Math. Program., Ser. A 86: 565–594 (1999) © Springer-Verlag 1999

Digital Object Identifier (DOI) 10.1007/s101079900096

François Vanderbeck

Computational study of a column generation algorithm for bin packing and cutting stock problems

Received October 18, 1996 / Revised version received May 14, 1998
Published online July 19, 1999

Abstract. This paper reports on our attempt to design an efficient exact algorithm based on column generation for the cutting stock problem. The main focus of the research is to study the extend to which standard branch-and-bound enhancement features such as variable fixing, the tightening of the formulation with cutting planes, early branching, and rounding heuristics can be usefully incorporated in a branch-and-price algorithm. We review and compare lower bounds for the cutting stock problem. We propose a pseudo-polynomial heuristic. We discuss the implementation of the important features of the integer programming column generation algorithm and, in particular, the implementation of the branching scheme. Our computational results demonstrate the efficiency of the resulting algorithm for various classes of bin packing and cutting stock problems.

Key words. integer programming – column generation – bin packing – cutting stock

Introduction

The efficiency of a branch-and-bound algorithm for an integer programming (IP) application depends on the quality of bounds used, the design of the branching scheme, the search priority, but also on enhancements such as variable fixing, the tightening of the formulation with cutting planes, rounding heuristics, and the like. In search for an efficient branch-and-price algorithm[1], one would like to incorporate the features that have proved helpful in standard branch-and-bound algorithms. However, most of these features are not straightforwardly applicable in a column generation context. The difficulty arises from the fact that there is a large number of variables to be considered and the working formulation (the restricted master) contains only a subset of the variables while the others are known implicitly through the solution of a column generation subproblem. Then, adding a branching constraint or a cutting plane in the master, for instance, results in modifying the definition of variable reduced costs and, hence, the structure of the column generation subproblem. Because of these modifications, the subproblem can become intractable.

In previous papers (Vanderbeck and Wolsey, 1996, and Vanderbeck, 1996), we have shown how branching could be efficiently implemented in a branch-and-price algorithm

F. Vanderbeck: Mathématiques Appliquées Bordeaux (MAB), Université de Bordeaux 1, 351, Cours de la Libération, F-33405 Talence Cedex, France, e-mail: fv@math.u-bordeaux.fr, Url: http://www.math.u-bordeaux.fr/∼fv

[1] A branch-and-price algorithm, also called an integer programming column generation algorithm, is a procedure that uses a column generation algorithm at each node of a branch-and-bound tree.

by introducing in the master, auxiliary variables that represent the number of selected columns with a prescribed property and by forcing these auxiliary variables to their integer value. Adding cutting planes can be viewed in the same way. However, the scheme efficiency depends on how easily one can recognise the prescribed column property that defines the auxiliary variable when solving the column generation subproblem, i.e. on the complexity of the resulting subproblem modifications. In this paper, we demonstrate how such a scheme can be efficiently implemented in practice. We also consider adding simple cuts in the master formulation. Moreover, we show that the standard techniques of variable fixing and rounding heuristics can be used in a column generation context if applied to these auxiliary variables.

The Cutting Stock Problem (CSP) has been chosen for these experimentations because it is one of the simplest models well suited for branch-and-price. Consequently, one's intuition on algorithmic design and understanding of the effects of different algorithmic features are not impaired by the complexity of the problem structure. We consider the classical one-dimensional model with no side constraints. Then, the column generation subproblem is an integer knapsack problem and the master problem is that of selecting a minimum number of cutting patterns (solutions to the knapsack problem) that together satisfy demands for cut items. The Bin Packing Problem (BBP) can be seen as a special case of CSP with unit demands. In this paper, we generalise some known results for BBP to CSP. We also show that the lower bound provided by the LP relaxation of the column generation formulation dominates the combinatorial bounds, L^2, proposed by Martello and Toth (1990) and we propose a pseudo-polynomial heuristic for the CSP.

The paper discuss in some details the elements of the branch-and-price algorithm that have a significant influence on the overall efficiency. In particular, we compare different initialisations of the column generation procedure; we consider criteria for early termination of the column generation procedure; we discuss early branching, variable fixing, and the use of cutting planes; we say how we select appropriate branching priorities, and how we implement a rounding heuristic. This study, we hope, will provide a useful reference for those wishing to develop an efficient branch-and-price algorithm that incorporates some of the above enhancement features.

Our algorithm solves cutting stock instances with 50 items, an average demand of 50, a bin capacity of 10 000, and various sorts of item size distributions within 25 seconds of CPU time on average (on an HP 712/80). The algorithm is also suitable and equally efficient for bin packing problems (large bin packing instances with integer item size tend to have several items of the same size which are better replaced by a single item with demand equal to the number of duplicates). Standard test problems with 500 items, integer item sizes from various uniform distribution, and a bin capacity less than 150 are solved within 2.5 seconds.

Several exact algorithms for bin packing and cutting stock problems have appeared in the literature. Martello and Toth (1990) summarise previous attempts at solving the bin packing problem and propose a branch-and-bound algorithm based on combinatorial(a priori) bounds and a "first-fit decreasing" branching strategy. Scholl et al. (1997)

propose further combinatorial bounds and heuristics that can allow to obtain provably optimum solutions for instances that proved difficult for Martello and Toth's algorithm. Vance et al. (1994) propose a column generation based algorithm for bin packing, where columns correspond to feasible bins. The branching strategy is based on enforcing either join or separate assignment for item pairs. Valerio de Carvalho (1996) also uses a column generation based algorithm but the master is an arc flow formulation —the underlying network is that of a dynamic programming solution of the knapsack problem— and branching is on the arc flow variables. This approach is otherwise known as the variable redefinition approach as introduced by Eppen and Martin (1987).

The cutting stock problem is notorious for the strength of its linear programming (LP) lower bound. In practice, the round up of the master LP value is often the optimal IP value (this is known as the integer round-up property, see Marcotte, 1985). Moreover, the cutting stock problem has a "flat" objective function with many solutions of the same cost. Consequently, an exact algorithm for the cutting stock problem is mostly a search for one of the many feasible integer solutions whose cost is equal to the lower bound known from the outset. For this problem, Goulimis (1990) obtains optimal solutions for small instances by generating all feasible cutting patterns in advance and using a combination of a cutting plane procedure (based on Gomory cuts) and a branch-and-bound algorithm. Goulimis also solves problems with both lower and upper bounds on the item requirements.

Scheithauer and Terno (1995) find exact solutions to cutting stock problems using a three stage procedure: they solve the continuous relaxation of the problem by column generation, they round down the continuous solution to obtain a partial integer solution, and they solve the residual problem (of finding cutting patterns to cover the remaining demands) by branch-and-bound. Their solution proves to be optimal if the total number of cutting patterns used in the partial integer solution and the residual problem is no greater than the continuous lower bound rounded up. Their algorithm for the residual problem is a branch-and-price algorithm where branching consists in fixing a specific cutting pattern to zero or one. Then, in the column generation subproblem, they generate a minimum reduced cost pattern from among those not set to zero (i.e. they compute the n^{th} best solution – see Barnhart et al. 1994).

Vance (1996) develops a branch-and-price algorithm for the cutting stock problem based on generating maximal cutting patterns only (i.e. patterns that cannot accommodate an extra item) and branching on cutting patterns that are used fractionally. Then, one can avoid regenerating patterns that have been bounded up by adjusting the item upper bounds in the knapsack subproblem. The approach of Valerio de Carvalho (1996) can also be used to solve cutting stock problems. However, the size of the underlying network used by Valerio de Carvalho and hence the computational burden increases with the bin capacity. In the branch-and-price algorithm used by Degraeve and Schrage (1998), branching is performed by bounding pattern selection variables (master variables) up or down. Then, patterns that have been bounded-up should not be considered in the column generation subproblem, hence one might have to compute the n^{th} best solution of the subproblem.

Here, we use a branch-and-price algorithm that differs from that of Vance (1996) or Degraeve and Schrage (1998) by its branching scheme. The branching scheme we use is that of Vanderbeck (1996) – There, we did some comparative testing, showing that the proposed branching scheme seems to lead to smaller branch-and-bound trees than the branching scheme used by Vance (1996) or by Valerio de Carvalho (1996). Moreover, the efficiency of our algorithm is enhanced by our use of cuts and the fixing of (auxiliary) variables. An essential ingredient of our algorithm is a rounding heuristic that concentrates the computational effort on a reduced size formulation: it applies a depth-first-search heuristic variant of our branch-and-price procedure to the residual problem that remains after the rounded down LP solution has been used as a partial solution. In a sense, we have rediscovered the idea on which Scheithauer and Terno (1995) based their algorithm.

The rest of this paper is organised as follows. We begin by comparing bounds for the cutting stock: we prove the relative strength of the column generation bound using a combinatorial argument. We also propose a pseudo-polynomial heuristic for the problem. In Sect. 2, we present the branch-and-price algorithm and in particular the branching scheme. We also discuss ways to tackle the modified subproblem. Sect. 3 is devoted to the implementation of the branch-and-price algorithm. There, we review the important ingredients of such algorithm (initialisation, early termination of the column generation procedure, branching priorities, ...) and we discuss the enhancements outlined above (cutting planes, variable fixing, early branching, and the rounding heuristic). In Sect. 4, we provide our test results showing what is the effect of various algorithmic settings and we compare our computational results to previous studies.

1. Bin packing and cutting stock problems

The bin packing problem can be described as follows. Given a set of n items of weights (or sizes) $w_i \in \mathbb{N}$ for $i = 1, \ldots, n$ and a capacity $W \in \mathbb{N}$, combine items into groups, called bins, in such a way that the total item weight in each bin does not exceed the capacity W, each item is assigned to one bin and the number of bins used is minimised. The cutting stock problem is a generalisation where each item must be covered $d_i \in \mathbb{N}$ times for $i = 1, \ldots, n$ and a bin may contain multiple copies of the same item. The cutting stock problem finds applications in the paper industry where demands (d_i) for paper reels of various width (w_i) must be met by cutting so-called master reels (of width W). Then the bins correspond to feasible cutting patterns and the objective is to satisfy the demands using the minimum number of master reels.

1.1. Formulations

One way to formulate these problems is to introduce variables x_i^k that represent the number of items of size w_i in bin k and variables y^k that take value one if bin k is used and zero otherwise for $k = 1, \ldots, K$, where K is an upper bound on the number of bins used in an optimum solution. Then, the formulation takes the form:

$$
\begin{aligned}
& Z^F = \min \quad \sum_{k=1}^{K} y^k \\
[F] \qquad & \text{s.t.} && (1)\\
& \sum_{k=1}^{K} x_i^k \geq d_i && i = 1, \ldots, n \qquad (2)\\
& \sum_{i=1}^{n} w_i\, x_i^k \leq W\, y^k && k = 1, \ldots, K\\
& y^k \in \{0, 1\} && k = 1, \ldots, K\\
& x_i^k \in \mathbb{N} && i = 1, \ldots, n;\ \ k = 1, \ldots, K
\end{aligned}
$$

In the case of the bin packing problem, $d_i = 1$ for $i = 1, \ldots, n$ and $x_i^k \in \{0, 1\}$ for $i = 1, \ldots, n$ and $k = 1, \ldots, K$.

