the knapsack problem if we consider a weaker version. Indeed,

$$\sum_{q \in Q} \sum_{k=0}^{\lfloor \log_2 q_i^{\max} \rfloor} \left(\left\lceil \frac{\gamma\, q_0\, 2^k}{d_i} \right\rceil - 1 \right)^+ b^i_k\, \lambda_q \leq \sum_{q \in Q : q_i > 0} \left(\left\lceil \frac{\gamma\, q_0\, q_i}{d_i} \right\rceil - 1 \right) \lambda_q \,.$$

The 0-1 form of the core knapsack problem (28) involves a polynomial number of variables and is easier to solve than (29). In our computations, we experimented with both the multiple choice knapsack and the 0-1 bounded knapsack as core subproblem. We found that even when using the former, the time spent in the column generation subproblems remained below 16% of the total CPU time on average.

In solving subproblem (27), we distinguish between increasingly complex cases corresponding to various assumptions on functions G, V^i, and F^i. As we outlined in Section 3, a brute force approach to solving subproblem (27) consists in enumerating all possible values of q_0 and in solving the integer knapsack problem (28) associated with each value of q_0. This method is used as a last resort. Below, we review 3 cases where only an implicit enumeration of all values of q_0 is needed, possibly leading to fewer calls to the knapsack solver.

Case 1:
When G is a linear function of q_0, $V^i(q_0, q_i) = q_0\, U^i(q_i)$ for some function U^i for all i, and $F^i(q_i) = 0$ for all i as is the case in particular for the initial master LP formulation, the procedure presented in Section 3 applies, although (19) may have to assume the more general form (28).

Case 2:
When G is a non-linear function of q_0 but $V^i(q_0, q_i) = q_0\, U^i(q_i)$ for some function U^i for all i, and $F^i(q_i) = 0$ for all i as above, the procedure of Section 3 can be amended so as to be applicable: for a given knapsack solution x^* and associated maximum multiplicity m^*, one must enumerate on $q_0 = 1, \ldots, m^*$ to identify the optimal multiplicity q_0^* associated with solution x^*; nevertheless, the next knapsack problem to be considered is defined by setting the item upper bounds to $u_i(m^* + 1)$ according to (20). This special case arises when one uses only

cuts (22) and branching constraints (i-ii). Note that these cuts and branching constraints, that aim at fixing the pattern multiplicity to their integer values, have the most significant impact on the lower bound on the minimum number of different patterns. Hence, one could already obtain a good lower bound without considering more complex subproblems. Moreover, as we noted above, by defining branching constraints (iii-v) as $\sum_{q \in \hat{Q}} q_0 \; \lambda_q \leq \lfloor \beta \rfloor$ or $\geq \lceil \beta \rceil$, we eliminate the need for terms $F^i(q_i)$ in the objective of the subproblem. Then, one can also use branching constraints (iii-v) while satisfying the assumptions of Case 2.

Case 3:

When the only assumption is that $V^i(q_0, q_i) = q_0 \, U^i(q_i)$, as is the case so long as we do not use cuts (23) and branching constraints (vi-vii), the subproblem can still be solved through an implicit enumeration on q_0, using a branch-and-bound procedure. Each node problem is defined by restricting $q_0 \in [a, b]$: i.e. at the root $q_0 \in [0, q_0^{\max}]$, while leaf nodes correspond to a fixed value of q_0. For $q_0 \in [a, b]$, the subproblem is $SP(a, b) \equiv$

$$v(a,b) = \max\{G(q_0) + q_0 \, U(q) + \; F(q) : a \leq q_0 \leq b, \; \sum_i w_i \, q_i \leq W, \; q_0 \, q_i \leq d_i \; \forall i\} \; ,$$

where $U(q) = \sum_{i=1}^n U^i(q_i)$ and $F(q) = \sum_{i=1}^n F^i(q_i)$. To obtain an upper bound on $v(a, b)$, we relax constraints $q_0 \, q_i \leq d_i$ into $a \, q_i \leq d_i$ for all i, which leads to a decomposition:

$$v(a,b) \leq \max_{a \leq q_0 \leq b} G(q_0) \quad + \quad \max_{\substack{\sum_i w_i \, q_i \leq W \\ a \, q_i \leq d_i \; \forall i \\ a \leq q_0 \leq b}} q_0 \, U(q) + F(q)$$

