A Branch-and-Cut-and-Price Algorithm for One-Dimensional Stock Cutting and Two-Dimensional Two-Stage Cutting

G. Belov *,[1] G. Scheithauer

University of Dresden, Institute of Numerical Mathematics, Mommsenstr. 13, 01062 Dresden, Germany

Abstract

The one-dimensional cutting stock problem (**1D-CSP**) and the two-dimensional two-stage guillotine constrained cutting problem (**2D-2CP**) are considered in this paper. The Gilmore-Gomory models of these problems have very strong continuous relaxations providing a good bound in an LP-based solution approach. In recent years, there have been several efforts to attack the one-dimensional problem by LP-based branch-and-bound with column generation (called branch-and-price) and by general-purpose Chvátal-Gomory cutting planes. In this paper we investigate a combination of both approaches, i.e., the LP relaxation at each branch-and-price node is strengthened by Chvátal-Gomory and Gomory mixed-integer cuts. The branching rule is that of branching on variables of the Gilmore-Gomory formulation. Tests show that, for 1D-CSP, general-purpose cuts are useful only in exceptional cases. However, for 2D-2CP their combination with branching is more effective than either approach alone and mostly better than other methods from the literature.

Key words: cutting, cutting planes, column generation, branch-and-bound

1 Introduction

Branch-and-cut algorithms [1–3] are branch-and-bound algorithms that are based on the LP relaxation of the model and are enhanced by the following feature: cutting planes tighten the LP relaxation in branch-and-bound nodes. Such algorithms are nowadays the most effective schemes for most integer and

* Corresponding author.
URL: `www.math.tu-dresden.de/~capad` (G. Belov).

[1] Research was supported by the Daimler-Benz Foundation.

mixed-integer programming problems [2] and have been implemented in many mixed-integer optimizers such as ILOG CPLEX [4], XPRESS-MP, MINTO, etc. However, the effectiveness, especially the strength of the LP relaxation, strongly depend on the chosen model of the problem. In some models, the number of variables is not polynomially bounded. In such models it may be advantageous to work with a restricted formulation called *restricted master problem* and to generate new variables as they become necessary. This approach is called *column generation*. Typically, column generation is performed by solving the so-called *pricing problem* or *slave optimization problem* where, e.g., the reduced cost of the new column should be maximized or minimized. If column generation is integrated into branch-and-cut so that new variables may be added to the model in any branch-and-bound node, we speak of *branch-and-cut-and-price* algorithms [5]. The purpose of the current work is to investigate a branch-and-cut-and-price algorithm for 1D-CSP and 2D-2CP.

For 1D-CSP, the first model with column generation was proposed by Gilmore and Gomory [6]. Each variable denotes how many times a specific cutting pattern is contained in the solution. Vance et al [7] applied *branch-and-price*, i.e., branch-and-bound with column generation, to a special kind of 1D-CSP, the bin-packing problem. The authors discussed *branching rules*, i.e., ways how to split the subproblems during branch-and-bound. They argued that the splitting should be balanced, i.e., the new subproblems should be nearly equal in size. The authors proposed to branch on the number of patterns containing two certain piece types together.

Moreover, branching constraints can change the structure of the pricing problem. Vanderbeck [8] proposed several new branching rules. Valério de Carvalho (cf. [9]) introduced an *arc flow formulation* of 1D-CSP and a branching rule based on the variables of this formulation. The changes in the pricing problem were only negligible. Kupke [10] and Degraeve and Peeters [11,12] investigated branch-and-price with branching on variables of the Gilmore-Gomory formulation. In [12] we find that the results are comparable to the scheme of Vanderbeck. Unfortunately, a substantial distance in computational power and algorithmic features has not allowed a unified comparison of the schemes. In the current work we apply branching on variables of the Gilmore-Gomory formulation.

Cutting planes are commonly used to get a tight bound at a node of the branch-and-bound tree and aim for reducing the total number of nodes [13]. In the case of 1D-CSP, the optimum LP relaxation value of the Gilmore-Gomory formulation rounded up at the root node often equals the optimum solution. As a result, the bound is very tight, even without cuts. In 2D-2CP, the objective function has its values among integer combinations of item prices, which makes the optimality test using a lower bound much more difficult. We shall see that for 2D-2CP a combination of Chvátal-Gomory cuts and branching is better

than either a pure cutting plane algorithm or branch-and-price.

Note that many branch-and-price implementations employ simple cuts which do not provide finite description of the convex hull of integer solutions as is the case for CHVÁTAL-GOMORY cuts. For example, a special case of GOMORY mixed-integer cuts, where only a single previous constraint is used for cut generation, are applied in [14]. This corresponds to widely-used practices and general beliefs expressed in the literature. Usually, in the context of an LP-based branch-and-bound algorithm, where the LP relaxation is solved using column generation, cutting planes are carefully selected in order to avoid the destruction of the structure of the pricing problem. This viewpoint is shared by numerous authors. Barnhart, Hane, and Vance [15], for example, mention as one of the main contributions of their branch-and-price-and-cut algorithm "*a pricing problem that does not change even as cuts are added, and similarly, a separation algorithm that does not change even as columns are added.*" Vanderbeck [8] writes in the introduction of his computational study of a column generation algorithm for 1D-BPP and 1D-CSP: "*Adding cutting planes can be viewed in the same way [as branching]. However, the scheme efficiency depends on how easily one can recognize 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.*"

Our algorithm can be seen as an investigation 'against the mainstream'. The LP relaxation in the nodes is strengthened by CHVÁTAL-GOMORY and GOMORY mixed-integer cuts. A standalone cutting plane algorithm for 1D-CSP was investigated in [16,17]. When such cutting planes are added to the LP relaxation, the pricing problem becomes very complex and often cannot be solved optimally in an acceptable time. Moreover, the modeling and implementation of its solution method as well as of the cutting plane apparatus for the chosen kind of cuts require much effort. Here we develop a new upper bound for this pricing problem. We develop local cuts (valid only in a subproblem of the solution tree). For 2D-2CP we propose a pricing procedure which enables the search for strips of different widths without repetitions. We are not aware of any published results on branch-and-price algorithms for 2D-2CP.

Below we introduce the problems considered and their Gilmore-Gomory models. In Section 3 we present the main features of our branch-and-cut-and-price scheme. In Section 4 we discuss cutting planes and in Section 5 column generation. Computational results are presented in Section 6.

2 Gilmore-Gomory Models

2.1 One-Dimensional Stock Cutting

The one-dimensional cutting stock problem (1D-CSP) is defined by the following data: (m, L, $l = (l_1, \ldots, l_m)$, $b = (b_1, \ldots, b_m)$), where L denotes the length of each stock piece, m denotes the number of smaller piece types and for each type $i = 1, \ldots, m$, l_i is the piece length, and b_i is the order demand. In a *cutting plan* we must obtain the required set of pieces from the available stock lengths. The objective is to minimize the number of used stock lengths. In a real-life cutting process there are some further criteria, e.g., the number of different cutting patterns (setups), open stacks, due dates [18,19].

A special case in which the set of small objects is such that only one item of each product type is ordered, i.e., $b_i = 1\ \forall i$ (sometimes also when b_i are very small), is known as the *bin-packing problem* (**1D-BPP**). This special case, having a smaller overall number of items, is more suitable for pure combinatorial (non-LP-based) solution approaches [20,21].

In 1961, Gilmore and Gomory [6] proposed the following model of 1D-CSP. A *cutting pattern* describes how many items of each type are cut from a stock length. Let column vectors $a^j = (a_{1j}, \ldots, a_{mj})' \in \mathbb{Z}_+^m$, $j = 1, \ldots, n$, represent all possible cutting patterns. To be a valid cutting pattern, a^j must satisfy

$$\sum_{i=1}^{m} l_i a_{ij} \leq L \tag{1}$$

(knapsack condition). Moreover, we consider only *proper* patterns:

$$a_{ij} \leq b_i, \quad i = 1, \ldots, m, \quad j = 1, \ldots, n \tag{2}$$

because this reduces the search space of the continuous relaxation in instances where the demands b are small [22].

Let x_j, $j = 1, \ldots, n$, be the *frequencies*, also called *intensities*, *multiplicities*, *activities*, *run lengths*, i.e., the numbers of application of the patterns in the solution. The model of Gilmore and Gomory is as follows:

$$z^{\text{1D-CSP}} = \min \sum_{j=1}^{n} x_j \tag{3}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} a_{ij} x_j \geq b_i, \quad i = 1, \ldots, m \tag{4}$$

$$x_j \in \mathbb{Z}_+, \quad j = 1, \ldots, n. \tag{5}$$

The huge number of columns is not available explicitly for practical problems. Usually, necessary patterns are generated during a solution process (*column generation*, *pricing*). However, the number of different patterns in a solution

cannot be greater than the number of physical stock pieces used and is usually comparable with the number of product types.

The model has a very strong relaxation. There exists the conjecture *Modified Integer Round-Up Property* (**MIRUP**, [23]), stating that the gap between the optimum value and the LP relaxation value of the Gilmore-Gomory model is always smaller than 2. Actually, there is no instance known with a gap greater than 6/5 [24]. Moreover, instances with a gap smaller than 1 constitute the vast majority. These are called *Integer Round-Up Property* (**IRUP**) instances. Instances with a gap greater than or equal to 1 are called **non-IRUP** instances.

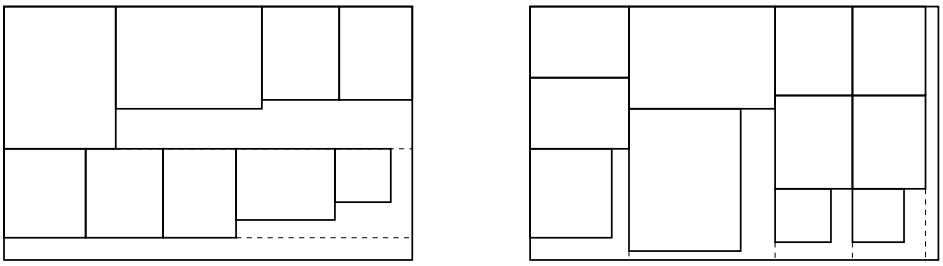
The model has a huge number of variables. But no variable values can be exchanged without changing the solution, i.e., the model has no symmetry. This and the strength of the relaxation make it advantageous for an enumerative approach. In the survey [9] and in [19] we find several alternative ILP models of 1D-CSP.