Alternatively the problem can be formulated in terms of the variables associated with the selection of feasible item combinations. Let Q be the set of feasible combinations (or feasible bin assignments), i.e. $Q = \{q \in \mathbb{N}^n : \sum_{i=1}^{n} w_i\, q_i \leq W\}$, and λ_q be the number of times combination $q \in Q$ is selected in the solution. Then, the formulation is an integer program of the form:

$$
\begin{aligned}
& Z^M = \min \quad \sum_{q \in Q} \lambda_q \\
[M] \qquad & \text{s.t.} && (3)\\
& \sum_{q \in Q} q_i\, \lambda_q \geq d_i && i = 1, \ldots, n \qquad (4)\\
& \lambda_q \in \mathbb{N} && q \in Q.
\end{aligned}
$$

Formulation $[M]$ was introduced by Gilmore and Gomory (1961). It involves a large number of columns, one for each feasible combination of items.

1.2. Lower bounds

The above formulations are equivalent integer programs but differ in their linear programming relaxation. Let Z_{LP}^F and Z_{LP}^M denote the solution value of the linear programming relaxation of $[F]$ and $[M]$, respectively. Then $L^1 = \lceil Z_{LP}^F \rceil$ and $L^* = \lceil Z_{LP}^M \rceil$ are valid lower bounds on $Z^F = Z^M$. It is well known that $Z_{LP}^F \leq Z_{LP}^M$ as M_{LP} is the Lagrangian dual that arises from the dualization of constraints (2) of $[F]$. Vance et al. (1994) have shown that $Z_{LP}^F = \frac{\sum_{i=1}^{n} w_i d_i}{W}$ and that this trivial lower bound has a worst-case performance ratio of $\frac{1}{2}$. On the other hand, Marcotte (1985) has shown that $\lceil Z_{LP}^M \rceil$ gives the optimum value of the integer problem for certain classes of cutting stock instances. But Marcotte (1986) has also shown that it is NP-hard to decide whether this so called integer round-up property holds for any given instance.

In their study of the bin packing problem, Martello and Toth (1990) have derived an a priori bound, denoted L^2, that is empirically better than L^1 for instances with large

weight items. Their L^2 bound also applies to the cutting stock problem and can be expressed as follows. Given an integer $\alpha:\ 0 \le \alpha \le \frac{W}{2}$, let

$$\begin{aligned} I^1 &= \{i : w_i > W - \alpha\}\ , \\ I^2 &= \{i : W - \alpha \ge w_i > W/2\}\ , \\ I^3 &= \{i : W/2 \ge w_i \ge \alpha\}\ , \end{aligned} \tag{5}$$

and let $D^1 = \sum_{i\in I^1} d_i$ and $D^2 = \sum_{i\in I^2} d_i$. Then,

$$L(\alpha) = D^1 + D^2 + \max\left\{0, \left\lceil \frac{\sum_{i\in(I^2\cup I^3)} w_i\, d_i}{W} \right\rceil - D^2\right\}$$

is a valid lower bound. The maximum value of $L(\alpha)$ is achieved for $\alpha = w_i \le W/2$ for some i and

$$L^2 = \max\{L(\alpha) : \alpha = w_i \le W/2 \text{ for some } i\}$$

can be computed in linear time once the items have been sorted according to size. Martello and Toth (1990) have also proved that $L^2 \ge L^1$.

We now show that $L^2 \le L^*$. The result follows from the fact that any solution of M_{LP} satisfies the combinatorial conditions implicit in the definition of L^2.

Proposition 1. *For any feasible solution $\lambda \in \mathbb{R}_+^{|Q|}$ of M_{LP}, and $0 \le \alpha \le \frac{W}{2}$,*

$$\left\lceil \sum_{q\in Q} \lambda_q \right\rceil \ge L(\alpha)\ .$$

Proof. For the sake of this proof, we consider formulation M_{LP} with equality constraints in (4) which has the same value Z^M_{LP}. Let $Q^k = \{q \in Q : q_i > 0 \text{ for some } i \in I^k\}$, for $k = 1,\ 2,$ and 3, where I^1, I^2, and I^3 are defined in (5). Observe that for all $q \in Q^1 \cup Q^2$, $\sum_{i\in(I^1\cup I^2)} q_i = 1$. Moreover $Q^1 \cap Q^2 = \emptyset$ and $Q^1 \cap Q^3 = \emptyset$. Therefore,

$$\sum_{q\in Q^1} \lambda_q = \sum_{i\in I^1}\sum_{q\in Q} q_i\lambda_q = \sum_{i\in I^1} d_i = D^1\ .$$

Similarly, $\sum_{q\in Q^2} \lambda_q = D^2$, and

$$\sum_{q\in Q\setminus(Q^1\cup Q^2)} \lambda_q = \max\left\{0, \sum_{q\in Q\setminus Q^1} \lambda_q - \sum_{q\in Q^2} \lambda_q\right\}\ .$$

Now the proposition follows from the fact that $\sum_{q\in Q\setminus Q^1} \lambda_q \ge \frac{\sum_{i\in(I^2\cup I^3)} w_i\, d_i}{W}$. Indeed,

$$\sum_{i\in(I^2\cup I^3)} w_i d_i = \sum_{i\in(I^2\cup I^3)} w_i \sum_{q\in Q} q_i\lambda_q = \sum_{q\in Q\setminus Q^1} \left(\sum_{i\in(I^2\cup I^3)} w_i q_i\right)\lambda_q \le W \sum_{q\in Q\setminus Q^1} \lambda_q\ .$$

□

The following example due to Mc Diarmid (see Goulimis, 1988) shows that L^2 can be strictly dominated by L^*. Take $n = 3$, $W = 30$, $w = (15, 10, 6)$, and $d = (1, 2, 4)$, then $Z^F_{LP} = \frac{59}{30}$, $L^1 = L^2 = 2$, but $L^* = Z^M = 3$. Observe however that in this case, the minimum wasted capacity in any item combination is 3, and therefore the capacity can be redefined as $W' = 27$ since it is the maximal weight that can be achieved by any group of items whose total weight does not exceed 30. With the modified capacity W', $L^1 = L^2 = 3$. In practice, we shall preprocess the instances we solve by redefining the capacity as the maximum usable capacity which we obtain by solving a knapsack problem.

Finally, Martello and Toth (1990) have also proposed a reduction scheme that attempts to construct a partial solution using dominant item combinations. A feasible item combination $e \in Q$ is dominant if, for any other feasible combination of items $f \in Q$, the items can be grouped into subsets whose total weight fits into the item weights of the dominant combination, i.e. if $e \in Q$ is a dominant combination that contains k items (counting multiple copies of the same item independently), then $\forall f \in Q$, $\exists$ a partition $(P_1, P_2, \ldots, P_k)$ of the items present in f such that $\sum_{i \in P_l} w_i \leq w_l$ for $l = 1, \ldots, k$. If a dominant combination is identified, it can be used to (partially) meet the demands of the corresponding items, and hence to reduce the problem size while insuring optimality of the partial solution. However, checking the dominance of a feasible combination is impractical, even more so for the cutting stock problem where multiple copies of the same item can be found in a feasible item combination. Martello and Toth have applied this reduction technique to bin packing instances with large item sizes, searching for dominant combinations of up to three items. When a dominant combination is identified, the L^2 bound is computed for the reduced instance, then the instance is relaxed by dropping the smallest item and the process is reiterated. This procedure give rise to yet another "improved" bound, L^3. For the above example, there is no dominant item combination and $L^3 = 2$ which shows that L^3 does not dominate L^*.

1.3. Heuristic solutions

Approximation procedures for the bin packing problem have been developed and theoretically assessed by many authors (see Martello and Toth, 1990). The procedures with best worst-case performance ratios (= 1.222..) are *First Fit Decreasing (FFD)* and *Best Fit Decreasing (BFD)* for a complexity of $O(n \log n)$. For the cutting stock problem, the (FFD) procedure can be described as follows. Iteratively select an item with maximum size from among items with yet unsatisfied demand, and assign it to the first already initialised bin that has sufficient remaining capacity, or if there are none, initialise a new bin with that item. In the (BFD) procedure, the item is assigned to the bin with smallest residual capacity from among those that can accommodate this item. In our computational experiments, we have not found the performance of these procedure to be very good in practice. We have therefore developed another procedure which tends to perform better for instances with small item weights.

The procedure, which we call *Fill Bin* heuristic, constructs *pseudo-dominant* bins, one at a time. Because, the dominance of an item combination, as defined above, is impractical to check, we have relaxed the concept to concentrate on item combinations

with capacity usage equal to the maximum that can be achieved and using the largest items in priority. Let the items be indexed in order of non-increasing sizes

$$w_1 \geq w_2 \geq \ldots \geq w_n \ .$$

Then, an item combination is *pseudo-dominant* if it is a solution x^* of the following knapsack problem

$$\max \left\{ \sum_i w_i x_i \ : \sum_i w_i x_i \leq W \ , x_i \leq d_i \text{ for } i = \ 1, \ldots, n \ , \ x \in I\!N^n \right\}$$

and x^* is lexicographically smaller than any other solution x of this knapsack problem.

The *Fill Bin* heuristic iteratively obtains a pseudo dominant combination of items, x^*, using a standard procedure for the knapsack problem on the sorted set of items for which there is a remaining demand. Then, it uses $\min_i \left\lfloor \frac{d_i}{x_i^*} \right\rfloor$ copies of bin x^*, updates the demand vector, and iterates until all the item demands are covered. Although the procedure is pseudo-polynomial in theory, it is quite fast in practice and, in any case, the computational burden will not be prohibitive in the context of a column generation algorithm that requires solving knapsack subproblems routinely.

The fill bin heuristic is naturally a myopic procedure with obvious drawbacks. Concentrating on filling up bins to the maximum level at the outset of the procedure, might result in leaving aside large items that could not have lead to a "well filled" bin. Then, the bins constructed in the final stages of the procedure may involve a lot of waste. In an attempt to avoid this effect, we also consider a variant of the Fill Bin procedure, where bins are initialised with one copy of the largest item yet to be covered and a pseudo dominant combination is used to fill the remaining bin capacity. We have also consider another variant consisting in constructing maximal bins (bins where the unused capacity is less then the smallest item size) one at the time, using the largest items in priority. But, for the latter variant, the resulting heuristic solutions are similar to solutions of the FFD or BFD heuristics.

2. An algorithm based on column generation

We solve the cutting stock problem using a branch-and-bound procedure based on the lower bound L^*, i.e. we use formulation M whose LP relaxation provides the best known lower bound on the problem. The large number of columns and associated variables of M are dealt with implicitly. At each node of the branch-and-bound tree, a standard column generation procedure is applied to solve the LP relaxation of $[M]$, which is commonly referred to as *the master*. The restricted master LP that contains only a subset of the columns is solved optimally, then the dual solution is used to price out the other columns

which are only known implicitly as the solutions of a knapsack problem. The so-called *column generation subproblem* consists in finding a feasible column with minimum reduced cost. If a negative reduced cost column is found, it is added to the restricted master and the process is reiterated. Otherwise, the current master solution is optimum.