The first problem can be solved by a simple enumeration. For the second problem, note that if $(q_0^*, q_1^*, \ldots, q_n^*)$ is an optimal solution, then either $U(q^*) \leq 0$ and $q_0^* = a$ or $U(q^*) > 0$ and $q_0^* = b$. Therefore, we solve the two knapsack problems

$$v^a = \max_{\substack{\sum_i w_i \, q_i \leq W \\ q_i \leq \lfloor \frac{d_i}{a} \rfloor \; \forall i}} a \, U(q) + F(q) \quad \text{and} \quad v^b = \max_{\substack{\sum_i w_i \, q_i \leq W \\ q_i \leq \lfloor \frac{d_i}{a} \rfloor \; \forall i}} b \, U(q) + F(q)$$

and we derive the upper bound

$$\text{Upper Bound:} \quad \max_{a \leq q_0 \leq b} G(q_0) \quad + \quad \max\{v^a, v^b\}$$

Let x^a and x^b denote the optimal solution of the two knapsack problems and m^a and m^b denote their maximum multiplicity. Note that x^a solves problem $SP(a,a)$. If $m^b \geq b$, then x^b solves problem $SP(b,b)$. Indeed, x^b is a feasible solution to $SP(b,b)$ and the existence of a strictly better solution $\hat{x}$ to $SP(b,b)$ would imply $b\,U(\hat{x}) + F(\hat{x}) > b\,U(x^b) + F(x^b)$, which contradicts the definition of x^b. We use x^a and x^b to derive lower bounds (to be compared to the incumbent solution to $SP(0, q_0^{\max})$) by optimising in q_0 for the fixed partial solutions x^a and x^b, i.e.

$$\text{Lower Bound: } LB^a = \quad F(x^a) \quad + \quad \max_{q \leq m^a} G(q) + q\,U(x^a)$$

and LB^b is defined in the same way. As we observed above, LB^a is also a valid upper bound on $SP(a,a)$. Hence, if the current node cannot be pruned, we will need to divide problem $SP(a+1,b)$. Moreover, if $m^b \geq b$, LB^b is a valid upper bound on $SP(b,b)$ and the remaining problem is $SP(a+1,b-1)$. If branching is needed, we partition the remaining interval of q_0 values in such a way that the current upper bound will no longer be valid in any of the two sub-nodes. For instance, in the case $v^a \leq v^b$ and $a < m^b < b$, we divide $SP(a+1,b)$ into $SP(a+1,m^b)$ and $SP(m^b+1,b)$.

7 Implementation Details

Before tackling the pattern minimisation problem, we solve the associated cutting stock problem using the algorithm presented in Vanderbeck (1996). The solution of CSP provides an initial incumbent solution for PMP and we set K to be the number of stock sheets used in the CSP solution. Then, we compute the L2 lower bound (see Martello and Toth, 1990) for the corresponding Bin Packing Problem (BBP) that is derived by setting $d_i = 1$ for all i, and we use it as an initial lower bound for PMP. At the root node, the master initially contains a single column, with entries equal to the constraint RHS for greater or equal to constraints and zero otherwise. This column corresponds to an artificial variable whose cost is set equal to that of the incumbent solution. The artificial variable enables us to combine the Phase 1 and the Phase 2 of the simplex algorithm. It will remain in the master formulation at all time. We found that initialising the master with the CSP solution resulted in longer computation times.