2.2 Two-Dimensional Two-Stage Constrained Cutting

The two-dimensional two-stage constrained cutting problem (2D-2CP) [25–28] consists of placing a subset of a given set of smaller rectangular pieces on a single rectangular plate. Note that our choice of the problem name follows the latest publication [28]. The total profit of the chosen pieces should be maximized. *n-stage* cutting means that the current plate is cut *guillotine-wise* from edge to edge in each stage and the number of stacked stages is limited to n. In two-stage cutting, strips (*strip patterns*) produced in the first stage are cut into pieces in the second stage. *Constrained* means that the pieces can be distinguished by types and the number of produced items of each type is limited. We consider the *non-exact* case, in which items do not have to fill the strip width completely, as illustrated in Figure 1. Pieces have a fixed orientation, i.e., rotation by 90° is not considered.

The problem 2D-2CP is defined by the following data: L, W, m, (l_i, w_i, p_i, b_i), $i = 1, \ldots, m$, where L and W denote the length and width of the stock plate, m denotes the number of piece types and for each type $i = 1, \ldots, m$, l_i and w_i are the piece dimensions, p_i is the value, and b_i is the upper bound on the quantity of items of type i. A *two-stage pattern* is obtained by cutting strips (1st stage), by default in the L-direction, then cutting single items from the strips (2nd stage). The task is to maximize the total value of pieces obtained. If piece prices are proportional to the areas, a trim loss minimization problem arises.

The Gilmore-Gomory formulation for 2D-2CP is similar to that for 1D-CSP.



a) The first cut is horizontal b) The first cut is vertical

Fig. 1. Two-stage patterns

Let column vectors $a^j = (a_{1j}, \ldots, a_{mj})' \in \mathbb{Z}_+^m$, $j = 1, \ldots, n$, represent all possible strip patterns, i.e., a^j satisfies $\sum_{i=1}^m l_i a_{ij} \leq L$ and $a_{ij} \leq b_i$, $i = 1, \ldots, m$ (*proper* strip patterns). Let x_j, $j = 1, \ldots, n$, be the intensities of the patterns in the solution. The model is as follows:

$$z^{\text{2D-2CP}} = \min \textstyle\sum_{j=1}^n \left(\sum_{i=1}^m -p_i a_{ij}\right) x_j \tag{6}$$

$$\text{s.t.} \quad \textstyle\sum_{j=1}^n a_{ij} x_j \leq b_i, \quad i = 1, \ldots, m \tag{7}$$

$$\textstyle\sum_{j=1}^n w(a^j) x_j \leq W \tag{8}$$

$$x_j \in \mathbb{Z}_+, \quad j = 1, \ldots, n, \tag{9}$$

where $w(a^j) = \max_{i:\, a_{ij}>0}\{w_i\}$ is the strip width. We represent the objective as a minimization function in order to keep analogy with 1D-CSP. This model can be viewed as representing a multidimensional knapsack problem.

Lodi and Monaci [27] propose two Integer Linear Programming formulations of 2D-2CP with a polynomial number of variables, so that no column generation is needed. Some new models interpret strip pattern widths as variables [29]. Such 'assignment' formulations usually provide a weaker continuous relaxation than the Gilmore-Gomory model [27].

3 Overview of the Algorithm

In this section, we discuss the issues which can be found in basically every specialization of branch-and-bound and are adapted in our model. These are: a lower bound and an optimality criterion which uses this bound, an upper bound, i.e., a rounding procedure, branching strategy, and preprocessing. Some more features are presented in [30]. For a general scheme of branch-and-cut, the reader is referred to [31,1,2] and for branch-and-cut-and-price to [5].

Consider the general form of the models (3)–(5), (6)–(9) after introducing slacks:

$$z^{IP} = \min\left\{cx : \; Ax = b,\; x \in \mathbb{Z}_+^n\right\}, \tag{10}$$

where $A \in \mathbb{Z}^{m\times n}$, $c \in \mathbb{Z}^n$, $b \in \mathbb{Z}^m$. The continuous relaxation of (10), the *LP master problem*, can be obtained by discarding the integrality constraints on the variables:

$$z^{LP} = \min\{cx : \; Ax = b,\; x \in \mathbb{R}_+^n\}. \tag{11}$$

z^{LP} is a lower bound on z^{IP}. The value of z^{LP} is used by the optimality criterion. For 1D-CSP, because of the MIRUP conjecture, it is extremely effective. For 2D-2CP the possible values of z^{IP} belong to the set of integer combinations of item prices. This makes the optimality test more difficult because usually there are many such combinations between z^{LP} and z^{IP}.

The continuous relaxation is solved by column generation and is strengthened by branching and cutting planes. We work with a subset of columns called a *variable pool* or *restricted master problem*. It is initialized with an easily constructible start basis. After solving this restricted formulation with the primal simplex method, we look for new columns. Let A_B be the basis matrix, and let $d = c_B A_B^{-1}$ be the vector of simplex multipliers. The *column generation problem* (*pricing problem*, *slave problem*) arises:

$$z^{\text{CG}} = \min\left\{c_j - da^j : \; j = 1,\ldots,n\right\}. \tag{12}$$

If the *minimum reduced cost* z^{CG} is zero then the continuous solution is optimum. Otherwise, a column with a negative reduced cost is a candidate to improve the current restricted formulation. For 1D-CSP, problem (12) is a standard knapsack problem with upper bounds (or a longest path problem in the arc flow formulation):

$$z^{\text{1D-CG}} = 1 - \max\left\{da : \; la \le L,\; a \le b,\; a \in \mathbb{Z}_+^m\right\}. \tag{13}$$

Dual simplex method is applied to the restricted master LP after adding cuts or branching constraints because dual feasibility of the basis is preserved. Primal simplex method is applied after adding columns and deleting cuts. To control the size of LPs and the complexity of column generation, we restrict the maximum number of added cuts and remove cuts which have not been tight for a large number of iterations at the current node. However, all cuts constructed at a node are stored and in the separation procedure we look for violated cuts among those of the current node and its parents. If no violated cuts are found, new cuts are constructed. To prevent cycling when the same deleted cuts are violated and added again iteratively, we count the number

of deletions of a cut at the current node and, after each deletion, we increase the minimum number of iterations that the cut should be kept the next time. Section 4 gives details on the usage of cutting planes and Section 5 describes the adaptation of the column generation procedure to the case with cuts.

3.2 Rounding of LP Solutions

Sometimes the solution of the relaxation is integer which means that the current node is solved. Otherwise, in order to find good solutions, we thoroughly investigate the neighborhood of the LP optimum by an extensive rounding procedure. As in [17], a rounded part $[x]$ of an LP solution is obtained by rounding some components up or down. Then, $b' = b - A[x]$ forms the demand vector of the *residual problem* which is solved by the heuristic *sequential value correction* (SVC). For 1D-CSP, SVC is thoroughly investigated in [18]. Moreover, the variation of $[x]$ in [17] has reduced by a factor of 10 the number of 1D-CSP instances which could not be solved optimally after the root LP relaxation.

Another way to obtain feasible solutions is to declare the variables integer and then use the mixed-integer solver of CPLEX on the currently known set of columns. The rounding procedure is called at every 30th node and the MIP solver every 200 nodes with a time limit of 30 seconds.

For residual problems in 2D-2CP we apply the following modification of SVC. For a general scheme of SVC, we refer the reader to [17,18]. Let us define a *pseudo-value* y_i of piece i so that it reflects the average material consumption in a strip. After constructing strip a containing a_i items of type i, the *current pseudo-value* y'_i is

$$y'_i = p_i \frac{w(a)}{w_i} \frac{L}{L-h},$$

where $h = L - la$ is the free length of the strip. The final pseudo-value is a weighted average of y'_i and the previous value, similar to the one-dimensional case.

After filling the plate, for those items which are not in the solution, we also distribute the space not occupied by the strips, i.e., the area $L(W - \sum_j w(a^j)x_j)$. It is divided among the piece types, being added to their pseudo-values in proportion to their free amounts b'_i. This is done in order to facilitate some exchange of items packed. The heuristic was not tested intensively because in 2D-2CP good solutions are found quickly. The nature of SVC is to minimize trim loss. Value maximization, specific to 2D-2CP, could be only partially considered.

3.3 Branching Strategy

In a branch-and-bound scheme there are two important questions: how to split a problem into subproblems and which node is to be processed next. As mentioned in the Introduction, we apply branching on single variables. If variable j has a fractional value $\overline{x}_j$ in the local optimum LP solution, the one subproblem is obtained by adding the constraint $x_j \leq \lfloor \overline{x}_j \rfloor$ *(left son)* and the other by adding the constraint $x_j \geq \lceil \overline{x}_j \rceil$ *(right son)*.

To select a branching variable, we apply the *most infeasibility* rule. This rule chooses a variable with the fractional part closest to 0.5. The hope is for the greatest impact on the LP relaxation. We also tested *pseudo-costs* [1] which showed good results for some non-IRUP instances. However, we do not present these results in the current work.

For node selection, the following strategies are tested.

(1) *Best first search (bfs).* A node is chosen with the weakest parent lower bound (unrounded LP value), promising the *best* solution. The goal is to improve the global lower bound, which is the minimum of local bounds of all unprocessed leaves. However, if this fails to occur early in the solution process, the search tree grows considerably.
(2) *Depth first search (dfs).* This rule chooses one of the deepest nodes in the tree. The advantages of dfs are the small size of the search tree and fast reoptimization of the subproblem. Also, good feasible solutions are found very quickly. The main disadvantage is that the global lower bound stays untouched for a long time resulting in bad solution guarantees, which leads to a long optimality proof. Another drawback is that if a wrong branch is chosen at a high level then much time can be spent inside. In [1] we read that pure dfs is not good for large problems. For IRUP 1D-CSP instances, however, the correct LP bound is known from the beginning, so pure dfs was quite efficient.
(3) In order to avoid investigating wrong branches, we propose a combination of bfs and dfs which can be called *diving from a bfs node*. Some number of nodes, say, 10, are chosen and processed according to bfs. The last bfs node is taken and its children are followed downwards for, say, m nodes, i.e., its subtree is investigated in a depth-first fashion. This diving is similar to the rounding procedure from [14].