2.1. The column generation subproblem

At the root node, the column generation subproblem takes the form

$$v(\pi) = \max \left\{ \sum_{i=1}^{n} \pi_i x_i \ : \sum_{i=1}^{n} w_i x_i \leq W \ , \ x \in \mathbb{N}^n \right\} \tag{6}$$

where $\pi \in \mathbb{R}_+^n$ is the vector of dual variables associated to the demand covering constraints (4). The solution x^* of this knapsack subproblem defines a column whose reduced cost $1 - v(\pi)$ is minimum. To solve these integer knapsack subproblems, we transform them into 0-1 knapsack problems and use branch-and-bound procedure similar to that presented in Nemhauser and Wolsey (1988), page 455. Indeed, in Vanderbeck (1998), we show that an integer knapsack problem can be transformed into a bounded multiple-class 0-1 knapsack problem. This transformation does not involve a duplication of solution representations as the standard transformation presented in Martello and Toth (1990), and it can be solved as efficiently as the standard 0-1 knapsack problem.

The transformation goes as follows. For each item $i \in \{1, \ldots, n\}$, let the implicit upper bound (i.e the maximum value that can be assumed by the associated column entry) be

$$q_i^{\max} = \min \left\{ d_i \ , \ \left\lfloor \frac{W}{w_i} \right\rfloor \right\} \ ,$$

and define $n_i = \lceil \log(q_i^{\max} + 1) \rceil$ for $i = 1, \ldots, n$ and $n' = \sum_{i=1}^{n} n_i$. Then, the 0-1 vector $q' \in \{0,1\}^{n'}$ associated with feasible column $q \in \mathbb{N}^n$ is defined by the relation $q_i = \sum_{k=0}^{n_i - 1} 2^k \ q'_{p_i + k}$ for $i = 1, \ldots, n$, where $p_i = 1 + \sum_{k=1}^{i-1} n_k$. For notational convenience, we define a profit vector $\pi' \in \mathbb{R}_+^{n'}$ and an item weight vector $w' \in \mathbb{N}^{n'}$ that are the counterpart of π and w for the 0-1 form of the knapsack problem and a vector $m \in \mathbb{N}^{n'}$ that denotes the multiplicities of the binary items, i.e. $\pi'_l = 2^k \, \pi_i$, $w'_l = 2^k \, w_i$, and $m_l = 2^k$ for $l = p_i + k$, $k = 0, \ldots, n_i - 1$, and $i = 1, \ldots, n$. By convention, we use index $i = 1, \ldots, n$ to refer to the original items, and we use index $l = 1, \ldots, n'$ to refer to the 0-1 components of the transformed knapsack problem which takes the form:

$$v(\pi) = \max \left\{ \sum_{n=1}^{n'} \pi'_l \, x_l \ : \sum_{n=1}^{n'} w'_l \, x_l \leq W \, , \ \sum_{l=p_i}^{p_i + n_i - 1} m_l \, x_l \leq q_i^{\max} \ \forall i \ , \ x \in \mathbb{N}^{n'} \right\} \tag{7}$$

2.2. The branching scheme

In an algorithm based on column generation, branching is not straightforward as the rounding of master variables is usually not a viable scheme. In Vanderbeck (1996), we discussed this issue at length and proposed a general framework to branch in a branch-and-price algorithm. We compared different branching schemes for the cutting stock problem, and suggested to use the following:

Given a fractional solution λ of M_{LP}, find a subset of columns

$$\hat{Q} = \{q \in Q : \; q'_l = 0 \;\; \forall l \in O \text{ and } q'_l = 1 \;\; \forall l \in P\} \tag{8}$$

such that $\alpha = \sum_{q \in \hat{Q}} \lambda_q$ is fractional and branch by enforcing that either

$$\sum_{q \in \hat{Q}} \lambda_q \leq \lfloor \alpha \rfloor \, , \tag{9}$$

or

$$\sum_{q \in \hat{Q}} \lambda_q \geq \lceil \alpha \rceil \, . \tag{10}$$

The subsets O and $P \subset \{1, \ldots, n'\}$ completely characterise $\hat{Q}$. For any fractional master solution, there exist subsets O and P with $|O| + |P| \leq \lfloor \log f \rfloor + 1$ for which the associated α is fractional, where $f = \sum_{q \in Q} (\lambda_q - \lfloor \lambda_q \rfloor)$ (Vanderbeck, 1996).

At a branch-and-bound node u, after such branching constraints have been added to the master problem, the master LP formulation takes the form:

$$\begin{aligned}
& & \min \quad & \sum_{q \in Q} \lambda_q & & \\
& \left[M^u_{LP}\right] & \text{s.t.} & & & \\
& & \sum_{q \in Q} q_i \, \lambda_q \geq \; & d_i & \quad i = 1, \ldots, n & \\
& & \sum_{q \in Q^j} \lambda_q \leq \; & K^j & \quad \forall \; j \in G^u & (11) \\
& & \sum_{q \in Q^j} \lambda_q \geq \; & L^j & \quad \forall \; j \in H^u & (12) \\
& & \lambda_q \geq \; & 0 & \quad q \in Q \, , &
\end{aligned}$$

where G^u and H^u are sets of branching constraints of the form (9) and (10), respectively, that define the problem at node u and the pairs $(Q^j \, , \; K^j)$ (resp. $(Q^j \, , \; L^j)$) for $j \in G^u$ (resp. $j \in H^u$) are respectively the associated column subsets, $\hat{Q} \subseteq Q$, and rounding down (resp. up) of the fractional values α.

The associated column generation subproblem takes the form

$$v(\pi, \mu, \nu) = \max \sum_{l=1}^{n'} \pi'_l \, x_l - \sum_{j \in G^u} \mu_j \, g_j + \sum_{j \in H^u} \nu_j \, h_j$$

$$[SP^u] \qquad \text{s.t.} \tag{13}$$

$$\sum_{l=1}^{n'} w'_l \, x_l \leq W$$

$$\sum_{l=p_i}^{p_i+n_i-1} m_l \, x_l \leq q_i^{\max} \qquad \forall i \tag{14}$$

$$g_j \geq 1 - \sum_{l \in O^j} x_l - \sum_{l \in P^j} (1 - x_l) \qquad \forall j \in G^u \tag{15}$$

$$h_j \leq (1 - x_l) \qquad \forall l \in O^j \, , \; j \in H^u \tag{16}$$

$$h_j \leq x_l \qquad \forall l \in P^j \, , \; j \in H^u \tag{17}$$

$$x_l \in \{0, 1\} \qquad \forall l = 1, \ldots, n' \, ,$$

where $(\pi, \mu, \nu) \in \mathbb{R}_+^{m+|G^u|+|H^u|}$ is an optimal dual solution of the restricted master LP, and π' and w' are the 0-1 form equivalent of π and w, respectively.

Observe that for branching constraints with $|O^j|+|P^j| = 1$, the modifications in the subproblem simply amount to amending the profits π' and there is no need to introduce an extra variable g_j or h_j. When $|O^j| + |P^j| \leq 2 \; \forall j$ the subproblem can be formulated using quadratic terms instead of introducing extra variables g_j or h_j. For the general case, the subproblem can be approached by dualizing the extra constraints (15-17). The idea is to solve the subproblem using a branch-and-bound procedure similar to the one used for the bounded multiple-class 0-1 knapsack problem, but where upper bounds are solutions of the LP relaxation of the Lagrangian function. As the Lagrangian function is a bounded multiple-class 0-1 knapsack problem, its LP relaxation admits a closed form solution (Vanderbeck, 1998) and therefore Lagrangian parameters can be heuristically chosen that minimise this LP bound. Alternatively, after relaxation of the class bound constraints (14), the modified subproblem (13) can be transformed into a 0-1 knapsack problem with additional variables and Special Ordered Set (SOS) constraints (also called Generalised Upper Bound (GUB) constraints).

The transformation can be briefly described as follows: for each branching constraint with $|O^j| + |P^j| > 1$, define a group made of the variables x_l involved; merge groups that are not disjoint; for each group, discard the variables concerned and replace them by new binary variables, one for each possible combinatorial combination of the variables concerned; compute the weight and the cost of each of the combinations newly defined; solve the resulting 0-1 knapsack with additional SOS constraints enforcing that, in each group, at most one of the newly defined variables can be selected in the solution. Of course, this transformation is only viable when the branching constraints involve a few variables x_l and tend to concern disjoint set of variables. There is an algorithm due to Johnson and Padberg (1981) to compute the LP bound of a knapsack problem with disjoint SOS constraints.

3. Implementation details and enhancement features

There are many practical issues that are important in designing an efficient implementation of a column generation based algorithm. We now describe our implementation of the important features of our algorithm. We also discuss the enhancements that we have tested.

3.1. Initialisation

At the outset of the algorithm, we perform some preprocessing of the data: we eliminate duplicate item sizes by grouping the corresponding demands and we redefine the bin capacity as the maximal usable capacity. Then we compute the L^2 bound and initialise the incumbent with the best of the heuristic solution obtained using FFD, BFD, the Fill Bin heuristic and its variant. If the incumbent has cost equal to L^2, the algorithm stops, else the constraint $\sum_{q\in Q}\lambda_q \geq L$ with L initially equal to L^2 is added to the master to round-up its LP solution and to ensure that the number of columns used is integer.

At each branch-and-bound node an initial feasible LP solution is required to start-up the column generation procedure. At the root node, the master is initialised with the columns of the best heuristic solutions, but this will not guarantee the presence of a feasible LP solution further down the tree when branching constraints have been added. For that purpose, we include in the formulation a single artificial column whose entries equal to 1 for all rows corresponding to a "greater or equal to" constraint and whose cost is equal to L. If the artificial variable is still in the LP solution upon termination of the column generation procedure, its cost is multiplied tenfold and the column generation procedure is reiterated. We have also tried an initialisation of the column generation procedure using an "unit matrix", i.e., a matrix made of one column for each item with a single non-zero entry, $q_i = q_i^{\max}$. The resulting algorithm performance is significantly worse (in terms of number of master iterations and columns generated).

3.2. Intermediate lower bounds

The column generation procedure can have a slow convergence. To avoid this potential drawback, it is customary to terminate the procedure early by monitoring the master LP duality gap. At each branch-and-bound node u, the purpose of solving the master LP M^u_{LP} is to obtain the lower bound $L^*_u = \left\lceil Z^{M^u}_{LP} \right\rceil$. However, the solution of the restricted master LP, which we denote be $\overline{Z}^{M^u}_{LP}$, gives only an upper bound on $Z^{M^u}_{LP}$. On the other hand, lower bounds on $Z^{M^u}_{LP}$ are readily available at each column generation iteration. The Lagrangian lower bound is the value of the Lagrangian problem resulting from the dualization of constraints (2) for the current dual solution (π, μ, ν). In Vanderbeck and Wolsey (1996), we propose intermediate lower bounds that improve slightly on the Lagrangian bound. In the case of the cutting stock problem, where the column cost are all equal to 1, there is an even tighter lower bound that has been introduced by Farley (1990).