At each iteration of the column generation procedure, a valid lower bound on the master LP solution can be derived from the optimal reduced cost value. (The solution of the *restricted* master LP that only includes a subset of columns is an upper bound on the master LP solution.) For the cutting stock problem, the best such lower bound is due to Farley (1990) (see Vanderbeck, 1996, for a comparison with the Lagrangian bound). At a given branch-and-bound node u, Farley's bound is given by

$$\left\lceil \frac{Z_u^R}{v_u} \right\rceil$$

where Z_u^R is the current value of the restricted master LP and v_u is the value of the solution of the column generation subproblem defined by (27). Let LB denote the current best valid lower bound for PMP, and LB^u denote the current lower bound at node u (i.e the maximum of LB and all computed Farley's bounds at node u). The column generation procedure is interrupted when either no more columns with negative reduced costs can be found, or the master LP gap is closed (i.e. $Z_u^R \leq LB^u$), or the current lower bound is worse than an aspiration level that we define as $LB + 1$ (i.e. $LB^u \geq LB + 1$).

When the column generation procedure is interrupted, we check whether the artificial variable is still in the basic master LP solution. If so, we increase its cost and return to the column generation procedure. Otherwise, we search for a most violated cut (22-23). If a violated cut is found, we add it to the master and return to the column generation procedure. Otherwise, if the node cannot be temporarily pruned by bound (i.e. if $LB^u = LB$), we branch using the first disjuntive constraints defined by rules (i) to $(viii)$ of Section 5 that is violated by the current LP solution. The next node to be processed is selected according to the best-bound-first priority rule and, amongst nodes with the same lowest bound, LB, we select the deepest. This tree search strategy, together with interrupting node computation when the node bound exceeds the aspiration level $LB + 1$, ensures that our algorithm focuses on finding an integer solution of cost LB first. It will either find one or it will prove that the lower bound LB can be incremented by one unit. In the latter case, the branch-and-bound nodes u for which LB^u is equal to $LB + 1$ will be re-visited.

On completion of the column generation procedure at the root node, we apply a straightforward **rounding heuristic**. Subsequently, we apply this heuristic before examining a new node (having deleted all branching constraints) provided that 3 n new columns have been generated since the last application. The heuristic consists in recursively rounding a fractional master variable up to 1. Variables that are fixed in this way are eliminated from the optimisation problem: we amend the RHS of the master constraints to account for the part of demands that are covered by the selected columns, for their usage of stock sheets, and for their contribution to the cuts and branching constraints. Before rounding takes place, however, variables that are at value 1 are fixed to that value. As we fix more and more variables the size of the master problem reduces. After fixing a variable, we return to the column generation procedure, generating columns for the reduced-size master (they are of course valid for the full problem). This is important since it is very likely that the current set of columns does not include a feasible solution. The procedure terminates when the lower bound obtained in the course of the column generation procedure is worse than the current incumbent solution.

In the rounding procedure, the variable that has the largest value is selected for rounding. The first variable that is rounded-up can be seen as the *seed* of the heuristic solution. Variables that have been used as seed in previous application of the rounding procedure are recorded and can no longer be selected as the first variable to be rounded. When all variables currently in the solution have been a seed, the recorded list of seeds is reinitialised to the empty list. In one application of the heuristic, the rounding procedure is repeated $(1 + 3 \times \textit{optimality gap})$ times, each time with a different seed. Of course, this number of passes and how often we apply the heuristic procedure are parameters. The values that we give here illustrate what seemed to work well in our experiments. The cutting plane algorithm also involves some parameters, such as the violation threshold (set at 0.001) and whether or not we apply it within the rounding heuristic. Although the separation procedure is quite fast, the cutting plane procedure is time consuming because it requires returning to the column generation procedure every time a cut is added. Therefore, we do not use it during the rounding heuristic. The cuts that have been generated at each node are recorded as local cuts to be used in all descendent nodes. At each call of the separation routine, we erase the cuts currently in the formulation for which the slack is greater than 0.2.