Note that in 1D-CSP, for IRUP instances, the choice of the strategy can only change the moment when an optimum is found. For non-IRUP instances, it has no effect because we have to process all nodes with the lower bound smaller than z^{IP}. In [11,10,32] the node selection strategy was always depth-first search. We prefer diving because this allows us to switch between different

subtrees. For 2D-2CP, we found best-first search much more successful than dfs. We always prefer right nodes to be processed first.

3.4 Node Preprocessing

Node preprocessing is a name for LP/IP constraint propagation rules, allowing us to tighten variable bounds, eliminate variables and constraints, or determine infeasibility before solving [2]. Most mixed-integer solvers apply such rules in every node of the solution tree. Here are the features we used.

In 2D-2CP, the LP bound can be tightened by finding the *effective knapsack width* $W^{\text{eff}}(\leq W)$ which is the largest integer combination of piece widths not greater than W. Further, suppose the right-hand side is reduced because some columns are bounded from below at the current node, namely $b' = b - \sum_{j=1}^{n} a^j \mathcal{L}_j$ where $\mathcal{L}_j$ is the lower bound of column j. In 2D-2CP we also get $W' = W - \sum_{j=1}^{n} w(a^j)\mathcal{L}_j$. Then for each column we set the upper bound $\mathcal{U}_j = \min_{i:\, a_{ij}>0}\{\lfloor b'_i/a_{ij} \rfloor\}$, which can be already done at the root node (*integer bounding*). Also, in column generation we should take care not to generate columns in which the number of items of type i is greater than b'_i.

4 Cutting Planes

For the Gilmore-Gomory models of the problems under investigation, no problem-specific facet-defining cutting planes seem to be available. Thus, *general-purpose* cuts have to be used, e.g., Gomory fractional and mixed-integer cuts. They are known to provide a finite description of the convex hull of feasible solutions. Here we repeat some necessary information about the chosen kind of cuts (see, e.g., [33]) and, based on earlier works [16], describe the changes made in the current algorithm.

In Subsection 4.1 we describe how we use superadditive cuts which are based on linear combinations of current constraints. An interesting property is that the slack variables of current cuts have coefficients in the derived cuts. Then we propose an approach to use strengthened Chvátal-Gomory cuts in a numerically stable way. Mixed-integer cuts may have non-integer slacks; this has to be considered in the derived cuts.

In Subsection 4.2 local cuts are developed, i.e., cuts that consider branching constraints (upper bounds on the variables). These cuts are valid in the subproblem where they are created and in its children.

We define a cut as an inequality

$$\sum_{j=1}^{n} F(ua^j)x_j \leq F(ub), \tag{14}$$

where u is some vector producing a linear combination of the existing constraints and $F : \mathbb{R}^1 \to \mathbb{R}^1$ is some *superadditive non-decreasing function* [33]. If the solution is not optimum, further inequalities are constructed recursively based on linear combinations of original constraints and already added cuts. Suppose there are μ cuts in the LP. The dependence of the coefficients of cut r on the original constraints and previous cuts is shown by the following recursion:

$$\pi_j^r = \pi^r(a^j) = F(\sum_{i=1}^{m} u_i^r a_{ij} + \sum_{t=1}^{r-1} u_{m+t}^r \pi^t(a^j)), \quad r = 1, \ldots, \mu, \tag{15}$$

where u_i^r, $i = 1, \ldots, m+r-1$, are the coefficients of the linear combination of all previous constraints. Moreover, the following issue has to be kept in mind: when adding a cut, the constraint matrix is extended by a slack column, $\sum_{j=1}^{n} F(ua^j)x_j + s = F(ub)$. Further cuts which depend on this cut have their own coefficients in this column. Thus, cut r is represented as

$$\sum_{j=1}^{n} \pi_j^r x_j + \sum_{v=1}^{r-1} \pi^r(s_v)s_v + s_r = \pi_0^r. \tag{16}$$

To be precise, suppose for simplicity that cut slack variables are integers. Set $\pi^v(s_v) = 1$, $v = 1, \ldots, \mu$. Then coefficient $\pi^r(s_v)$ of cut r for the slack variable of cut v is $\pi^r(s_v) = F(\sum_{t=1}^{r-1} u_{m+t}^r \pi^t(s_v))$, in analogy to (15).

In [33] it is proposed to eliminate cut coefficients for slacks in the following way: just substitute $s_v = \pi_0^v - \sum_{j=1}^{n} \pi_j^v x_j$ in each cut dependent on cut v. Then in (16) we would have zero coefficients for s_v, $v < r$. An advantage would be that we could keep only active cuts in the LP formulation and remove those cuts on whose slacks the active cuts depend. But experience shows that the resulting constraints have worse numerical properties: the more cuts added, the greater the maximum absolute values of the coefficients. We add the cuts directly as in (16), which has always had good numerical properties: even with hundreds of cuts added, the absolute values of the coefficients were mostly below 1000.

CHVÁTAL-GOMORY valid inequalities for S [33] which were applied to 1D-CSP already in [16] are constructed using the superadditive function $F(ua) = \lfloor ua \rfloor$ (rounding down). They do have the convenience of integer slack variables. Combining inequality $\lfloor uA \rfloor x \leq \lfloor ub \rfloor$ with $uAx = ub$ yields a GOMORY fractional cut $\sum_{j=1}^{n} \mathrm{fr}(ua^j)x_j \geq \mathrm{fr}(ub)$, where $\mathrm{fr}(d) = d - \lfloor d \rfloor$. It is very effective to strengthen CHVÁTAL-GOMORY cuts in the following way. If $\mathrm{fr}(ub) = ub - \lfloor ub \rfloor < 1/k$, k integer, then multiplying u by k produces a

stronger or equivalent cut. Some other methods are possible [34]. After multiplication of u by an integer, it is advantageous to prevent numerical difficulties, namely too large numbers. This can be done by setting each component of u to the fractional part of its absolute value with the original sign. An equivalent cut is obtained then, cf. [16].

GOMORY mixed-integer cuts can be constructed as follows. In [33] the following function is described: $F_\alpha(d) = \lfloor d \rfloor + \max\{0, (\mathrm{fr}(d) - \alpha)/(1 - \alpha)\}$ which is a generalization of the rounding down function $\lfloor\rfloor$. The strongest cut is produced with $\alpha = \mathrm{fr}(ub)$. Inequality (14) with function F_α and $\alpha \neq 1$ may have a non-integer slack variable. Thus, other inequalities which depend on this cut have another type of coefficients for the slack of this cut: $\pi^r(s_v) = \overline{F}_{\alpha_r}(\sum_{t=1}^{r-1} u^r_{m+t}\pi^t(s_v))$ with $\overline{F}_\alpha(d) = \min\{0, d/(1-\alpha)\}$ and $\pi^v(s_v) = 1$. CHVÁTAL-GOMORY cuts cannot be constructed after mixed-integer cuts because they cannot handle the non-integer slacks. Note that a mixed-integer cut of the first rank (i.e., not dependent on any previous cuts) is stronger than or equivalent to the corresponding non-strengthened integer cut.

Now we should clarify where to get the multipliers u. Without column generation we would construct a fractional cut from row i of the simplex tableau with a non-integer basis variable x^B_i. This is modeled by taking row i of the optimum basis inverse for a u-vector if $x^B_i = ub \notin \mathbb{Z}$ ([33], basic cuts). Then for the optimum LP solution $\overline{x}$, $\lfloor uA \rfloor \overline{x} > \lfloor ub \rfloor$ holds which means that $\overline{x}$ is cut off by $\lfloor uA \rfloor \overline{x} \leq \lfloor ub \rfloor$.

While it is possible to set $-\mathbf{1} < u < \mathbf{1}$ for CHVÁTAL-GOMORY cuts, there is no such obvious transformation for mixed-integer cuts. However, no numerical difficulties were observed either. We construct all cuts which are possible and store them in a local pool; only a few most violated cuts are added to the LP.

4.2 Local Cuts

The term *cut-and-branch* [3] denotes the variation of the algorithm when cuts are constructed only at the root node; if no optimum is found and proved after a certain effort, branching is initiated and no new cuts are created in the subproblems. However, experience shows the effectiveness of creating *local cuts*, i.e., cuts valid only in a subproblem (and its descendants) [3].

Suppose we set lower and upper bounds on some variables as branching constraints. Then the LP of the current node looks like

$$\min\{cx : Ax = b, \mathcal{L} \leq x \leq \mathcal{U}\}.$$

Let $J = \{1, \ldots, n\}$. In an LP solution, let B be the index set of basic variables; let $L = \{j \in J : x_j = \mathcal{L}_j\}$, $U = \{j \in J : x_j = \mathcal{U}_j\}$ be the index sets of non-basic variables at their lower and upper bounds. Let A_B be the basis, $\overline{A} = A_B^{-1}A$, $\overline{b} = A_B^{-1}b$. The last simplex tableau implies

$$x_B + \overline{A}_L(x_L - \mathcal{L}_L) - \overline{A}_U(\mathcal{U}_U - x_U) = \overline{b} - \overline{A}_L\mathcal{L}_L - \overline{A}_U\mathcal{U}_U,$$

where $\mathcal{U}_U - x_U$ is substituted for the non-basic variables x_U which are at their upper bounds. Constructing inequality (14) for each row of this system with the rounding-down function, we obtain Chvátal-Gomory cuts

$$x_B + \lfloor\overline{A}_L\rfloor(x_L - \mathcal{L}_L) + \lfloor-\overline{A}_U\rfloor(\mathcal{U}_U - x_U) \leq \lfloor\overline{b} - \overline{A}_L\mathcal{L}_L - \overline{A}_U\mathcal{U}_U\rfloor$$

or, equivalently,

$$x_B + \lfloor\overline{A}_L\rfloor x_L - \lfloor-\overline{A}_U\rfloor x_U \leq \lfloor\overline{b} - \overline{A}_L\mathcal{L}_L - \overline{A}_U\mathcal{U}_U\rfloor + \lfloor\overline{A}_L\rfloor\mathcal{L}_L - \lfloor-\overline{A}_U\rfloor\mathcal{U}_U. \quad (17)$$

The rounding down function in (17) can be replaced by F_α which produces local mixed-integer cuts. Note that some variables already have upper bounds at the root node (see Node Preprocessing), thus some globally valid cuts may consider them.