Farley's bound for the cutting stock problem can be expressed as

$$\frac{\overline{Z}_{LP}^{M^u}}{v(\pi, \mu, \nu)} \leq Z_{LP}^{M^u}$$

where $v(\pi, \mu, \nu)$ is the solution of the column generation subproblem (13) and (π, μ, ν) is an optimal dual solution of the restricted master LP. Therefore, at each column generation iteration we compute

$$L^i = \max \left\{ L^i, \left\lceil \frac{\overline{Z}_{LP}^{M^u}}{v(\pi, \mu, \nu)} \right\rceil \right\}, \tag{18}$$

the best intermediate lower bound. At the root node L^i is initially set equal to L^2, while at other nodes, the bound of the ancestor node is used. The column generation procedure is terminated if

$$L^i \geq \left\lceil \overline{Z}_{LP}^{M^u} \right\rceil \tag{19}$$

meaning that $L^i = L_u^*$, or if

$$L^i \geq \textit{ incumbent} \tag{20}$$

meaning that $L_u^* \geq$ *incumbent.*

In comparison the Lagrangian bound at the root node is

$$\overline{Z}_{LP}^{M} + K\ (1 - v(\pi, \mu, \nu)) \leq Z_{LP}^{M} \tag{21}$$

where K is a valid upper bound on the number of bins in an optimum solution. In the final stages of the column generation procedure, $\overline{Z}_{LP}^{M} \leq K$ and the minimum reduced cost converges towards zero by negative values, i.e. $1 - v(\pi, \mu, \nu) \cong -\epsilon$, where $\epsilon \in \mathbb{R}_+$ and $\epsilon \to 0$. Then, Farley's bound dominates the Lagrangian bound since

$$\frac{\overline{Z}_{LP}^{M}}{1+\epsilon} \geq \overline{Z}_{LP}^{M}\ (1 - \epsilon) \geq \overline{Z}_{LP}^{M} - K\ \epsilon$$

for ϵ sufficiently small. The same is true at any branch-and-bound node.

The column generation termination criteria (19-20) give rise to a priori bounds on the minimum column reduced cost (Vanderbeck and Wolsey, 1996). Termination of the column generation procedure will take place unless

$$v(\pi, \mu, \nu) \geq \max \left\{ 1 + \epsilon, \frac{\overline{Z}_{LP}^{M}}{\left\lceil \overline{Z}_{LP}^{M} \right\rceil - 1}, \frac{\overline{Z}_{LP}^{M}}{\textit{incumbent} - 1} \right\} \tag{22}$$

where ϵ is the precision level of computations. This a priori subproblem bound is used as an artificial incumbent in the solution of the knapsack subproblem.

3.3. Selection of branching constraints and branching priorities

For the cutting stock problem, the lower bound L^* that is obtained at the root node is often the optimum solution value. Then, branching typically does not lead to improved lower bounds, but aims at reducing the fractionality of the master solution. Hence, we adopt a depth-first tree-search strategy, processing the deepest branch-and-bound node from amongst the many for which the a priori bound equals the current lower bound. However, it remains important to select branching rules that give rise to balanced subtrees, i.e. that partition the solution set into roughly equal cardinality subsets. Indeed, the strategy that consists in using a branching constraint that is very restrictive on one branch but not on the other, in the hope that an optimum solution will be found on the former branch, is risky. If the bet is unsuccessful (i.e. no optimum solution can be found on the very restricted branch), backtracking must take place. Then, the computational time spend in exploring the restricted branch is wasted and the remaining problem associated with the not-very-restricted branch is almost as hard as the ancestor node problem. Hence an algorithm based on an unbalanced branching scheme is typically not stable, yielding low computing time when it gets lucky but requiring very large computing time for some instances.

Branching by fixing (or bounding) master variables, i.e. choosing to use (or not to use) a specific cutting pattern, is the sort of unbalanced branching scheme that can lead to large computing time when unlucky. On the other hand, the family of branching constraints defined by (8), yield a more balanced tree and in particular, the branching rules (8) that are the simplest to implement are also the ones that lead to the most balanced subtrees. Branching constraints defined by sets O^j and P^j of higher cardinality involve fewer feasible item combinations and are therefore more specific. Hence, we use primarily branching constraints defined by a single subproblem variable, i.e. with $|P^j| = 1$. In fact, for instances of the cutting stock problem, successive application of the simplest branching rule

$$\sum_{q\in Q:q'_l=1} \lambda_q \le \lfloor \alpha \rfloor \quad \text{or} \quad \sum_{q\in Q:q'_l=1} \lambda_q \ge \lceil \alpha \rceil$$

will typically allow us to eliminate all fractional solutions. Instances of the bin packing problem on the other hand also require branching on pairs of knapsack components, resulting in a branching rule of the form

$$\sum_{q\in Q:q'_l=q'_k=1} \lambda_q \le \lfloor \alpha \rfloor \quad \text{or} \quad \sum_{q\in Q:q'_l=q'_k=1} \lambda_q \ge \lceil \alpha \rceil \ .$$

Indeed, as item demands are typically equal to 1, $q_i^{\max} = 1$ for many items and $\sum_{q\in Q:q'_l=q_i=1} \lambda_q = 1$ although λ might be fractional. To limit the extend of the resulting modifications in the subproblem, we only consider disjoint subsets, P^j, of binary knapsack item. None of the instances that we have tackled has required the use of branching constraints with $|P^j| > 2$ or with non-disjoint sets P^j. Observe that we do not need to consider branching constraints with $O^j \neq \emptyset$ because they are the complement of the former: $\sum_{q\in Q:q'_l=0} \lambda_q = \sum_{q\in Q} \lambda_q - \sum_{q\in Q:q'_l=1} \lambda_q$, and the integrality of $\sum_{q\in Q} \lambda_q$ has been enforced.

The subproblem modifications that result from these simple branching rules can be accounted for in a standard branch-and-bound procedure for the 0-1 knapsack problem. When dealing with branching constraints with $P^j = \{l\}$, we simply replace π'_l by $\pi'_l - \mu_j$ or $\pi'_l + \nu_j$ in (7), where μ_j or ν_j are the dual price associated with constraints (11) or (12) respectively. When a branching constraint is defined by a component set $P^j = \{l,\ k\}$ (disjoint from other such branching pairs at the node), we define an extra variable $x_{l\,k}$, with weight $w(l,k) = w'_l + w'_k$ and profit $\pi(l,k) = \pi'_l + \pi'_k - \mu_j$ or $\pi(l,k) = \pi'_l + \pi'_k + \nu_j$. We solve the resulting 0-1 problem using a depth-first-search branch-and-bound algorithm similar to that used for a standard 0-1 knapsack problem, handling the extra constraint $x_l + x_k + x_{l\,k} \leq 1$ implicitly: when one of the variables x_l, x_k, or $x_{l\,k}$ is set to one in a forward move, the others are set to zero; if backtracking occurs, this setting to zero is undone. The LP bounds are computed ignoring the SOS constraints of the form $x_l + x_k + x_{l\,k} \leq 1$, and the class upper bound constraints (7).

We have found that the performance of the algorithm is relatively sensitive to the order in which variables are considered for branching. A scheme that performs well is to give priority to knapsack components l with large weights w'_l. Given a fractional solution $\lambda \in \mathbb{R}_+^{|Q|}$ of the master LP, we consider successively the 0-1 components l in non-increasing order of their weight w'_l, we compute $\alpha^l = \sum_{q \in Q: q'_l = 1} \lambda_q$, and we stop as soon as we find an α^l with fractionality $f = \min\{\alpha^l, 1 - \alpha^l\}$ bigger than a prescribed threshold ϵ. The smaller ϵ, the more priority is given to branching on components with large weights. We set the threshold ϵ to be

$$\epsilon = \frac{\sum_i w_i\, d_i}{100\ (w_{\max} - w_{min})\ \sum_i d_i}\ .$$

In this manner, ϵ is set lower for instances that have a wide range of item weights. If no component l gives rise to a fractional α^l, we search for a pair (l,k) on which to branch in a similar way. Then, we explore first the branch defined by $\sum_{q \in \hat{Q}} \lambda_q \geq \lceil \alpha \rceil$ and search the successor of that node before returning to the branch $\sum_{q \in \hat{Q}} \lambda_q \leq \lfloor \alpha \rfloor$. The computational tests of the next section show that giving priority to large weight items i rather than to large weight 0-1 components l, yields a larger branch-and-bound tree and a 70% increase in computation time.

3.4. Cutting planes

We have experimented with the use of cutting planes to strengthen the master LP formulation. First, we consider *feasibility cuts* similar to those used for the Vehicle Routing Problem by Desrosiers and Soumis (1996). In any feasible IP solution to the cutting stock problem, the number of bins covering a subset $S \subseteq \{1, \ldots, n\}$ of items must be larger or equal to the total weight required by these items divided by the bin capacity:

$$\sum_{q \in Q: q \cap S \neq \emptyset} \lambda_q \geq \left\lceil \frac{\sum_{i \in S} w_i\, d_i}{W} \right\rceil \tag{23}$$

These inequalities are clearly valid since (4) implies that the master solution will verify

$$\sum_{i\in S} w_i d_i = \sum_{i\in S} w_i \sum_{q\in Q} q_i \lambda_q = \sum_{q\in Q: q\cap S\neq\emptyset} \left(\sum_{i\in S} w_i q_i\right) \lambda_q \;\leq\; W \sum_{q\in Q: q\cap S\neq\emptyset} \lambda_q \;.$$

An inequality (23) is unlikely to be violated by the master LP solution if $\sum_{i\in S} w_i q_i < W$ in several of the columns involved in the constraint, which is typically the case when items in S are combined with items not in S in columns of the current solution. Therefore, the search for cuts should concentrate on small cardinality sets S or on subsets of items that share the same bins in the LP solution.

We concentrate on inequalities involving a single item, $S = \{i\}$, for which the right-and-side can be made even tighter:

$$\sum_{q\in Q: q_i>0} \lambda_q \geq \left\lceil \frac{d_i}{q_i^{\max}} \right\rceil \tag{24}$$

where $q_i^{\max}$ denotes the maximum number of copies of item i in a bin. The addition of such a constraint in the master involves a modification of the reduced cost of columns for which $q_i > 0$. Observe that

$$\{q \in Q :\; q_i > 0\} = Q \setminus \{q \in Q :\; q'_l = 0 \;\forall l \in O^i\}$$

where $O^i = \{p_i, \ldots, p_i + b_i - 1\}$ and hence the subproblem modifications resulting from these cuts are similar to those resulting from the branching scheme we described. To suit the modification scheme that we have implemented for branching, we further restrict our attention to cuts involving items for which $q_i^{\max} \leq 3$ and hence $b_i \leq 2$.