8 Computational Results

Our algorithm was tested on a set of real-life instances. In Table 1, the instances whose name includes the letters kT were provided by Greycon Ltd, a software company specialised in cutting stock problems (commercial code DeckleBench^{TM}) and directed by Dr. Goulimis. They are representative of the hardest one-dimensional cutting stock problems that arise in practice. The other test problems were obtained from Vance (1996): the smaller problems are real-life instances while the larger ones have been constructed by combining several real-life problems into one. The algorithm was implemented in C. CPLEX 3.0 was used to solve the master LPs. The computations were carried out on an HP9000/712/80 workstation with 64Mb of main memory. Computations are interrupted after 2 hours of CPU time.

In Table 1, we present our numerical results. The columns of the table contain the following entries:

name is the name of the instance;

n is the number of items,

N is the number of variables in the core knapsack problem (29), i.e. $N = \sum_i u_i(1)$ where $u_i(q_0)$ is defined in (20); the ratio $\frac{N}{n}$ gives the average number of times an item can fit in the knapsack;

BBP is the initial lower bound obtained by computing the L2 bound for the associated BBP;

Mlp is the value of the master LP relaxation before applying the cutting plane procedure;

MC is the value of the master LP relaxation after adding all violated cuts (22-23);

LB is the best known lower bound that we could prove by running the branch-and-bound procedure for no more than 2 hours of CPU time;

UB is the best incumbent solution value that we encountered in the course of the algorithm (either it was obtained during the rounding heuristic, or it has arisen as an intermediate solution to a node master LP);

CSP is the number of different patterns in our solution of the standard cutting stock problem, which was denoted by $\overline{Z}$ in Section 1;

K is the minimal number of stock sheets required;

dth is the maximum depth of the branch-and-bound tree achieved during our (at most 2 hours of) computations;

nod is the number of node problems solved in the branch-and-bound tree;

mast is the number of master LPs that were solved in the course of the algorithm;

sp is the number of subproblems (27) that were solved in the course of the algorithm;

cuts is the number of cuts that were generated in the course of the algorithm;

timLb is the total CPU time in seconds up to the last improvement in the lower bound LB;

timUb is the total CPU time in seconds up to the last improvement in the incumbent solution and the upper bound UB; if the problem was solved to optimality within the allocated 2 hours of CPU time, then the largest of *timLb* and *timUb* represents the total CPU time needed to solve the problem;

tM is the percentage of the total time (i.e. 2 hours for the problems that were not solved to optimality) that was used to solve master LPs;

tSp is the percentage of the total time that was used to solve subproblems.

The last row of the Table 1 gives the average figures over the 16 problem instances.

Out of the 16 problem instances, 12 were solved to optimality. For the others, a solution was found within 1 unit of optimality. (For problem $7p18$, we found a primal solution of value 6, while running the algorithm with another set of parameters.) The hardest problems seem to be those with high N and K. The larger these inputs are, the larger the solution space (i.e. large N and K imply a large number of different cutting patterns and multiplicities). The algorithm spends most of its time (71.7% on average) solving master LPs. Although adding