5 Column Generation

In the standard column generation process, the dual multipliers are used to assess the cutting pattern. In practice, when implementing a column generation approach on the basis of a commercial IP Solver such as ILOG CPLEX [4], we may employ their standard procedures to solve (12), which was done in [27] for 2D-2CP. An advantage of this design is the low implementation effort. Traditionally there were two methods to solve (12): dynamic programming, applied already in [6], and branch-and-bound, cf. [35,36]. With cutting planes added, it is necessary to anticipate the values of the coefficients the column will have in the cuts during this process. These coefficients are non-linear and non-separable functions (15) of the elements of the cutting pattern vector. Finally, the dual information from the cut rows helps to evaluate the attractiveness of the column. Thus, selecting the most attractive column involves the solution of an integer programming problem with a non-separable objective function, which does not seem to have any usable structure, and has to be solved with branch-and-bound. It is much more difficult than the linear knapsack problem for column generation arising when no cuts are added in the LP.

In order to outline the principles of column generation with general-purpose cutting planes, we begin with the one-dimensional case. The branching strat-

egy was taken from [16]. Here we repeat it and then introduce a new upper bound for the pricing problem and a pricing procedure for 2D-2CP.

A technical issue is how to avoid generation of columns which are already in the LP. This is necessary when some variables have upper bounds and can be done easily in the context of a branch-and-bound method. The corresponding concept to find k^{th}*-best solutions* was properly described, e.g., in [10,11].

5.1 Column Generation for 1D-CSP

Let $d_1, \ldots, d_m$ be the simplex multipliers of the demand constraints (4). Without cutting planes, the pricing problem is the standard knapsack problem which can be solved, e.g., with branch-and-bound [36]. Pieces are sorted according to $d_1/l_1 \geq \ldots \geq d_m/l_m$ and feasible cutting patterns are enumerated in a decreasing lexicographical order.

Suppose μ cuts have been added; their dual multipliers are $d_{m+1}, \ldots, d_{m+\mu}$. The pricing problem is the knapsack problem

$$\max\{\overline{c}(a) : a = (a_1, \ldots, a_m) \in \mathbb{Z}_+^m, \sum_{i=1}^m l_i a_i \leq L, a \leq b\} \tag{18}$$

with the non-linear objective function

$$\overline{c}(a) = \sum_{i=1}^m d_i a_i + \sum_{r=1}^{\mu} d_r \pi^r(a). \tag{19}$$

$\overline{c}(a)$ is non-separable in the components of a because of the cut coefficients (15). Hence, dynamic programming cannot be applied, at least in its standard concept.

5.1.1 Enumeration Strategy

In our branch-and-bound method for the pricing problem, when cuts are added to the master LP, we preserve the same enumeration principle as without cuts, i.e., the lexicographical order. However, sorting of pieces has to reflect their approximate impact on the objective function. To estimate this impact, the objective function can be linearly approximated by omitting the non-linear superadditive function in the formula for cut coefficients (15). Thus, we define *approximative cut coefficients* recursively as

$$\widetilde{\pi}^r(a^j) = \sum_{i=1}^m u_i^r a_{ij} + \sum_{t=1}^{r-1} u_{m+t}^r \widetilde{\pi}^t(a^j), \quad r = 1, \ldots, \mu. \tag{20}$$

Now we have linearity: $\widetilde{\pi}^r(a) = \sum_{i=1}^m a_i \widetilde{\pi}^r(e^i)$ for all r, where e^i is the i-th unit vector. The approximative objective function is then

$$\widetilde{c}(a) = \sum_{i=1}^m d_i a_i + \sum_{r=1}^{\mu} d_{m+r} \widetilde{\pi}^r(a) = \sum_{i=1}^m \widetilde{d}_i a_i \tag{21}$$

with $\widetilde{d}_i = d_i + \sum_{r=1}^{\mu} d_{m+r}\widetilde{\pi}^r(e^i)$. The sorting of pieces according to $\widetilde{d}_1/l_1 \geq \ldots \geq \widetilde{d}_m/l_m$ proved very effective in finding good solutions [16].

5.1.2 The Upper Bound

In order to avoid the full enumeration, we need an upper bound for the objective value $\overline{c}(a)$ on a given set of patterns. This bound should be easy to calculate.

The idea of the bound proposed in [17] is to estimate the change of a cut coefficient when adding a piece of type i to the current subpattern: $\lfloor u(a + e_i)\rfloor \leq \lfloor ua\rfloor + \lceil u_i\rceil$. However, usually we want to add several items and this could not be represented satisfactorily in such a way. Here we propose the obvious estimation $ua - 1 \leq \lfloor ua\rfloor \leq ua$ or, generally, $ua - \alpha \leq F_\alpha(ua) \leq ua$.

Proposition 1 *The objective function* $\overline{c}(a)$ *of* (18) *is bounded from above as*

$$\overline{c}(a) \leq \widetilde{c}(a) + \sum_{r=1}^{\mu} \widetilde{\alpha}_r(d_{m+r}) \tag{22}$$

for all patterns a*, where the* approximation error of cut r *is*

$$\widetilde{\alpha}_r(x) = \begin{cases} x\overline{\alpha}_r, & x \geq 0 \\ x\underline{\alpha}_r, & x < 0 \end{cases} \tag{23}$$

and the upper/lower approximation errors of cut r *are*

$$\overline{\alpha}_r = \textstyle\sum_{t<r} \widetilde{\alpha}_t(u^r_{m+t}), \quad \underline{\alpha}_r = -\alpha - \sum_{t<r} \widetilde{\alpha}_t(-u^r_{m+t}).$$

Proof Assume $\widetilde{\pi}_r(a) + \underline{\alpha}_r \leq \pi_r(a) \leq \widetilde{\pi}_r(a) + \overline{\alpha}_r$ for $r \leq k-1$ (for $k = 2$ trivial) which implies the proposition is true for $\mu \leq k-1$. Then

$$\begin{aligned}\pi_k(a) &\stackrel{\text{def}}{=} F_{\alpha_k}\Big(\textstyle\sum_{i=1}^{m} u^k_i a_i + \sum_{t<k} u^k_{m+t}\pi_t(a)\Big) \\ &\stackrel{!}{\leq} \textstyle\sum_{i=1}^{m} u^k_i a_i + \sum_{t<k}\big(u^k_{m+t}\widetilde{\pi}_t(a) + \widetilde{\alpha}(u^k_{m+t})\big) \stackrel{\text{def}}{=} \widetilde{\pi}_k(a) + \overline{\alpha}_k\end{aligned}$$

(remember that the components of u may be either positive or negative) and

$$\pi_k(a) \stackrel{!}{\geq} -\alpha_k + \textstyle\sum_{i=1}^{m} u^k_i a_i + \sum_{t<k}\big(u^k_{m+t}\widetilde{\pi}_t(a) - \widetilde{\alpha}(-u^k_{m+t})\big) \stackrel{\text{def}}{=} \widetilde{\pi}_k(a) + \underline{\alpha}_k.$$

Thus, for the non-linear part of $\overline{c}()$, we get
$\sum_{r=1}^{k} d_{m+r}\pi_r(a) \leq \sum_{r=1}^{k} d_{m+r}\widetilde{\pi}_r(a) + \sum_{r=1}^{k} \widetilde{\alpha}(d_{m+r})$. ∎

This bound is linear and much easier to calculate than the original objective function. The approximation errors can be calculated *before enumeration*

because they are constant.

The enumeration procedure works as follows: having a partial solution $a' = (a'_1, \ldots, a'_p)$, we check the value of the upper bound (22) on it. If it is greater than the current lower bound, then we calculate the objective function and check whether it is a better solution; if it is, the lower bound is increased.

Now, there may be some free space in the knapsack: $dL = L - la'$. The question arises whether to fill this space by adding further items or modify a' by deleting some items from it, thus moving to another subset of solutions. In other words, we need to decide whether the subset of patterns with the first p components determined by a' **and some other non-zero components** can contain an improvement.

Proposition 2 *Let a partial solution $a' = (a'_1, \ldots, a'_p)$ be given. Then an upper bound for $\overline{c}()$ on all the patterns built by adding some items of types $i = p+1, \ldots, m$ to a' is given as follows:*

$$\max\{\overline{c}(a) : la \le L,\, a \le b,\, a \in \mathbb{Z}^m_+,\, (a_1, \ldots, a_p) = a',\, \exists i > p :\, a_i > 0\} \\ \le ub(p, dL(a')) + \widetilde{c}(a') + \textstyle\sum_r \widetilde{\alpha}_r(d_{m+r}), \quad (24)$$

where

$$ub(p, dL) = \max\{\widetilde{c}(\Delta a) : l\Delta a \le dL,\, \Delta a \le b,\, \Delta a \in \mathbb{Z}^m_+, \\ \Delta a_i = 0\, \forall i \le p,\, \Delta a \ne 0\} \quad (25)$$

and $dL(a') = L - la'$.

Proof follows from the linearity of $\widetilde{c}()$ and Proposition 1. ∎

Note that $ub(p, dL)$ is not dependent on the current objective value $\overline{c}(a')$. For all possible p and dL, $ub(p, dL)$ can be calculated efficiently using dynamic programming **before** the main enumeration. However, we found the following simplification sufficient in practice:

Proposition 3 *Let $\overline{i} = \arg\max\{\widetilde{d}_i :\, i > p\}$. If $\widetilde{d}_{\overline{i}} < 0$ then $ub(p, dL) \le \widetilde{d}_{\overline{i}}$. Otherwise $ub(p, dL) \le \max\{\widetilde{d}_i / l_i :\, i > p\} \cdot dL$.*

Proof for the case $\widetilde{d}_{\overline{i}} \ge 0$ follows from the relaxation of $ub(p, dL)$. Otherwise recall that we must add at least one item in the free space dL. ∎

The case $\widetilde{d}_{\overline{i}} < 0$ means that we are going to investigate **solutions with a smaller bound** than the bound of the current partial solution. However, a smaller bound does not mean that they cannot have a greater objective

value. Moreover, after encountering a forbidden pattern we have to look for alternative solutions, even with a worse value.