We also use the trivial cut

$$\sum_q \lambda_q \geq L$$

where L is the current best lower bound on the optimum IP solution. Such rounding of the objective function has been used in other studies, e.g. for vehicle routing problems (Desaulniers et al. , 1994, and Desrosiers et al., 1994). We have also considered the use of a cut on the amount of unused capacity as proposed by Valerio de Carvalho (1996). Knowing a lower bound L on the number of bins and the total capacity requirements, one can derive a *lower bound on the total waste* (unused capacity)

$$\mathit{waste} \;\geq L\; W - \sum_i w_i d_i \tag{25}$$

where $\mathit{waste} = \sum_q (W - \sum_i w_i q_i)\lambda_q$. Note that enforcing this minimum waste can be done by adding an artificial item of unit size with demand equal to the minimum waste (although this creates symmetry problem if the instance already contains an item of unit size or a very small size item). Alternatively, one can enforce the minimum waste by working with a set partitioning formulation instead of a set covering one (i.e. imposing strict equality in the demand covering constraints).

We also experiment with optimality cuts that can be briefly described as follows. If $\sum_{q\in Q:q'_l=1}\lambda_q = \alpha \notin \mathbb{N}$, we temporarily set a new item with weight w'_l and demand $\lceil\alpha\rceil$ and update the demand $d_i = \max\{0,\ d_i - 2^k\ \lceil\alpha\rceil\}$ for the item i associated with the knapsack component $l = p_i + k$. Then, we compute the associated L^2 bound. If it is larger than the incumbent solution value, we add the cut

$$\sum_{q\in Q:q'_l=1} \lambda_q \le \lfloor\alpha\rfloor \tag{26}$$

in the master. We refer to these cuts as L^2 *cuts*. Similarly, we search for L^2 cuts involving a pair of knapsack components.

In practice, the L^2 cuts are particularly useful to test whether the auxiliary variable that has been chosen for branching can be bounded and hence, whether one of the descendant branch-and-bound node can be pruned without even being processed. In fact, we have noticed that the typical branch-and-bound search basically goes down the tree on the successive branches $\sum_{q\in\hat{Q}}\lambda_q \ge \lceil\alpha\rceil$, until it encounters a node where $\sum_{q\in\hat{Q}}\lambda_q \ge \lceil\alpha\rceil$ means that an extra bin would be required. Then, it backtracks one level, moves to the branch $\sum_{q\in\hat{Q}}\lambda_q \le \lfloor\alpha\rfloor$ and goes depth from there. Proving that $\sum_{q\in\hat{Q}}\lambda_q \ge \lceil\alpha\rceil$ results in an extra bin typically requires quite a few column generation iteration. In comparison, going down the tree on the successive branches $\sum_{q\in\hat{Q}}\lambda_q \ge \lceil\alpha\rceil$ often requires only a single column generation iteration per node. Therefore, avoiding the computational burden of pruning the odd node, can potentially speed up the algorithm. Testing if $\sum_{q\in\hat{Q}}\lambda_q \le \lfloor\alpha\rfloor$ is a valid L^2 cut may allow us to cheaply prune the node without having to use the column generation procedure. We refer to this test as L^2 *pruning*.

3.5. Variable fixing

Variable fixing refers to one's attempt to fix or bound the value of variables based on consideration of optimality or feasibility. In particular, reduced cost fixing uses the reduced cost information to make such decisions: e.g. if an integer variable is at value zero in the current LP solution and has reduced cost bigger than the gap between lower bound and incumbent, it can be fixed to zero. The question is whether similar "tricks" can be used in a column generation framework. We argue that this should not be attempted on the master variables λ_q themselfs but on the auxiliary variables on which we branch. Indeed, bounding the value of a particular λ_q may not be very helpful (as there are so many other columns and alternative solutions) and it may lead to intractable modifications to the subproblem. On the other hand, bounding the number of columns sharing a specific property (e.g. including a particular item) does not have these drawbacks. With this perspective, the cuts that we described above can be viewed as the fixing of auxiliary variables.

We now examine the reduced cost fixing of auxiliary variables. An auxiliary variable (of the branching scheme) can be fixed to 0 if, when it takes value 1, the Lagrangian intermediate lower bound (21) exceeds the incumbent value. The test is as follows: Take an auxiliary variable defined by a subset $\hat{Q} \subset Q$ that has value 0 in the current

LP solution, i.e. take $\hat{Q}$ such that $\sum_{q\in\hat{Q}} \lambda_q = 0$ (for instance, consider an knapsack component that is not in any columns selected in the master LP solution). Compute the reduced cost of columns in $\hat{Q}$, i.e. modify the column generation subproblem by fixing subproblem variables to insure that the solution belongs to $\hat{Q}$. Let $\hat{v}(\pi, \mu, \nu)$ denotes the optimal value of this subproblem. Test whether the intermediate lower bound that results from taking at least 1 column in $\hat{Q}$ exceeds the incumbent value. i.e. whether

$$\left\lceil \overline{Z}^M_{LP} + 1\ (1 - \hat{v}(\pi, \mu, \nu)) + (K - 1)\ (1 - v(\pi, \mu, \nu)) \right\rceil \geq \textit{incumbent}$$

where K is an upper bound on the number of bins in an optimum solution and $v(\pi, \mu, \nu)$ is the solution of the unconstrained subproblem.

Note that, since we have combined the phase 1 and phase 2 of the LP solution by introducing artificial variables in the master, this test encompasses both feasibility and optimality considerations. We have used this scheme to fix knapsack components to zero and hence reduce the size of the subproblem. However there is not much benefit to be drawn. Moreover the test is computationally costly as it requires solving an extra subproblem. We have therefore abandoned this line of investigation.

3.6. Early branching

Early Branching refers to branching before the computation of the bound at a branch-and-bound node has been completed. The purpose is to truncate a potentially slow convergence of the bound computation procedure. In our case, the node computations are stopped when one of the column generation termination criteria (19-20) or (22) is satisfied. In practice, we use an "artificial incumbent" in stopping rule (20). The stopping criteria we use is $L^i \geq L + 1$, where L is the current lower bound on the optimal solution. As long as $L + 1 < \textit{incumbent}$, the current best integer solution, the node that are pruned on the basis of $L^i \geq L + 1$ are returned to the queue of nodes that require further processing. This scheme makes the performance of our algorithm less dependent on the quality of the incumbent initially obtained with a heuristic. Moreover, as for most instances, L^* is the optimum value, nodes temporarily interrupted are typically pruned at a later stage.

We have experimented with stopping the processing of a node even earlier. The first scheme we tried consisted in introducing a tolerance in the stopping conditions, i.e. stop if

$$L^i + TOL \geq \left\lceil \overline{Z}^{M^u}_{LP} \right\rceil \quad \text{or} \quad L^i + TOL \geq \textit{incumbent}$$

where $TOL \geq 0$. The scheme yielded much larger branch-and-bound trees and computation times (even for TOL as small as 0.001). Our interpretation is that the columns generated in the last stages of the column generation procedure are the "best" ones, i.e. the ones that are the most likely to appear in an optimal solution. We have then used another scheme that consists in stopping the column generation when the last column that we have generated has a reduced cost that is "close to" zero, i.e. when $v(\pi, \mu, \nu) \leq 1+\epsilon$. The latter scheme can be useful for larger instances where computational rounding errors are significant.

3.7. Rounding heuristic

A rounding heuristic is a procedure that attempts to find a "good" integer solution by rounding the current LP solution. Here again the standard procedure that consists in rounding the fractional variables does not straightforwardly apply in the context of a column generation procedure, as bounding λ_q's destroys the structure of the subproblem. Moreover, apart from this difficulty, the standard procedure is doomed to fail because the restricted set of columns in the master formulation typically does not contain an optimal integer solution. Instead, we use a procedure based on the rounding of the auxiliary variables that are used for branching and we continue to generate columns after rounding. That is we perform a heuristic depth first search of the branch-and-bound tree, and, along the way, we construct a partial integer solution by rounding down intermediate master LP solutions. The scheme is applied a every branch-and-bound node as follows.

We first take the round down of the master LP solution λ at the current branch-and-bound node as a partial solution. Then, we consider the remaining master LP where the right-hand-sides have been appropriately updated, i.e. d_i has been replaced by $\max\{0\,,\ d_i - \sum_q q_i \lfloor \lambda_q \rfloor\}$. We select a branching variable as we would do for normal branching and we bound it down, i.e we add the constraint $\sum_{q \in \hat{Q}} \lambda_q \geq \lceil \alpha \rceil$ in the remaining master problem, unless the L^2 pruning procedure reveals that we should explore solutions where $\sum_{q \in \hat{Q}} \lambda_q \leq \lfloor \alpha \rfloor$. Then, we return to the column generation procedure but with a smaller master problem and a smaller subproblem, since typically d_i, and hence $q_i^{\max}$ and n_i are now reduced for some items i. The procedure is reiterated until an integer solution is found or the intermediate lower bound shows that no solution better than the incumbent will be found along this route.

The rounding heuristic is a way to concentrate the computational effort on the fractional part of the solution: after extracting a partial solution, the master problems is a covering problem where the remaining demands are those that were covered by fractional columns and the column generation process only produces columns containing those items that account for the fractionality of the solution. The L^2 pruning is more effective on the reduced size problem and the feasibility cuts are used more often as more items have $q_i^{\max} \leq 3$ after updating the demands. This heuristic is very helpful as we shall see in the computational results.

4. Computational results

For our computational tests, we have used a few real-life instances, but, for the main part, we worked with randomly generated instances. For the cutting stock problem, we consider instances similar to those generated by Vance (1996), with 50 items, an average demand of 50, and item weights generated uniformly in intervals [1, 7500], [1, 5000], and [1, 2500] for a bin capacity of 10 000. Moreover, we also generated instances with item weights in [500, 5000] and [500, 2500] because, in the real life instances we have seen, the item size is hardly smaller than 5% of the bin size and rarely larger than 50% of the bin size. These different item weight distributions are referred to as class 1 to class 5. We also generate class 4 problems with an average item demand of 100, in order

to test the effect of demand size for the item size distribution that seems to reflect more closely real life instances.

For the bin packing problem, we have followed the foot steps of Martello and Toth (1990) and Vance et al. (1994). We consider instances with item sizes generated uniformly in [1, 100], [20, 100], and [50, 100], and bin capacity equal to 100, 120, and 150. We tried all combinations, generating instances with 500 items. The instances with $w_i \sim U[20, 100]$ are from the OR-Library (Beasley 1990). We also try the algorithm on a batch of problems with 1000 items and on the difficult "triplets" instances described in the OR-Library (Beasley 1990). The latter have been generated in such a way that the optimal solution consists of bins filled at capacity with exactly three items.

The code of our algorithm is in C. We use CPLEX 3.0 to solve the master LPs. The column generation subproblems are solved using our own implementation of a branch-and-bound code for the 0-1 knapsack with real number data (based on the presentation found in Nemhauser and Wolsey, 1988, p.455). The computations have been carried out on an HP9000/712/80 (80MHz) workstation with 64Mb of main memory (SPECint92 = 84.1, SPECint95 = 22.1, SPECfp92 = 122, SPECfp95 = 29.2).