name	n	N	iB	Mlp	MC	LB	UB	CSP	K	dth	nod	mast	sp	cuts	timLb	timUb	tM	tS
kT03	7	15	3	5	5	6 =	6	8	66	17	91	510	286	95	4.0	28.1	45	2
kT05	10	22	4	6	8	9 =	9	12	47	15	37	346	219	44	10.9	3.3	38	3
kT01	5	34	1	3	3	3 =	3	7	14	0	1	138	104	25	1.0	2.8	19	3
kT02	24	39	13	16	18	18 =	18	66	66	0	1	114	94	14	1.7	3.1	22	3
kT04	16	41	6	7	8	9 =	9	20	38	23	57	628	494	67	35.8	3.6	56	2
d16p6	16	41	6	7	8	9 =	9	17	38	18	39	530	420	59	29.0	5.4	56	2
7p18	7	41	2	4	5	6 <	7	8	91	48	1354	5738	2291	871	2105.7	11.5	73	
d33p20	23	81	5	7	7	8 =	8	17	29	36	655	3462	2970	105	2054.1	9.1	75	
12p19	12	84	2	3	4	5 =	5	15	23	14	47	1152	960	128	141.2	10.5	58	2
d43p21	32	101	7	8	9	10 =	10	28	40	10	33	1722	1583	96	230.4	143.5	58	3
kT06	9	110	1	2	3	4 =	4	13	51	11	51	2020	1477	446	1796.7	66.2	67	2
kT07	11	114	2	3	4	5 =	5	13	65	17	163	4118	3167	721	6818.8	208.8	78	1
14p12	14	130	2	4	5	5 =	5	24	56	0	1	690	622	63	29.8	68.3	20	6
kT09	14	145	2	4	5	5 <	6	17	110	18	140	4127	3171	693	21.2	66.1	71	1
11p4	11	148	1	3	4	4 <	5	17	101	9	19	4886	4424	379	76.2	2804.2	66	2
30p0	26	184	4	6	7	7 <	8	40	90	8	13	9435	9085	286	37.5	3562.9	73	2
16	14.8	83.3	3.8	5.5	6.4	7.1	7.3	20.1	57.8	15	170	2476	1961	255	837.12	437.32	72	1

Table 1: Computational results for a set of real-life instances of the pattern minimisation problem.

cuts to the master makes the LP harder to solve, it is worthwhile because it yields better bounds, a smaller branch-and-bound tree, and hence fewer master LPs to solve. Moreover, the cuts also contribute to the emergence of integer solutions. To reduce the number of master LPs that have to be solved, it is also essential to implement strong criteria for the early termination of the column generation procedure (such as the aspiration level criteria presented in the previous Section) and to derive an efficient branching scheme. Given that, for our test data, solving master LPs is the computational bottleneck of our algorithm, we do not hesitate using even the branching constraints that yield the most complex modifications to the subproblem and we do not use the weaker version of the cuts which would allow us to linearise the objective of the core knapsack subproblem. However, the time spent in generating columns remains reasonable (15.9 % of the total time on average).

9 Concluding Remarks

An application that implicitly requires a huge number of setup variables was considered. Such a model, involving the selection of patterns with a fixed cost attached to their use, may find applications beyond the cutting stock problem. We have seen that re-defining patterns so as to include their multiplicity in their definition leads to a strong formulation. This approach amounts to *applying Dantzig-Wolfe decomposition to a non-linear integer program* ($[P]$ in this case). The strength of the resulting master formulation is due to the fact that we effectively model setups within the pattern generation subproblem.

In an effort to limit the complexity of the separation of rank 1 C-G cuts for the knapsack polytope (defined by a single row of our master integer program), we have identified a subclass of inequalities that can be seen as super-additive inequalities and that dominates the class of rank 1 C-G cuts. Proposition 1 should be of interest beyond this particular application. Interesting issues are the questions of whether this subclass can be restricted even further and what facet-defining inequalities are encompassed by this subclass.

Using cutting planes in a context of dynamic generation of columns raises the issue of subproblem tractability. Previous work in the area involved cuts that did not imply any modifications to the subproblem. Here, the cuts we used resulted in non-linearities in the core knapsack subproblem. We refrained from using cuts based on more than one row of the master because of their effect on the subproblem. Note that some classes of cuts, where the coefficient cannot be defined in closed form as a function of the column entries, are not amenable to column generation. For instance, cover inequalities for the knapsack polytope would not be practical to use in this context unless cover membership can be expressed as a function of the column entries that can be modelled in the subproblem.

As with most approaches for difficult combinatorial problems, decomposition is a divide and conquer method. The problem is divided into a master and a subproblem. The tractability of the problem is that of the most difficult (the computational bottleneck) of these two components. Hence it is important to balance the difficulty between master and subproblem, which led to considering a complex subproblem in this case (we found it worthwhile to use cutting planes even though it made our subproblems harder to solve). The second computational tip we learned from this study is that partial enumeration is a very effective optimisation tool when the enumeration is sufficiently restricted. Indeed, we initially tackled the subproblem by linearising the products $q_0\, q_i$. We later found that solving $O(K)$ knapsack problems instead was very much faster.