5.1.3 Practical Issues

The bound (24) is sometimes not strong enough when many cuts are added. If a good solution (with a negative reduced cost) is found and enumeration has been performed for several thousand backtracking steps (this limit should be increased with each generated column to reduce the number of columns), we terminate, see [17]. In the current implementation the following feature proved very practical: if no good solution is found for several million steps, the number of cuts is reduced in all nodes of the branch-and-price tree.

5.2 The Two-Dimensional Case

Let d_0 be the simplex multiplier of the width constraint (8), $d = (d_1, \ldots, d_m)$ those of the upper bound constraints (7). The pricing problem is

$$z^{2CG} = \min \{\underline{c}(a) = \textstyle\sum_{i=1}^{m}(-p_i - d_i)a_i - d_0 w(a) : \\ a \in \mathbb{Z}_+^m,\, la \le L,\, a \le b,\, w(a) \le W\Big\}. \quad (26)$$

The constraint $w(a) \le W$ may be relevant if W is reduced by branching on the main problem. Assume pieces are sorted so that $w_1 \ge w_2 \ge \cdots \ge w_m$. To reduce the problem to a series of 1D knapsacks, we consider each piece type to initialize the strip, which also fixes the width. When taking piece i first, we can add only those pieces which are not wider. Then

$$z^{2CG} = \min\{z'(-p - d, L - l_{i_0}, \widetilde{b}^{(i_0)}) - p_{i_0} - d_{i_0} - d_0 w_{i_0} : \\ i_0 = 1, \ldots, m,\, w_{i_0} \le W,\, b_{i_0} > 0\}, \quad (27)$$

where

$$z'(\widetilde{d}, \widetilde{L}, \widetilde{b}) = \min\{\widetilde{d}a :\, a \in \mathbb{Z}_+^m,\, la \le \widetilde{L},\, a \le \widetilde{b}\} \quad (28)$$

and

$$\widetilde{b}_1^{(i_0)}, \ldots, \widetilde{b}_{i_o-1}^{(i_0)} = 0, \quad \widetilde{b}_{i_0}^{(i_0)} = b_{i_0} - 1, \quad \widetilde{b}_i^{(i_0)} = b_i,\, \forall i > i_0. \quad (29)$$

The initial upper bound for each solution of (28) is set using the best known solution value. The proposed scheme allows us to avoid repeated investigation of equal strip patterns. This scheme also applies if cuts are added.

Table 1
Notations for test results

v_1, v_2	$l_i \in [v_1 L, v_2 L) \cap \mathbb{Z}$
IP	the best integer solution value
$\mathrm{IP}_{\min}$, $\mathrm{IP}_{\max}$	the minimum / maximum best solution value among the class
inst_name*	*: the 1D-CSP instance is non-IRUP
LP	the strongest LP value after branching / adding cuts
LPval0	the initial LP value (no cuts/branching)
t_{LP}, t_{IP}	time to obtain the best LP bound / integer solution, seconds
t_{opt}	optimum solution time
N_{opt}, N_{subopt}	number of optimally solved / unsolved instances
$N_{\mathrm{subopt}}^{\mathrm{BCP}}$, $N_{\mathrm{subopt}}^{\mathrm{BP}}$	number of instances unsolved by BCP (i.e., with cuts) and BP
N_+	number of instances unsolved before applying cuts / branching
nodes	number of processed nodes
iter	number of cutting plane iterations in all nodes
$\mathrm{col}_{\mathrm{LP}}$, $\mathrm{col}_{\mathrm{IP}}$	number of columns in the initial LP optimum / all columns
MIP	number of cases in which the best solution was found
	by the CPLEX MIP solver
N_{nI}	number of proven non-IRUP 1D-CSP instances

6 Computational Results

We used the ILOG CPLEX 7.5 Callable Library on an Athlon K7 1400 PC under Linux to solve the LP relaxations and the mixed-integer solver included there to find integer solutions on the current restricted LP. Some of the test instances for 1D-CSP can be obtained from *www.math.tu-dresden.de/~capad*; 2D-2CP instances also from *ftp://panoramix.univ-paris1.fr/pub/CERMSEM/hifi/2Dcutting/*. Notations for results are given in Table 1.

6.1 One-Dimensional Stock Cutting

At first we present some benchmark results for (today's) large instances: $m \in \{200, 400\}$. Then we consider some special instance sets: the 'hard28' set of Schoenfield [20] and Falkenauer's triplet instances [37].

6.1.1 Benchmark Results

To benchmark the behavior of the algorithm on 1D-CSP, we could not use any classes of previous authors because they computed up to $m = 100$ with at most 100 instances per class which is at present time not enough to find difficulties. Some 1D-BPP instances have larger m, see, e.g., the triplet instances below. Some other 1D-BPP classes, e.g., in the OR Library and [7], have 'formally' up to 1000 piece types, but when $L = 150$ and l_i are integer, the actual number of distinct piece types must obviously be much smaller. In the benchmark table below we see that in all classes with $m = 200$ there are less than 1% of unsolved instances. Thus, it was necessary to increase the problem size m and the number of instances per class. We compute $N = 1000$ instances per class for $m = 200$ and 400 and one difficult class with $N = 10000$. The time limit is 2 minutes per instance.

Moreover, a comparison to other algorithms is questionable because we have an excellent rounding procedure which reduces the number of instances unsolved in the root by a factor of 10 [17], and an excellent heuristic SVC, as illustrated by the values of N_+ in Table 2.

Following the tradition established in [38], we generate instances with item lengths l_i uniformly distributed in $[v_1 L, v_2 L) \cap \mathbb{Z}$, $v_1 < v_2$, where

$$v_1 \in \{0.001, 0.05, 0.15, 0.25\} \quad \text{and} \quad v_2 \in \{0.2, 0.3, \ldots, 0.8\}, \tag{30}$$

and order demands uniformly distributed in $[b_{\min}, b_{\max}) \cap \mathbb{Z}$. L is chosen as $100000 = 10^5$ because for smaller L we usually have too many equal sizes in classes such as $(v_1, v_2) = (0.15, 0.2)$. Equal items are merged, resulting in a slightly smaller average dimension m. We define the *basic class* by

$$(v_1, v_2) = (0.001, 0.7), (b_{\min}, b_{\max}) = (1, 100), m = 200. \tag{31}$$

By always varying just one or two of these parameter groups at a time, relatively to the basic class, we obtain different classes and avoid an explosion of the number of classes. A reason to choose $(v_1, v_2) = (0.001, 0.7)$ as basic is that it had the largest number of unsolved instances: 25 out of 1000 at the root node, 5 without cuts, and 9 with cuts, as shown in Table 2. Actually, the instances in the hard28 set (see the next subsection), which were collected from a huge test, have piece sizes distributed over nearly the same range.

The branching strategy was diving dfs with 10 nodes selected according to bfs, then m nodes dfs in the subtree of the last bfs node, and so on. $t_{\text{IP}}^{\text{BP}}$ and $t_{\text{IP}}^{\text{BCP}}$ give the average time to find the best integer solution for the instances which could not be solved in the root, for BP and BCP, respectively. Note that the time limit is 2 minutes and BP has usually found more optimal solutions than BCP.

Table 2
1D-CSP: benchmark results

	N_+	$N^{\mathrm{BP}}_{\mathrm{subopt}}$	$N^{\mathrm{BCP}}_{\mathrm{subopt}}$	t_{LP}	$t^{\mathrm{BP}}_{\mathrm{IP}}$	$t^{\mathrm{BCP}}_{\mathrm{IP}}$	N_{nI}	MIP	$\mathrm{IP}_{\mathrm{min}}$	$\mathrm{IP}_{\mathrm{max}}$	$\mathrm{col}_{\mathrm{IP}}$
$v_1 : 0.001$, $v_2 : 0.2$	0			17.5			0	0	811	1216	894
$v_2 : 0.3$	3	2	2	7.4	32.5	32.7	0	0	1214	1822	1027
$v_2 : 0.4$	5	2	1	7.8	21.6	24.9	0	0	1617	2428	1186
$v_2 : 0.5$	4	4	4	16.6	20.1	19.7	0	0	2020	3034	1435
$v_2 : 0.6$	10	3	4	11.5	14.7	14.3	0	0	2423	3640	1289
%% $v_2 : 0.7$	25	5	9	4.1	20.0	10.5	0	2	2826	4263	1083
$v_2 : 0.8$	19	0	2	1.6	9.2	6.4	0	2	3229	5370	845
$v_1 : 0.05$, $v_2 : 0.2$	0			9.6			0	0	1055	1470	788
$v_2 : 0.3$	0			5.4			0	0	1459	2075	1063
$v_2 : 0.4$	1	0	0	6.5	13.6	17.7	0	0	1863	2679	1212
$v_2 : 0.5$	3	1	1	15.3	51.4	47.3	0	0	2266	3284	1402
$v_2 : 0.6$	13	1	4	5.4	29.9	21.4	0	0	2670	3890	1160
$v_2 : 0.7$	21	0	6	2.4	17.2	16.7	0	7	3075	4661	977
$v_2 : 0.8$	23	1	2	1.0	6.4	8.0	2	9	3482	5573	779
$v_1 : 0.15$, $v_2 : 0.2$	-	-	-	-	-	-	-	-	-	-	-
$v_2 : 0.3$	2	0	1	7.3	32.8	35.5	0	0	1929	2595	1283
$v_2 : 0.4$	6	2	3	12.8	17.7	16.4	0	0	2354	3200	1425
$v_2 : 0.5$	17	4	7	5.4	21.4	13.5	0	0	2760	3806	1267
$v_2 : 0.6$	18	2	4	2.4	17.4	13.5	0	8	3166	4414	1051
$v_2 : 0.7$	11	1	1	1.1	6.0	3.0	0	5	3577	5175	850
$v_2 : 0.8$	8	0	0	0.4	1.9	1.4	0	3	3989	6209	681
$v_1 : 0.25$, $v_2 : 0.3$	0			0.9			0	0	2854	3750	459
$v_2 : 0.4$	0			2.7			0	0	2854	3750	887
$v_2 : 0.5$	9	0	1	3.2	17.9	27.0	0	5	3306	4523	1135
$v_2 : 0.6$	3	0	0	1.7	7.7	11.7	0	1	3666	5094	1395
$v_2 : 0.7$	0			1.1			0	0	4082	6073	1328
$v_2 : 0.8$	0			0.5			0	0	4663	7232	942
$v_1 : 10^{-4}$, $N : 10^4$	145	**32	**50	9.2	23.7	15.7	1	16	2711	4400	1092
$v_1 : 10^{-4}$, $b_{\max} : 1$	10	1	3	8.1	18.3	17.6	1	0	63	79	1386
$m : 400$	24	15	15	35.4	40.3	40.5	0	0	6197	8138	2274

%%: this is the basic class, all other classes differ in the indicated parameters.
**: for this class, 10000 instances were computed, 1000 instances for all other classes.