4.1. Testing the algorithm

To discover to what extend the algorithmic features that we described do help in solving cutting stock and bin packing instances, we have carried out comparative tests. We have used as a test bed a set of 34 problem instances whose characteristics are given in Table 1. The first 10 instances are real-life cutting stock problems, the others are randomly generated instances which are not trivially solved by the heuristic. Problems 11 to 20 are cutting stock instances with 20 items and average demand of 50. They are 2 instances for each of the five weight distribution classes described above. Problems 21 to 34 are bin packing instances. Instances 21 to 24 have item weights drawn from $U[1, 100]$, instances 25, 26, and 29 to 34 have item weights drawn from $U[20, 100]$, instances 27 and 28 have item weights drawn from $U[50, 100]$. The bin capacity is 100 for instances 21 and 22, 120 for instances 23 to 28, 150 for instances 29 to 34. The number of items are 500 for instances 21 to 28 and 31 to 32, 250 for instances 29 and 30, and 1000 for instances 33 and 34.

For each instance, Table 1 gives the number of items n, the number of component in the associated 0-1 knapsack subproblem n', and the characteristics of the item weights and demands after preprocessing of the data. The item weights are expressed as fractions of the bin capacity. *min w* and *max w* are the smallest and largest item weight, while *aver* $w = \frac{\sum_i d_i w_i}{\sum_i d_i}$. Similarly, *min d*, *aver d* and *max d* give the minimum, average, and maximum demands.

Table 2 contains the solutions of these test instances when solved with a *benchmark* version of our algorithm. The columns contain respectively the instance number *nb*, the L^2 bound, the L^* bound, the optimal solution (IP), the best of the FFD and BFD heuristics ($H1$), the best of the two variants of the Fill Bin heuristic ($H2$), the depth of the branch-and-bound tree (*depth*), the number of nodes that have been processed in the branch-and-bound tree (*nod*), the number of times the master LP has been solved ($mast$), the number of times the column generation subproblem has been solved (SP),

Table 1. Instances used to test the algorithm

nb	name	n	n'	min w	aver w	max w	min d	aver d	max d
1	7p18	7	22	0.1206	0.1450	0.1758	5	88.7	337
2	11p4	11	46	0.0519	0.0744	0.1651	25	123.4	318
3	12p19	12	39	0.0797	0.1354	0.2640	4	14.1	31
4	14p12	14	50	0.0524	0.0968	0.2143	2	40.6	174
5	d16p6	16	34	0.2366	0.3399	0.4911	5	6.8	10
6	25p0	25	80	0.0797	0.1336	0.2640	4	34.4	337
7	28p0	28	102	0.0269	0.1186	0.2603	2	56.5	337
8	30p0	26	86	0.0759	0.1230	0.2784	2	28.0	252
9	d33p20	23	53	0.1011	0.2143	0.3177	1	5.8	16
10	d43p21	32	74	0.0521	0.2194	0.2998	1	5.6	17
11	20b50c1p3	20	47	0.0048	0.3440	0.7096	13	50.0	87
12	20b50c1p5	20	46	0.0323	0.3520	0.7490	2	50.0	107
13	20b50c2p1	20	60	0.0152	0.2483	0.4856	7	50.0	104
14	20b50c2p2	20	69	0.0031	0.1777	0.4641	5	50.0	93
15	20b50c3p1	20	70	0.0539	0.1462	0.2476	4	50.0	108
16	20b50c3p4	20	74	0.0025	0.1474	0.2389	4	50.0	101
17	20b50c4p1	20	57	0.0529	0.2286	0.4556	7	50.0	80
18	20b50c4p4	20	56	0.0576	0.2297	0.4444	7	50.0	111
19	20b50c5p2	20	70	0.0573	0.1398	0.2336	4	50.0	84
20	20b50c5p3	20	67	0.0520	0.1437	0.2462	4	50.0	96
21	u1t100W100n500p8	98	164	0.0100	0.5127	0.9900	1	5.1	12
22	u1t100W100n500p9	99	168	0.0100	0.4870	0.9900	1	5.1	10
23	u1t100W120n500p3	99	179	0.0083	0.4210	0.8250	1	5.1	11
24	u1t100W120n500p7	99	182	0.0083	0.4219	0.8250	1	5.1	12
25	u20t100W120n500p1	81	129	0.1667	0.5021	0.8333	1	6.2	12
26	u20t100W120n500p19	81	132	0.1667	0.4891	0.8333	1	6.2	12
27	u50t100W120n500p2	50	61	0.4167	0.6305	0.8250	2	10.0	18
28	u50t100W120n500p3	50	61	0.4167	0.6230	0.8250	4	10.0	19
29	u20t100W150n250p0	71	123	0.1333	0.3942	0.6667	1	3.5	8
30	u20t100W150n250p12	76	127	0.1333	0.4197	0.6667	1	3.3	8
31	u20t100W150n500p3	81	150	0.1333	0.4076	0.6667	1	6.2	13
32	u20t100W150n500p8	81	153	0.1333	0.3914	0.6667	2	6.2	12
33	u20t100W150n1000p3	81	155	0.1333	0.4109	0.6667	3	12.3	21
34	u20t100W150n1000p14	81	155	0.1333	0.3939	0.6667	3	12.3	22

and the total CPU time in seconds (*time*) inclusive of the initialisation and heuristic computations. The last line of Table 2 summarises the results by providing averages over the 34 instances. In particular it shows that in all cases $L^* = IP$, while the L^2 bound underestimates the optimum number of bins by 0.32%. The H1 and H2 heuristics are on average 1.19% and 0.99% above the optimum value respectively.

In the benchmark version of the algorithm, the master is initialised with the columns associated with the best heuristic solutions and an artificial column having entries equal to 1 in each row corresponding to a greater or equal to constraint. Termination of the column generation algorithm at each node occurs when either condition (19) or condition (20) is satisfied, but the associated a priori lower bound (22) on the subproblem solution is not used. Branching is done on a single knapsack component where possible and on a pair of components otherwise, priority being given to components with large weights. The feasibility cuts and L^2 cuts are not used. The rounding heuristic is turned off.

Table 2. Computational results for the benchmark version of the algorithm

nb	L^2	L^*	IP	H1	H2	depth	nod	mast	SP	time
1	91	91	91	94	94	2	3	5	2	0.26
2	101	101	101	103	101	0	0	0	0	0.25
3	23	23	23	24	23	0	0	0	0	0.97
4	56	56	56	57	56	0	0	0	0	0.26
5	37	38	38	41	40	7	9	17	9	0.47
6	115	115	115	121	116	33	35	100	66	4.52
7	188	188	188	189	191	59	65	199	139	8.85
8	90	90	90	93	91	61	68	173	111	6.31
9	29	29	29	30	29	0	0	0	0	1.41
10	40	40	40	41	40	0	0	0	0	12.98
11	345	348	348	352	360	10	13	29	19	1.17
12	353	362	362	371	369	15	17	36	20	1.15
13	249	250	250	253	253	31	35	67	35	2.05
14	178	178	178	179	185	31	33	83	51	2.65
15	147	147	147	148	148	57	63	114	56	4.41
16	148	148	148	149	149	48	54	108	59	4.51
17	229	229	229	233	235	31	33	83	51	2.48
18	230	231	231	235	233	21	23	58	36	1.79
19	140	140	140	142	141	49	52	107	57	5.23
20	144	144	144	147	145	41	43	101	59	3.95
21	261	262	262	262	262	0	1	20	20	1.54
22	245	246	246	247	247	10	16	50	40	2.75
23	211	211	211	212	213	15	16	164	148	7.25
24	211	211	211	212	212	19	26	101	81	5.27
25	256	258	258	258	258	0	1	10	10	1.15
26	246	246	246	247	247	11	15	79	67	3.24
27	409	411	411	411	411	0	1	1	1	0.78
28	402	403	403	403	403	0	1	1	1	0.80
29	99	99	99	100	100	15	18	96	80	4.11
30	105	105	105	107	106	6	8	114	107	3.60
31	204	204	204	207	206	30	31	137	106	8.10
32	196	196	196	199	197	47	57	191	143	9.60
33	411	411	411	416	414	58	61	168	109	11.76
34	394	394	394	400	395	82	91	215	132	17.45
aver.	193.62	194.25	194.25	196.56	196.18	23.20	26.14	77.27	53.38	4.212

We now compare the benchmark version of the algorithm to different variants in order to test alternative initialisations, branching priority and the effect of the enhancement features. Table 3 contains comparative results: measures of computational effort, averaged over the 34 instances, for the different variants of the algorithm. Each version of the algorithm differs from the benchmark version by only one feature. The CPU times summarise the effect of such variations. Version A gives branching priority to the large items i's rather than to the large knapsack components l's. Version A requires 170% of the benchmark time. Version B does not compute heuristic solution at the outset and uses only the artificial column to initialise the master. Version B requires 165% of the benchmark time. Version C uses an "unit matrix" instead of an artificial column to ensure an initial feasible solution to the master LP at each node of the branch-and-bound tree. Version C requires 127% of the benchmark time. Version D uses an artificial column whose entries are equal to the right-hand-sides of the greater or equal to constraints.

Version D requires approximately the same time as the benchmark. However, having entries of different magnitude in the master formulation can cause difficulties in solving these LPs.

Table 3. Comparison of different versions of the algorithm

Version		depth	nod	mast	SP	time
A:	branch on big items	33.64	41.63	102.17	67.85	7.163
B:	no use of heuristics	29.77	34.16	222.21	191.53	6.972
C:	unit matrix initialisation	24.56	30.71	87.58	61.29	5.366
D:	art var with entry = to rhs	24.37	27.88	78.33	53.21	4.185
E:	**benchmark**	**23.20**	**26.14**	**77.27**	**53.38**	**4.212**
F:	a priori bound on v	24.44	28.13	81.88	56.67	4.538
G:	feasibility cuts	22.01	24.29	75.50	52.21	4.167
H:	L^2 cuts	22.84	25.20	82.15	55.33	5.307
I:	L^2 pruning	23.37	25.68	78.54	54.52	4.126
J:	early branching	26.06	32.44	82.92	55.94	4.666
K:	lower bound on waste	28.56	34.44	89.24	58.95	4.808
L:	enhanced algorithm	22.94	25.00	77.20	52.95	4.201
M:	rounding heuristic	0.42	1.35	83.51	61.33	3.224

The following versions of the algorithm differ from the benchmark by an extra feature. Version F uses the a priori bound (22) on the column generation subproblem. There is an unexpected increase in computational effort. Our interpretation is that the use of such bound keeps the column generation procedure from generating the last column before termination and this "last" column is probably a very "good" one, i.e. one that is likely to appear in an optimum solution. Version G uses the feasibility cuts (24) for items i with $q_i^{\max} \leq 3$. This partial implementation of feasibility cuts shows a slight decrease in the size of the branch-and-bound tree. Version H uses the L^2 cuts (26). The resulting decrease in the size of the branch-and-bound tree does not compensates for the extra iterations in the column generation procedure and the computational burden of the separation algorithm. Version I applies L^2 cuts only to check whether the current node can be pruned before it is processed. We can observe that this pruning yields a decrease in the number of branch-and-bound nodes actually processed. In Version J, the column generation procedure is terminated when the subproblem value is no greater than 1.001. This form of early branching results in an increase of the computational effort. The increase is even much worse when early branching is based on using a tolerance on the duality gap. Version K uses the waste lower bound cut (25), adding an artificial item of unit weight and demand equal to the minimum waste. This cut proved not helpful for cutting stock problems, but it can be for bin packing problems with small bin capacity W as those solved by Valerio de Carvalho (1996). Hence, we use this cut only when comparing our results with those of Valerio de Carvalho (1996).