Acknowledgement

We thank Constantine Goulimis for making us aware of the importance of setup minimisation in cutting stock problems and for providing data sets. We are also very grateful to Adam Letchford who reviewed the draft of this paper, corrected numerous typos, and made valuable suggestions and to the anonymous referees who made constructive comments and specific remarks. The author is supported in part by a Management Research Fellowship granted by the British Economic and Social Research Council (ESRC).

References

Barnhart C., E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh and P.H. Vance (1994). Branch-and-Price: Column Generation for Solving Huge Integer Programs. *Mathematical Programming, State of the Art*, book of the International Symposium on Mathematical Programming, 1994, edited by J.R.Birge and K.G. Murty.

Caprara, A., M. Fischetti, and A. Letchford (1997), On the separation of Maximally Violated mod-k Cuts, *working paper*, DEIS, University of Bologna, Italy.

DeckleBench^{TM} V7 for Windows, Greycon Ltd, 7 Calico House, Plantation Wharf, London SW11 3UB, info@greycon.com.

Farley A.A. (1990), A note on bounding a class of linear programming problems, including cutting stock problems, Operations Research, vol. **38**, No. 5, 922-923.

Garey M.R. and D.S. Johnson (1979). *Computers and Intractability: a guide to the theory of NP-Completeness.* W.H. Freeman and Company, San Francisco.

Gilmore P.C. and R.E. Gomory (1961), A Linear Programming Approach to the Cutting Stock Problem, *Operations Research*, **9**, 849-859.

Goulimis, C.N. (1990), "Optimal Solutions for the Cutting Stock Problem", *European Journal of Operational Research*, **44**, 197-208.

Marchand, H. (1998), Ph.D. Thesis, CORE, Faculté des Sciences Appliqées, Université Catholique de Louvain, Louvain-la-Neuve.

Marcotte O. (1985). The Cutting Stock Problem and Integer Rounding, *Mathematical Programming*, **33**, 89-92.

Martello S. and P. Toth (1990). *Knapsack Problems: Algorithms and computer Implementations*, Wiley-Interscience Series in Discrete Mathematics and Optimization.

McDiarmid C. (1996). Pattern minimisation in cutting stock problems, *working paper*, Department of Statistics, University of Oxford.

Nemhauser G.L. and L.A. Wolsey (1988). *Integer and Combinatorial Optimization.* John Wiley & Sons, Inc.

Scheithauer G. and J Terno (1997). "Improving the formulation of the cutting stock problem", Presentation given at the International Symposium on Mathematical Programming, Lausanne, August 1997.

Teghem J., M. Pirlot, and C. Antoniadis (1995). Embedding of linear programming in a simulated annealing algorithm for solving a mixed integer production planning problem. Journal of Computational and applied mathematics ,64, 91-102.

Vance P.H. (1996), Branch-and-Price Algorithms for the One-Dimensional Cutting Stock Problem. Working paper. Department of Industrial Engineering, Auburn University, Auburn, Alabama 36849-5346.

Vanderbeck F. (1995). On Dantzig-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. Forthcoming in Operations Research.

Vanderbeck, F. (1996). Computational Study of a Column Generation algorithm for Bin Packing and Cutting Stock problems, *Research Papers in Management Studies*, F. Vanderbeck, University of Cambridge, no 1996-14.

Vanderbeck F. and L. A. Wolsey (1996). An Exact Algorithm for IP Column Generation, *Operations Research Letters* Vol. 19, No. 4, pp 151-159.

Vanderbeck, F. (1998). Extending Dantzig's Bound to the Bounded Multiple-class Knapsack Problem, *Research Papers in Management Studies*, University of Cambridge, no 14.