At first attempt we were unable to solve the classes with $v_2 = 0.2$. The many small items produce a huge amount of combinations, which makes pattern generation by branch-and-bound inefficient. Then we introduced early termination as in the case with cuts: after 10^6 backtracking steps, if a pattern with a negative reduced cost is found, we exit the knapsack solver. This allowed the solution of classes with $v_1 = 0.001$ and $v_1 = 0.05$ ($v_2 = 0.2$). In the class $(v_1, v_2) = (0.15, 0.2)$, piece sizes and dual multipliers are so strongly correlated that not far from LP optimum, no patterns with negative reduced cost could be found in an acceptable time. Similar difficulties were encountered in [11]. Special knapsack solver methods can be applied, cf. [36]. For this class, some combinatorial reductions are very effective [20]. Branching on hyperplanes induced by the arc flow formulation [32] seems advantageous for this class because dynamic programming can be used for column generation.

6.1.2 The 'hard28' Set

As we have seen, among randomly generated instances there are very few difficult ones (we forget the class $(v_1, v_2) = (0.15, 0.2)$ for the time). Thus, for illustration purposes it is interesting to consider some difficult instances. Especially, this would allow the investigation of non-MIRUP instances that are rare among randomly generated instances. We choose the 1D-BPP set 'hard28' of Schoenfield [20] with $m \in \{140, \ldots, 200\}$. They were selected from a huge test as the most difficult instances which could not be solved neither by the pure cutting plane algorithm from [17] nor by many reduction methods. One property of these instances is that the LP relaxation value is integer or slightly less which makes it difficult to find an optimum for the 23 IRUP instances among them.

Table 3 contains some problem data (see Table 1 for notations), then algorithmic results in two sections: pure branch-and-price (**BP**) and branch-and-cut-and-price (**BCP**) with at most two cuts in the LP formulation and new local cuts generated in every node. The CPLEX mixed-integer solver was not used because it takes very long time on these instances. Note that in IRUP instances, t_{opt} is equal to the time required to find the integer optimum. In non-IRUP instances, it is the time required to raise the LP bound to the IP value.

We see that without cuts, solution time is smaller for practically all instances except for the difficult non-IRUP instances bpp716 and bpp359. For 15 instances, cuts lead to a smaller number of nodes, which cannot be called systematic: cuts just alternate the LP solution without improving the integrality.

In BCP we allowed at most two cuts in the LP because adding more cuts complicated column generation significantly. However, in 1D-CSP usually not

Table 3
The hard28 set of the bin-packing problem

				BP – no cuts			**BCP**			
#	name	IP	LPval0	t_{opt}	nod	col_{IP}	t_{opt}	nod	iter	col_{IP}
1	bpp14*	62	60.998	14	168	995	24	130	141	1018
2	bpp832	60	59.997	4	10	809	17	23	33	862
3	bpp40	59	58.999	22	391	1184	126	267	301	1184
4	bpp360	62	62.000	3	4	985	5	2	7	969
5	bpp645	58	57.999	5	8	1026	41	61	81	1193
6	bpp742	64	64.000	7	22	1015	14	10	23	942
7	bpp766	62	61.999	4	15	855	9	8	15	844
8	bpp60	63	62.998	12	165	920	41	232	259	978
9	bpp13	67	67.000	22	161	1231	10	5	14	1119
10	bpp195	64	63.996	12	20	1196	14	6	16	1071
11	bpp709	67	66.999	18	119	1079	81	152	165	1137
12	bpp785	68	67.994	9	16	1071	19	16	14	1075
13	bpp47	71	71.000	4	6	961	13	7	18	966
14	bpp181	72	71.999	9	30	1026	17	33	49	1007
15	bpp359*	76	74.998	201	10412	1444	35	560	572	1021
16	bpp485	71	70.997	5	8	1113	19	16	33	1189
17	bpp640	74	74.000	2	2	832	4	1	5	832
18	bpp716*	76	75.000	105	6100	1400	4	2	5	1002
19	bpp119*	77	76.000	3	2	1045	6	2	5	1064
20	bpp144	73	73.000	23	173	1181	204	158	187	1262
21	bpp561	72	71.996	22	47	1347	36	21	45	1203
22	bpp781	71	70.999	9	10	1217	62	32	64	1419
23	bpp900	75	74.996	13	23	1242	29	16	36	1223
24	bpp175*	84	83.000	18	62	1261	34	46	56	1277
25	bpp178	80	79.995	4	3	1098	8	3	9	1111
26	bpp419	80	79.999	46	617	1644	42	41	55	1434
27	bpp531	83	83.000	1	0	971	1	0	0	971
28	bpp814	81	81.000	2	0	955	2	0	0	955
average				21	664	1111	33	66	79	1083

a single cut is active in IRUP instances (cf. [17]) and only one is active in non-IRUP instances.

We should mention the fact that already small changes in algorithm parameters, i.e., the starting basis or the intensity of the subgradient procedure, lead to drastic changes in the solution time of instance 18, mostly increasing the time to more than several days. This concerns in a lesser degree also instance 15 but on the other instances the algorithm is pretty stable. On the other hand, branching on hyperplanes seems to have other difficult instances and the stability of various branching schemes is a topic of further investigation.

6.1.3 Triplet Problems

Triplet problems [37] are 1D-BPP or 1D-CSP instances where an optimum solution has exactly three items in each stock bin and no waste. This means that the optimum value is equal to the material bound. In the OR Library of J. E. Beasley there is a collection of four 1D-BPP triplet sets, each with 20 instances. The instance sizes (m) are 60, 120, 249, and 501 in the four sets. The LP bound cannot be improved and cutting planes are of no use. Without cuts, the largest set with $m = 501$ was solved in 33 sec. per instance on average, which is almost identical to [12]. The time for the initial LP was 4 seconds, where 1300 columns were generated (total 1500 columns). With two cuts allowed in the LP, the time exceeded already 300 seconds. Note that non-LP-based methods [37,20] are very powerful on 1D-BPP and solve these problems much faster, which is not the case for the general 1D-CSP [21].

Last but not least, we mention that initializing the root LP by a subdiagonal FFD-like matrix was the best method.

6.2 Two-Dimensional Two-Stage Constrained Cutting

For 2D-2CP, the test set consists of a *medium* class [26,27] and a *large* class [28]. In the medium class, the first 14 instances are *weighted*, i.e., piece prices are not proportional to the areas, and 24 are *unweighted*. In the large class, the first 10 are unweighted and the other 10 are weighted. Problem sizes m and $n = \sum b_i$ are shown, e.g., in [30].

In guillotine cutting, we can distinguish between the first cut made horizontally or vertically, or, in other words, along the first dimension (L) or the second one (W). Unlike in 1D-CSP where we practically always have either the optimum LP bound (for IRUP instances) or the optimum solution obtained by rounding (for non-IRUP instances) from the beginning, in 2D-2CP the objective function and the LP bound can accept various values during