We conclude from these comparative tests that the helpful features are the feasibility cuts and the L^2 pruning. We then use these two features together and refer to this version L as the *enhanced algorithm*. Version L yields a smaller tree and fewer column generation iterations. We therefore use these features within the rounding heuristic. In Version M, we test the enhanced algorithm with calls to the rounding heuristic at every branch-and-bound node. Version M yields a further 23% reduction in computation time.

4.2. Numerical results and benchmarking

We now present the results we obtained for the larger cutting stock and bin packing instances and we compare them to those of previous studies. There are 6 classes of randomly generated instances for the cutting stock problem (CS) and 11 for the bin packing problem (BP). Table 4 summarises their characteristics. For each class of problems, we have generated 20 instances. The results we present are averages over these 20 instances. Table 4 gives the average sizes n and n' of these instances after preprocessing. It also provides average values for the lower and upper bounds on the optimum integer solution value, IP. In all cases, $L^* = IP$. The L^2 bound is not as tight for instances with large range of item sizes. The Fill Bin heuristic $H2$ tends to dominate the standard FFD and BFD heuristics $H1$ for instances with fewer large items.

Table 4 also contains the computational times ($time$ = CPU time in seconds) required by our algorithm (Version M). In comparison, the columns $tiH1$ and $tiH2$ give average times required by the heuristic $H1$ and $H2$ respectively, showing that the pseudo-polynomial complexity of the $H2$ heuristics does not translate into large computational time in practice. Table 4 therefore summarises the situation with regards to the difficulty of cutting stock problems. It appears that the computational effort increases with the number of different item sizes (as opposed to the initial number of items) and not so much with the magnitude of the demands. But, more importantly, it depends on the distribution of item sizes. However, the classes of problems that are the most difficult for the branch-and-bound enumeration (i.e. problems with small item weights) are often well approximated by the Fill Bin heuristic.

In Tables 5 to 7, we consider each class of problems and compare the computational effort required by our algorithm to that of previous algorithms. The computational effort is assessed in terms of the depth of the branch-and-bound tree, the number of processed nodes, the number of times the master and the subproblem are solved, and the CPU time in seconds. Table 5 gives the results for the cutting stock problem. *EA* refers to the Enhanced version of our Algorithm (denoted version L in Table 3). *EA with RH* refers to the same algorithm with the additional use of the Rounding Heuristic at each node of the branch-and-bound tree (this version was denoted M in Table 3). *Vance* refers to the results obtained by Vance (1996). Her branch-and-price algorithm ran on an IBM RS6000/590 using CPLEX 3.0 as LP solver. The column $solved$ indicates the number of instances in the sample of 20 randomly generated instances that could be solved within the prescribed time limit (5 minutes for us and 10 minutes in Vance's study). The table entries are averages over the 20 instances, unless some of them could not be solved within the prescribed time limit, in which cases the average is over the solved instances.

The computational effort required by the branch-and-price algorithm increases when the range of item sizes and hence the number of knapsack component n' (given in Table 4) increase. However, this fact may not clearly appear in the results because the Fill Bin heuristic has found optimal solutions for 18 out of 20 instances in class 3 and for 17 instances in class 5. The performance of the rounding heuristic is remarkable. It brings robustness (solving every instances) and efficiency. The comparison with Vance's results is only illustrative as CPU times are not on the same machines and the statistics are influenced by the number of problems solved (if, for instance, we consider only the 18 best solved class 2 instances, the statistics for algorithm "*EA*" are depth = 106.1, nod =

Table 4. Problem size, quality of upper and lower bounds, and computational times for the enhanced algorithm with rounding heuristic (version M)

Problem type	n	n'	L^2	L^*	IP	H1	H2	time	tiH1	tiH2
CS, class 1: $w \sim U[1, 7500]$, $\overline{d} = 50$	49.74	110.00	982.40	992.35	992.35	998.24	1007.8	4.84	1.932	0.728
CS, class 2: $w \sim U[1, 5000]$, $\overline{d} = 50$	49.65	143.05	635.10	635.49	635.60	641.65	647.60	24.61	1.374	1.212
CS, class 3: $w \sim U[1, 2500]$, $\overline{d} = 50$	49.39	185.31	317.03	317.03	317.09	318.14	317.19	12.70	0.767	1.071
CS, class 4a: $w \sim U[500, 5000]$, $\overline{d} = 50$	49.70	128.19	690.80	690.89	690.89	703.21	710.47	21.95	1.482	1.491
CS, class 4b: $w \sim U[500, 5000]$, $\overline{d} = 100$	49.70	129.25	1380.5	1381.1	1381.1	1405.5	1420.0	19.19	4.484	1.491
CS, class 5: $w \sim U[500, 2500]$, $\overline{d} = 50$	49.40	166.09	372.66	372.66	372.75	377.69	372.90	19.10	0.885	1.597
BP: $w \sim U[1, 100]$, $W = 100$, 500 items	98.45	166.76	255.29	256.14	256.14	256.46	256.26	1.53	0.536	0.329
BP: $w \sim U[1, 100]$, $W = 120$, 500 items	98.45	180.54	209.46	209.49	209.49	209.69	209.64	1.39	0.376	0.258
BP: $w \sim U[1, 100]$, $W = 150$, 500 items	98.45	199.74	167.44	167.44	167.44	167.56	167.44	0.92	0.311	0.313
BP: $w \sim U[20, 100]$, $W = 100$, 500 items	80.79	116.30	318.65	319.09	319.09	319.18	319.29	1.20	0.550	0.277
BP: $w \sim U[20, 100]$, $W = 120$, 500 items	80.79	130.20	256.96	257.94	257.94	258.05	258.05	1.11	0.403	0.234
BP: $w \sim U[20, 100]$, $W = 150$, 500 items	80.79	151.05	201.20	201.20	201.20	203.90	201.81	2.24	0.321	0.315
BP: $w \sim U[50, 100]$, $W = 100$, 500 items	50.00	51.00	495.16	495.16	495.16	495.16	495.16	0.88	0.554	0.111
BP: $w \sim U[50, 100]$, $W = 120$, 500 items	50.00	61.00	401.45	401.74	401.74	401.74	401.74	0.74	0.391	0.106
BP: $w \sim U[50, 100]$, $W = 150$, 500 items	50.00	76.00	252.29	253.45	253.45	253.45	253.45	0.64	0.225	0.082
BP: $w \sim U[20, 100]$, $W = 150$, 1000 items	81.00	155.00	400.56	400.56	400.56	405.41	401.36	3.41	0.765	0.484
BP: "Triplets", $IP =$ nb of items /3 $w \in [250, 500]$, $W = 1000$, 249 items	140.10	199.14	83.00	83.00	83.00	95.00	84.15	44.13	0.443	1.441

121.3, mast = 255.1, and time = 37.5 seconds and, for algorithm "*EA with RH*", depth = 1.82, nod = 2.8, mast = 349.9, and time = 18.25).

We have not explicitly compared our approach to that of Scheithauer and Terno (1995) because we only became aware of their work after completing this study. More-

Table 5. Computational results for Cutting Stock Problems (with 50 items, average demands of 50 or 100, and bin capacity $W = 10\,000$) and comparison with Vance's results

Problem type	CS, class 1: $w \sim U[1, 7500]$					
algorithm	depth	nod	mast	SP	time	solved
EA	14.05	16.74	67.49	52.64	5.33	20
EA with RH	0.29	1.29	72.96	58.01	4.84	20
Vance	55	129	403		32.1	19
Problem type	CS, class 2: $w \sim U[1, 5000]$					
algorithm	depth	nod	mast	SP	time	solved
EA	108.31	140.84	306.31	179.58	46.24	19
EA with RH	3.29	4.29	418.84	309.96	24.61	20
Vance	121	147	549		27.0	18
Problem type	CS, class 3: $w \sim U[1, 2500]$					
algorithm	depth	nod	mast	SP	time	solved
EA	8.42	9.16	22.00	13.53	6.84	19
EA with RH	0.10	0.22	104.50	85.80	12.70	20
Vance	157	227	1950		307.7	13
Problem type	CS, class 4a: $w \sim U[500, 5000]$, $\overline{d} = 50$					
algorithm	depth	nod	mast	SP	time	solved
EA	76.90	84.29	223.80	142.11	21.95	20
EA with RH	1.15	2.15	228.50	178.10	11.11	20
Problem type	CS, class 4b: $w \sim U[500, 5000]$, $\overline{d} = 100$					
algorithm	depth	nod	mast	SP	time	solved
EA	98.49	101.49	238.89	137.25	24.90	20
EA with RH	1.34	2.41	297.39	227.60	19.19	20
Problem type	CS, class 5: $w \sim U[500, 2500]$					
algorithm	depth	nod	mast	SP	time	solved
EA	24.33	24.50	57.11	32.60	12.97	18
EA with RH	0.46	0.66	205.56	167.21	19.10	20

over, Scheithauer and Terno find optimal solutions for instances where the cost of the partial integer solution plus that of the residual problem does not exceed the master LP bound, but it is not clear how they proceed when this is not the case. The most difficult class of instances solved by Scheithauer and Terno involved a knapsack capacity of 5000 and 10 to 100 items. Their results may be compared with those we obtain for classes 2 and 4. (However, we consider a knapsack capacity of 10 000). They solve problems with $n \sim U[41, 50]$, $w \sim U[116, 2500]$, $W = 5000$, and $d \sim U[82, 500]$ in an average of 167 seconds on a PC 486 DX, 66 MHz. For instances with $n \sim U[51, 60]$, $w \sim U[96, 2500]$, $W = 5000$, and $d \sim U[102, 600]$, their average CPU time is 339 seconds.