Table 4. 2D-2CP: medium instances, first cut along the first dimension

			HR	ILP-M1	ILP-M2	BCP. Chvátal-Gomory cuts						BCP. MI cuts	
	IP	LPval0	t_{opt}	t_{opt}	t_{opt}	t_{LP}	t_{IP}	nodes	iter	col_{LP}	col_{IP}	t_{LP}	t_{IP}
HH	10689	10872.0	0.2	0.1	0.0	0.1	0.0	0	2	11	12	0.0	0.0
2	2535	2658.9	2.9	0.1	0.1	1.6	0.0	628	669	25	97	1.9	0.0
3	1720	1893.6	0.2	0.1	0.1	0.2	0.4	88	99	40	53	0.2	0.2
A1	1820	1872.8	0.2	0.2	0.2	0.2	0.1	8	17	40	42	0.1	0.1
A2	2315	2391.6	0.8	0.3	0.3	0.8	0.2	28	36	43	57	0.9	0.2
STS2	4450	4529.0	5.6	6.9	4.8	0.3	0.5	44	57	61	71	0.3	0.5
STS4	9409	9580.1	9.2	4.5	0.9	6.2	0.2	1238	1279	41	162	4.3	0.2
CHL1	8360	8768.4	610.0	2.8	3.0	90.0	1.2	4332	4377	65	599	283.0	2.5
CHL2	2235	2272.2	0.1	0.0	0.0	0.1	0.0	0	1	22	22	0.1	0.0
CW1	6402	6820.1	1.0	1.0	0.3	0.5	0.0	10	18	52	58	0.8	0.1
CW2	5354	5462.7	2.1	0.3	0.2	0.4	0.1	6	14	72	74	0.2	0.1
CW3	5287	5427.3	6.4	0.8	0.5	0.2	0.1	18	32	83	94	0.2	0.2
Hchl2	9630	9742.2	N/A	12.9	20.2	5.9	5.4	576	704	75	143	9.2	5.4
Hchl9	5100	5195.0	858.0	2.1	1.1	0.2	0.2	0	1	74	75	0.2	0.2
average			115.1	2.3	2.3	7.5	0.5					21.5	0.6
2s	2430	2553.9	4.6	0.2	0.2	0.7	0.0	288	340	24	63	1.4	0.1
3s	2599	2668.6	0.1	0.1	0.1	0.5	0.0	28	41	39	42	0.5	0.1
A1s	2950	2970.4	0.1	0.1	0.2	0.1	0.0	0	1	39	39	0.1	0.1
A2s	3423	3474.3	0.2	0.5	0.6	0.2	0.0	6	14	42	44	0.7	0.1
STS2s	4569	4614.5	1.1	4.3	7.2	0.5	0.1	8	17	61	61	0.2	0.1
STS4s	9481	9643.4	8.9	3.1	7.4	1.4	0.2	328	366	41	99	2.4	0.2
OF1	2713	2713.0	0.1	0.0	0.1	0.0	0.0	0	0	22	22	0.0	0.0
OF2	2515	2716.3	0.1	0.1	0.1	0.7	0.0	40	55	22	34	0.2	0.0
W	2623	2785.0	0.2	0.2	0.1	0.3	0.1	100	108	39	84	0.2	0.1
CHL1s	13036	13193.9	6.1	2.0	2.4	0.8	0.1	12	20	61	66	44.1	0.1
CHL2s	3162	3306.9	0.1	0.1	0.1	0.1	0.1	54	66	22	33	0.2	0.0
A3	5380	5572.8	0.3	0.7	0.9	0.8	0.1	22	33	41	56	0.3	0.1
A4	5885	6100.8	1.6	0.8	0.6	0.3	0.2	60	76	39	68	0.4	0.2
A5	12553	13182.8	2.4	1.5	0.9	2.7	0.1	546	565	42	144	2.0	0.1
CHL5	363	379.0	0.1	0.0	0.0	0.0	0.0	0	1	22	22	0.0	0.0
CHL6	16572	16866.0	38.2	6.8	17.5	12.7	0.2	1532	1611	62	218	18.4	0.2
CHL7	16728	16813.4	44.6	9.3	65.1	0.4	0.2	54	65	71	80	0.3	0.2
CU1	12312	12500.0	1.0	9.0	2.1	0.7	0.1	14	22	51	54	0.5	0.1
CU2	26100	26250.0	2.7	2.0	0.7	0.7	0.2	6	15	69	70	0.1	0.2
Hchl3s	11961	12093.1	15.3	450.7	4.5	0.8	0.0	288	311	22	160	0.7	0.0
Hchl4s	11408	11753.5	507.0	192.4	2.1	1.5	1.0	264	321	23	114	2.1	0.7
Hchl6s	60170	60948.6	14.8	4.0	15.7	0.6	0.2	130	156	46	69	1.7	0.2
Hchl7s	62459	63154.5	96.2	56.9	280.3	3.5	0.6	284	309	82	112	11.6	1.4
Hchl8s	729	766.8	26.4	0.2	0.2	0.8	0.7	176	223	26	153	1.1	1.0
average			32.2	31.0	17.0	1.1	0.1					3.6	0.1

Table 5
2D-2CP: medium instances, first cut along the second dimension, average values only

	HR	ILP-M1	ILP-M2	BCP, CG cuts		BCP, MI cuts	
	t_{opt}	t_{opt}	t_{opt}	t_{LP}	t_{IP}	t_{LP}	t_{IP}
weighted	195	5.95	4.46	1.40	0.17	1.62	0.22
unweighted	20.28	28.43	63.39	2.06	0.05	2.25	0.35

optimization. Thus, we report both the times for the best LP bound t_{LP} and for the best integer solution t_{IP}. The maximum of them is the total solution time, at most 10 minutes.

6.2.1 Comparison with Other Methods

In Tables 4 and 5 we show, for the medium set, the optimum solution times of three other solution approaches. The combinatorial algorithm from [26] was tested on an UltraSparc10 250 MHz. Its runtimes are denoted by HR. M1 and M2 [27] are assignment formulations of 2D-2CP with a polynomial number of variables. In [29] we strengthen them by lexicographical constraints and recompute with CPLEX 7.5 on our Athlon K7 1400 PC. BCP has fewer difficulties with the instances and its average time is smaller on most classes.

Tables 6 and 7 show the results for the large classes. The second and the third column show the objective value and the solution time of the heuristic *extended substrip generation algorithm* (HESGA) [28] executed on an UltraSparc10 250 MHz. Then the data for M1 and M2 follow. M1 provides better solutions than the heuristic, especially on the last 10 weighted instances. The weighted case is much easier for M1 and M2 than the unweighted; M1 seems to provide better solutions than M2 but the smaller number of optimally solved instances suggests that M1 has a weaker LP relaxation.

Finally, results for BCP follow: the best objective value, the overall time and the time to find the best solution. The average objective value is mostly better than in the other approaches. The time to obtain the best solution is mostly negligible. No special difficulties are seen on the unweighted part, in contrast to M1 and M2.

6.2.2 Effectiveness of Algorithm Components for 2D-2CP

This subsection deals with the effectiveness of the three main components of BCP: branching, cutting planes, and rounding heuristics.

Table 8 contains, for each performance indicator, 3 lines which show results for pure branching, pure cutting plane algorithm (at most 40 cuts in the LP), and

Table 6
2D-2CP: large instances, first cut along the first dimension

	HESGA		M1		M2		BCP		
name	IP	time	IP	time	IP	time	IP	t_{All}	t_{IP}
ATP30	140168	7.3	138765	608.1	137813	601.9	140168	0.7	0.1
ATP31	818512	21.4	818364	610.7	813748	612.9	820260	601.0	11.5
ATP32	37880	6.2	37503	606.2	36940	609.0	37880	0.1	0.1
ATP33	234565	19.4	235580	608.2	233016	613.1	235580	0.1	0.1
ATP34	356159	12.7	356159	604.8	354962	600.4	356159	1.3	0.1
ATP35	613784	14.8	610554	601.9	611109	600.9	614429	230.0	8.3
ATP36	129262	6.4	129262	604.5	129262	361.2	129262	3.3	2.8
ATP37	382910	26.4	381224	608.7	380592	609.8	384478	4.8	0.1
ATP38	258221	14.2	259070	604.9	257540	610.2	259070	0.1	0.0
ATP39	265621	11.9	266135	603.1	264470	604.5	266135	46.4	1.5
average	323708.2	14.0	323261.6	606.1	321945.2	582.4	324342.1	88.7	2.4
unsolved		≥ 6		10		9		1	
ATP40	63945	12.1	63945	608.5	63622	610.0	63945	1.2	0.8
ATP41	202305	10.5	202305	119.9	202305	5.3	202305	0.0	0.0
ATP42	32589	21.8	32589	614.4	32589	253.2	32589	7.8	0.2
ATP43	208571	12.4	208998	414.9	208998	50.9	208998	158.0	97.1
ATP44	70678	12.5	70940	604.7	70940	42.5	70916	601.0	1.3
ATP45	74205	11.2	74205	31.7	74205	1.2	74205	0.0	0.0
ATP46	146402	14.1	146402	42.1	146402	10.6	146402	0.1	0.1
ATP47	143458	13.7	144317	105.9	144317	10.4	144168	601.0	0.1
ATP48	162032	12.5	165428	34.9	165428	7.7	165428	0.2	0.2
ATP49	204574	9.2	206965	602.5	206965	43.4	206965	601.0	0.1
average	130875.9	13.0	131609.4	317.9	131577.1	103.5	131592.1	197.0	9.9
unsolved		≥ 5		4		1		3	

pure optimization (no rounding heuristics, feasible solutions could be obtained only as integer LP solutions). We see that pure cuts are weaker than pure branching and the combined approach is the best, even without rounding. This shows the importance of local cuts at the nodes. Though the average best solution value without rounding is smaller than for pure branching on the weighted large set, the number of optimally solved instances is the same as with rounding. We also see that increasing the rounding frequency from every 100 nodes (BCP+rnd100, as in pure branching/cuts) to 10 nodes (BCP+rnd10)

Table 7
2D-2CP: large instances, first cut along the second dimension, summary values only

	HESGA		M1		M2		BCP		
name	IP	time	IP	time	IP	time	IP	t_{All}	t_{IP}
ATP30-39	323630.2	014.10	323671.6	0593.55	321843.4	0537.13	324547.1	0241.14	24.33
unsolved				9		8		4	
ATP40-49	130552.8	014.75	131112.8	0279.66	131118.7	028.52	131118.7	0246.51	38.53
unsolved				3		0		4	

leads to better solutions on the unweighted large set. Thus, finding an optimum integer solution is much easier for BCP than the optimality proof.

From Tables 4 and 5 it follows that strengthened Chvátal-Gomory cuts are more effective for 2D-2CP than Gomory mixed-integer cuts. At most 5 cuts were allowed in the LP and this number is necessary, because in 2D-2CP many cuts are usually active in an LP optimum, while in 1D-CSP only 1, at most 2 cuts are active. Only the best-first search strategy produces sensible results for 2D-2CP. FFD starting basis was the best in all aspects, as in 1D-CSP.

7 Conclusions and Outlook

A branch-and-price scheme for the one-dimensional cutting stock problem was enhanced by general-purpose cutting planes and also applied to the two-dimensional two-stage constrained cutting problem. Features and strategies of branch-and-cut(-and-price) were discussed. For 1D-CSP we preferred diving depth-first search, whereas best-first search is the clear winner for 2D-2CP. Comparisons with quite different algorithms for 2D-2CP show better performance on large-scale problem classes, especially on the unweighted instances which represent pure trim loss minimization. In 2D-2CP, very good or optimum solutions are found very easily, but the optimality proof is the bottleneck.

The optimality proof for 2D-2CP, similar to that for 1D-CSP with multiple stock lengths, is very difficult because possible objective values belong to the set of integer combinations of piece prices. This makes useful every means to strengthen the bound. Thus, a combination of cuts and branching is more effective than each approach alone. Local cuts are especially important. In 2D-2CP, cuts of high rank arise very often.

In 1D-CSP, most instances are IRUP, i.e., the root LP relaxation already gives the correct lower bound and pure branching is faster in finding an optimum

Table 8
2D-2CP: medium+large instances, pure approaches: pure branching (BP), pure cuts (CPA), pure branch-and-cut-and-price (no rounding, BCP)

Param.	Method	First cut along L				First cut along W			
		wei	unwei	largeU	largeW	wei	unwei	largeU	largeW
N_{all}		14	24	10	10	14	24	10	10
N_{opt}	BP	14	22	3	1	14	23	4	2
	CPA	9	10	3	5	10	14	3	4
	BCP	14	24	9	7	14	24	6	6
LPval0		5535	12376	325630	133803	5424	12241	326081	133172
LP	BP	5379	12184	324872	132419	5270	12089	324947	131655
	CPA	5418	12223	324979	132546	5292	12141	325151	131902
	BCP	5379	12172	324442	131812	5270	12089	324934	131265
IP	BP	5379	12168	324213	131570	5270	12089	324209	131061
	CPA	5372	12134	323988	131425	5267	12074	323903	130514
	BCP	5379	12172	324300	131244	5270	12089	324335	131051
	BCP+rnd100			324303	131592			324541	**131119**
	BCP+rnd10			**324342**	131592			**324547**	**131119**
	HESGA			323708	130876			323630	130553
	M1			323262	**131609**			323672	131113
	M2			321945	131577			321843	**131119**
t_{IP}	BP	6.7	2.2	6.7	60.0	0.1	0.5	9.8	92.7
	CPA	9.6	1.1	42.6	0.1	16.8	0.3	15.7	0.1
	BCP	1.4	0.4	6.7	11.4	0.4	1.1	66.8	41.5
nodes	BP	2434	4591	13963	15483	1101	3073	13394	18692
	CPA	0	0	0	0	0	0	0	0
	BCP	474	235	2112	2863	151	262	3827	5565
col_{IP}	BP	497	1061	4194	5859	393	581	4006	4118
	CPA	55	48	84	91	56	47	86	91
	BCP	131	86	702	962	89	117	1680	1079
MIP	BP	3	8	4	3	4	9	1	2
	CPA	4	10	4	3	5	10	1	3
	BCP	0	0	0	0	0	0	0	0

integer solution. Only in some non-IRUP instances, cuts help to prove optimality in less time.

Surprisingly, the pure BP approach is able to prove optimality for all 5 non-IRUP instances in the hard28 set. Let us remark that an optimum in non-IRUP instances is found in all known cases already at the root node, but the pure branch-and-price approach sometimes (very rarely) needs much time to close the gap and prove optimality, which is obviously a weaker deficiency than the other way around.

Another striking conclusion is that even an unbalanced branching scheme is often able to close the gap in the case of non-IRUP instances. A preliminary branch-and-price implementation with more balanced branching rules based on the arc flow formulation of 1D-CSP [32] seems to have no difficulties on non-IRUP instances. For 2D-2CP, a more balanced scheme would be *branching on slacks*. Slack variables e^i $(i = 1, \ldots, m)$ determine how many items of type i should be in a solution. Thus, branching on them is more 'spread' than on single patterns. However, this scheme does not guarantee optimality: even in a fractional LP solution, slacks may be integer. Thus, this scheme should be combined with another one.

The general-purpose cuts considered in this work make the pricing problem extremely complicated and much implementation effort is required. The question arises, whether any other classes of cuts, especially such that do not complicate the pricing problem, can provide a comparable contribution.

Acknowledgements

We are very grateful to the anonymous referees who made immense contributions to the presentation; to Jon Schoenfield for the 'hard28' set, for discussions, and for proofreading; to Joachim Kupke, Marc Peeters, and, quite recently, Cláudio Alves with José Manuel Valério de Carvalho for testing their branch-and-price implementations on the 'hard28' set.

References

[1] J. T. Linderoth, M. W. P. Savelsbergh, A computational study of strategies for mixed integer programming, INFORMS Journal on Computing 11 (1999) 173–187.

[2] A. Martin, General mixed-integer programming: computational issues for branch-and-cut algorithms, in: M. Jünger, D. Naddef (Eds.), Computat. Comb. Optimization, LNCS, Vol. 2241, 2001, pp. 1–25.

[3] E. Balas, M. Perregaard, Lift-and-project for mixed 0-1 programming: recent progress, Discrete Applied Mathematics 123 (1–3) (2002) 129–154.

[4] ILOG, ILOG optimization suite. Delivering a competitive advantage, White paper, ILOG Corporation, URL: `http://www.ilog.com` (2001).

[5] M. Jünger, S. Thienel, The ABACUS system for branch-and-cut-and-price algorithms in integer programming and combinatorial optimization, Software: Practice and Experience 30 (11) (2000) 1325–1352.

[6] P. C. Gilmore, R. E. Gomory, A linear programming approach to the cutting-stock problem, Operations Research 9 (1961) 849–859.

[7] P. H. Vance, C. Barnhart, E. L. Johnson, G. Nemhauser, Solving binary cutting stock problems by column generation and branch-and-bound, Comput. Opt. Appl. 3 (1994) 111–130.

[8] F. Vanderbeck, Computational study of a column generation algorithm for bin packing and cutting stock problems, Mathematical Programming, Ser. A 86 (3) (1999) 565–594.

[9] J. M. V. de Carvalho, LP models for bin-packing and cutting stock problems, European Journal of Operational Research 141 (2) (2002) 253–273.

[10] J. Kupke, Lösung von ganzzahligen Verschnittproblemen mit Branch-and-Price, Diplomarbeit, Universität zu Köln (1998).

[11] Z. Degraeve, M. Peeters, Optimal integer solutions to industrial cutting stock problems: Part 2, benchmark results, INFORMS Journal on Computing 15 (2003) 58–81.

[12] M. Peeters, One-dimensional cutting and packing: new problems and algorithms, Ph.D. thesis, Katholieke Universiteit Leuven, Belgium (2002).

[13] L. A. Wolsey, Integer Programming, Wiley Interscience Series in Discrete Mathematics and Optimization, John Wiley and Sons, Chichester, 1998.

[14] F. Vanderbeck, Exact algorithm for minimizing the number of setups in the one-dimensional cutting stock problem, Operations Research 48 (6) (2000) 915–926.

[15] C. Barnhart, C. A. Hane, P. H. Vance, Using branch-and-price-and-cut to solve origin-destination integer multicommodity flow problems, Oper. Res. 48 (2000) 318–326.

[16] G. Scheithauer, J. Terno, A. Müller, G. Belov, Solving one-dimensional cutting stock problems exactly with a cutting plane algorithm, Journal of the Operational Research Society 52 (2001) 1390–1401.

[17] G. Belov, G. Scheithauer, A cutting plane algorithm for the one-dimensional cutting stock problem with multiple stock lengths, European Journal of Operational Research 141 (2) (2002) 274–294, special issue on cutting and packing.

[18] G. Belov, G. Scheithauer, Setup and open stacks minimization in one-dimensional stock cutting, Technical report, Dresden University (2003).

[19] R. E. Johnston, E. Sadinlija, A new model for complete solutions to one-dimensional stock problems, European Journal of Operational Research 153 (2004) 176–183.

[20] J. E. Schoenfield, Fast, exact solution of open bin packing problems without linear programming, draft, US Army Space & Missile Defense Command, Huntsville, Alabama, USA (2002).

[21] S. Martello, P. Toth, Knapsack Problems – Algorithms and Computer Implementations, John Wiley and Sons, Chichester et al., 1990.

[22] C. Nitsche, G. Scheithauer, J. Terno, Tighter relaxations for the cutting stock problem, European Journal of Operational Research 112 (1999) 654–663.

[23] G. Scheithauer, J. Terno, The modified integer round-up property of the one-dimensional cutting stock problem, European Journal of Operational Research 84 (1995) 562–571.

[24] J. Rietz, Investigations of MIRUP for vector packing problems, Ph.D. thesis, Freiberg University, in German (2003).

[25] P. C. Gilmore, R. E. Gomory, Multistage cutting stock problems of two and more dimensions, Oper. Res. 13 (1965) 94–120.

[26] M. Hifi, C. Roucairol, Approximate and exact algorithm for constrained (un)weighted two-dimensional two-staged cutting stock problems, Journal of combinatorial optimization 5 (2001) 465–494.

[27] A. Lodi, M. Monaci, Integer linear programming models for 2-staged two-dimensional knapsack problems, Mathematical Programming, Ser. B 94 (2002) 257–278.

[28] M. Hifi, R. M'Hallah, Strip generation algorithms for constrained two-dimensional two-staged cutting problems, Working paper, Serie Bleu, Universite de Paris 1 (2003).

[29] G. Belov, G. Scheithauer, Variable width models for two-dimensional two-stage cutting, Technical report, Dresden University (2003).

[30] G. Belov, G. Scheithauer, A branch-and-cut-and-price algorithm for one-dimensional stock cutting and two-dimensional two-stage cutting, Technical report, Dresden University, uRL: `www.math.tu-dresden.de/~capad` (2003).

[31] C. Cordier, H. Marchand, R. Laundy, L. A. Wolsey, BC-OPT: A branch-and-cut code for mixed integer programs, Mathematical Programming 86 (2) (1999) 335–354.

[32] C. Alves, J. M. V. de Carvalho, A branch-and-price algorithm for integer variable sized bin-packing problems, Tech. rep., Universidade do Minho, Portugal (2003).

[33] G. L. Nemhauser, L. A. Wolsey, Integer and Combinatorial Optimization, John Wiley and Sons, New York, 1988.

[34] A. N. Letchford, A. Lodi, Strengthening Chvátal-Gomory cuts and Gomory fractional cuts, Operations Research Letters 30 (2) (2002) 74–82.

[35] P. C. Gilmore, R. E. Gomory, A linear programming approach to the cutting-stock problem (Part II), Operations Research 11 (1963) 863–887.

[36] H. Kellerer, U. Pferschy, D. Pisinger, Knapsack Problems, Springer Verlag, 2004.

[37] E. Falkenauer, A hybrid grouping genetic algorithm for bin packing, Journal of Heuristics 2 (1) (1996) 5–30.

[38] G. Wäscher, T. Gau, Heuristics for the one-dimensional cutting stock problem: A computational study, OR Spektrum 18 (1996) 131–144.