Table 6 compares algorithms "*EA*" and "*EA with RH*" with that of Martello and Toth (1990), denoted "*Mart. & Toth*", and that of Vance et al. (1994), denoted "*Vance et al.*", for a standard test set of Bin Packing problems. Martello and Toth's times are on an HP 9000/840 and Vance et al.'s times on a IBM RS6000/550 using CPLEX 2.0 as LP solver. Table 6 shows that our algorithm performs well across the range of item size distribution, and that the rounding heuristic is again very helpful for the most difficult instances. In contrast, the approach of Martello and Toth and that of Vance et al. were not able to

Table6. Results for Bin Packing Problems and comparison with previous studies: Martello and Toth (1990) and Vance et al. (1994)

BP with 500 items		$W = 100$					$W = 120$					$W = 150$				
	algorithms	dep	nod	mast	SP	time	dep	nod	mast	SP	time	dep	nod	mast	SP	time
$w \in [1, 100]$	EA	0.49	1.35	13.49	12.90	1.52	1.50	1.63	12.79	11.06	1.39	0.00	0.00	0.00	0.00	0.91
	EA with RH	0.15	0.94	14.40	13.20	1.53	0.00	0.16	16.30	12.90	1.39	0.00	0.00	0.00	0.00	0.92
	Mart. & Toth		2805			10.23		887			10.34		28			2.124
	Vance et al.		25.4	1425		4741				not	solved				not	solved
	algorithms	dep	nod	mast	SP	time	dep	nod	mast	SP	time	dep	nod	mast	SP	time
$w \in [20, 100]$	EA	0.85	1.34	6.55	5.50	1.24	0.56	1.29	11.20	10.51	1.17	19.2	42.64	105.00	73.40	10.87
	EA with RH	0.00	0.34	5.69	4.30	1.20	0.00	0.63	10.20	9.17	1.11	0.16	0.60	50.71	34.00	2.24
	Mart. & Toth		28			0.127		1663			10.06				not	solved
	Vance et al.		8.0	642.9		251.7		36.2	968.5		883.8		307.2	2344.8		3307.6
	algorithms	dep	nod	mast	SP	time	dep	nod	mast	SP	time	dep	nod	mast	SP	time
$w \in [50, 100]$	EA	0.00	0.00	0.00	0.00	0.89	0.00	0.26	0.80	0.80	0.74	0.00	0.71	5.60	5.60	0.64
	EA with RH	0.00	0.00	0.00	0.00	0.88	0.00	0.26	0.80	0.80	0.74	0.00	0.71	5.60	5.60	0.64
	Mart. & Toth		0			0.049		0			0.050		0			0.051
	Vance et al.		1.0	8.5		0.3		1.0	195.1		14.4		1.0	452.6		50.3

solved problems for some item size distributions (the ones marked "not solved" in the Table). The difficulties experienced by Martello and Toth and Vance et al. are partly due to the fact that they have not grouped items of the same size under a single item as has been done in Valerio de Carvalho's study and ours. Then, their solution space exhibits some symmetry, making the branching scheme less efficient.

Table 7 gives results for the bin packing instances with $w \in [20, 100]$, $W = 150$, and 500 or 1000 items, as well as for the so-called "triplets" instances from the OR-library,

Table 7. Results for Bin Packing instances when using the waste lower bound and results for "triplets" instances, plus comparison with Valerio de Carvalho's results

Problem type	$w \sim U[20, 100]$, $W = 150$, 500 items				
algorithm	depth	nod	mast	SP	time
EA	18.75	19.4	69.56	49.5	4.15
EA with RH	0.16	0.60	50.71	34.0	2.24
VdC		50.0		46.3	32.51
Problem type	$w \sim U[20, 100]$, $W = 150$, 1000 items				
algorithm	depth	nod	mast	SP	time
EA	38.25	39.2	106.44	67.0	7.20
EA with RH	0.21	0.85	83.15	56.0	3.94
VdC		59.2		37.7	35.11
Problem type	BP: triplets with 249 items				
algorithm	depth	nod	mast	SP	time
EA	46.55	48.9	521.99	474.2	137.80
EA with RH	0.54	1.5	425.7	374.7	44.13
VdC		58.3		439.4	649.04

and compare them with the results obtained by Valerio de Carvalho (1996), denoted by "*VdC*". Valerio de Carvalho does not report results for instances with $w \in [1, 100]$ or $w \in [50, 100]$. Here EA includes the use of the cut on the minimum amount of unused capacity (i.e. the lower bound on the waste) as in Valerio de Carvalho's algorithm and the comparison with Table 6 for instances with $w \in [20, 100]$, $W = 150$, and 500 items shows that this cut yields an improvement. Valerio de Carvalho's reported computation times are on a 40 MHz PC 486 DX, with 4 Mbytes of RAM. Therefore CPU times cannot be compared. However, judging from the counters, it appears that the two approaches are comparable for these classes of instances (our branching scheme typically yields fewer branch-and-bound nodes).

Valerio de Carvalho's approach and ours also differ by the way in which the subproblem is handled. Valerio de Carvalho solves the knapsack subproblems by dynamic programming (or equivalently as shortest path problems in an appropriate network). His approach could be more sensitive to an increase of the bin capacity. On the other hand, the variable redefinition approach used by Valerio de Carvalho permits the introduction of several forms of branching constraints or cuts without having to modify the subproblem. Finally, the current implementation of our algorithm is probably better suited for cutting stock problems than it is for bin packing problems because bin packing problems require the use of branching on pairs of items and we have not implemented a proper procedure to deal with the resulting modified subproblem. It would be interesting to see how Valerio de Carvalho's approach and ours compare on the cutting stock instances solved here which have wider range of item sizes and larger bin capacity.

5. Final remark

We have seen that the efficient solution of an integer program such as the cutting stock problem requires the combined use of tight lower bounds, efficient branching scheme, appropriate tree search and proper initialisation. Additional features, such as

good heuristics, cutting planes, early branching or rounding procedures can make a significant contribution as well. However, there is a tradeoff between their potential benefit and the extra computational burden they cause. Both the benefits and the associated computational burden depend on the problem on hand and the balance between the two will differ for other applications. For instance, in applications where the column generation subproblem is strongly NP-hard, the contribution of the a priori bound (22) is essential (Sutter, Vanderbeck and Wolsey, 1994). In applications where the master LP solution is not a good approximation of the IP solution, it is better not to include the columns of the heuristic solution in the master (Vanderbeck, 1994).

Acknowledgements. We thank Constantine Goulimis and Pamela Vance for the real-life cutting stock instances they have given to us. We also express our gratitude to Constantine Goulimis and Stefan Scholtes for their helpful comments on the draft of this paper.

References

1. Barnhart, C., Johnson, E.L., Nemhauser, G.L., Savelsbergh, M.W.P., Vance, P.H. (1994): Branch-and-Price: Column Generation for Solving Huge Integer Programs. In: Birge, J.R., Murty, K.G., eds., Mathematical Programming, State of the Art. Book of the International Symposium on Mathematical Programming
2. Beasley, J.E. (1990): OR-Library: Distributing test problems by electronic mail. J. Oper. Res. Soc. **41**, 1069–1072
3. CPLEX (1994): Using the CPLEX Linear Optimizer, version 3.0. CPLEX Optimization, Inc., Suite 279, 930 Tahoe Blvd., Bldg. 802, Incline Village, NV 89451-9436. (702) 831-7744
4. Degraeve, Z., Schrage, L. (1997): Optimal Integer Solutions to Industrial Cutting Stock Problems. To appear in INFORMS J. Comput.
5. Desaulniers, G., Desrosiers, J., Ioachim, I., Solomon, M.M., Soumis, F. (1994): A Unified Framework for Deterministic Time Constrained Vehicle Routing and Crew Scheduling Problems. Les Cahiers du GERAD, G-94-46
6. Desrosiers, J., Dumas, Y., Solomon, M.M., Soumis, F. (1994): Time Constrained Routing and Scheduling, In: Ball, M.E., Magnanti, T.L., Monma, C., Nemhauser, G.L., eds., Handbooks in Operations Research and Management Sciences: Networks, Chapter 2. North-Holland
7. Desrosiers, J., Soumis, F. (1996): Cuts for the Vehicle Routing Problem. Private communication
8. Eppen, G.D., Martin, R.K. (1987): Solving Multi-Item Capacitated Lot-Sizing Problems Using Variable Redefinition. Oper. Res. **35**, 832–848
9. Farley, A.A. (1990): A note on bounding a class of linear programming problems, including cutting stock problems. Oper. Res. **38**, 922–923
10. Gilmore, P.C., Gomory, R.E. (1961): A Linear Programming Approach to the Cutting Stock Problem. Oper. Res. **9**, 849–859
11. Gilmore, P.C., Gomory, R.E. (1963): A Linear Programming Approach to the Cutting Stock Problem: Part II. Oper. Res. **11**, 863–888
12. Goulimis, C.N. (1988): The cutting stock problem revisited. Ph.D. thesis, Dept. of Electrical Engineering, Imperial College, London SW7
13. Goulimis, C.N. (1990): Optimal Solutions for the Cutting Stock Problem. Eur. J. Oper. Res. **44**, 197–208
14. Johnson, E.L., Padberg, M.W. (1981): A Note on the Knapsack Problem With Special Ordered Sets. Oper. Res. Lett. **1**, 18–22
15. Marcotte, O. (1985): The Cutting Stock Problem and Integer Rounding. Math. Program. **33**, 89–92
16. Marcotte, O. (1986): An Instance of the Cutting Stock Problem for which the Rounding Property Does Not Hold. Oper. Res. Lett. **4**, 239–243
17. Martello, S., Toth, P. (1990): Knapsack Problems: Algorithms and computer Implementations. Wiley-Interscience Series in Discrete Mathematics and Optimization
18. Nemhauser, G.L., Wolsey, L.A. (1988): Integer and Combinatorial Optimization. John Wiley & Sons
19. Scheithauer, G., Terno, J. (1995): A Branch-and-Bound Algorithm for Solving One-dimensional Cutting Stock Problems Exactly. Appl. Math. **23**, 151–167
20. Scholl, A., Klein, R., Jurgens, C. (1997): Bison: a fast hybrid procedure for exactly solving the one-dimensional bin packing problem. Comput. Oper. Res. **24**, 667–645

21. Sutter, A., Vanderbeck, F., Wolsey, L.A. (1998): Optimal Placement of Add/Drop Multiplexers: Heuristic and Exact Algorithms. Oper. Res. **46**(5), 719–728
22. Valerio de Carvalho, J.M. (1996): Exact Solution of bin-packing problems using column generation and branch-and-bound. Working paper. Depart. Prouducao e Sistemas, Universidade do Minho, 4709 Braga Codex, Portugal
23. Vance, P.H., Banhard, C., Johnson, E.L., Nemhauser, G.L. (1994): Solving Binary Cutting Stock Problems by Column Generation and Branch-and-bound. Comput. Optim. Appl. **3**, 111–130
24. 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. To appear in Comput. Optim. Appl.
25. Vanderbeck, F. (1994): Decomposition and Column Generation for Integer Programs. Ph.D. Thesis, Faculté des Sciences Appliqées, Université Catholique de Louvain, Louvain-la-Neuve
26. Vanderbeck, F., Wolsey, L.A. (1996): An Exact Algorithm for IP Column Generation. Oper. Res. Lett. **19**, 151–159
27. Vanderbeck, F. (1996): On Dantzig-Wolfe decomposition in integer programming and ways to perform branching in a branch-and-price algorithm. Research Papers in Management Studies **29**, University of Cambridge, 1994-1995 (revised in 1996). To appear in Oper. Res.
28. Vanderbeck, F. (1998): Extending Dantzig's Bound to the Bounded Multiple-class Knapsack Problem. Research Papers in Management Studies **14**, University of Cambridge