Using Decomposition Techniques and Constraint Programming for Solving the Two-Dimensional Bin-Packing Problem

Article *in* Informs Journal on Computing · February 2007

DOI: 10.1287/ijoc.1060.0181 · Source: DBLP

2 authors:

David Pisinger
Technical University of Denmark
170 PUBLICATIONS **11,852** CITATIONS

Mikkel Mühldorff Sigurd
Maersk Line, Copenhagen, Denmark
13 PUBLICATIONS **649** CITATIONS

INFORMS Journal on Computing
Vol. 19, No. 1, Winter 2007, pp. 36–51
ISSN 1091-9856 | EISSN 1526-5528 | 07 | 1901 | 0036

DOI 10.1287/ijoc.1060.0181
© 2007 INFORMS

Using Decomposition Techniques and Constraint Programming for Solving the Two-Dimensional Bin-Packing Problem

David Pisinger, Mikkel Sigurd
DIKU, University of Copenhagen, Universitetsparken 1, Copenhagen, Denmark
{pisinger@diku.dk, msigurd@gmail.com}

The two-dimensional bin-packing problem is the problem of orthogonally packing a given set of rectangles into a minimum number of two-dimensional rectangular bins. The problem is $\mathcal{NP}$-hard and very difficult to solve in practice as no good mixed integer programming (MIP) formulation has been found for the packing problem. We propose an algorithm based on the well-known Dantzig-Wolfe decomposition where the master problem deals with the production constraints on the rectangles while the subproblem deals with the packing of rectangles into a single bin. The latter problem is solved as a constraint-satisfaction problem (CSP), which makes it possible to formulate a number of additional constraints that may be difficult to formulate as MIP models. This includes guillotine-cutting requirements, relative positions, fixed positions and irregular bins. The CSP approach uses forward propagation to prune inferior arrangements of rectangles. Unsuccessful attempts to pack rectangles into a bin are brought back to the master model as valid inequalities. Hence, CSP is used not only to solve the pricing problem but also to generate valid inequalities in a branch-and-cut system. Using delayed column-generation, we obtain lower bounds of very good quality in reasonable time. In all instances considered, we obtain similar or better bounds than previously published. Several instances with up to $n = 100$ rectangles are solved to optimality through the developed branch-and-price-and-cut algorithm.

Key words: programming, integer, algorithms, branch and bound; programming, integer, algorithms, cutting plane; production-scheduling, cutting stock
History: Accepted by John Hooker, Area Editor for Constraint Programming and Optimization; received December 2002; revised June 2004, July 2005; accepted January 2006.

1. Introduction

Optimization techniques have helped the industry to solve very complex planning problems. Two techniques have demonstrated their advantages: integer linear programming (ILP) dealing with optimization problems formulated in linear form, and constraint-satisfaction problems (CSP) dealing with decision problems formulated as logic problems. Solvers for CSP and ILP are based on branch-and-infer techniques (Bockmayr and Kasper 1998), but ILP solvers use bounds from linear programming to prune the search tree while CSP algorithms rely on advanced constraint propagation and graph theory to bound the tree.

Despite progress in both fields, several industrial problems cannot be solved by present techniques. This typically includes problems that cannot be formulated efficiently in linear form, or where the number of decision variables is so large that CSP techniques are not able to deal with the exponential growth of the solution space. Such problems include manpower scheduling problems (e.g. of cabin crew to airplanes), machine-scheduling problems (e.g. of tasks in a huge factory), timetabling problems (e.g. of trains and busses), and packing/cutting problems (e.g. cutting metal with minimum waste).

A natural approach is to *decompose* difficult problems such that ILP and CSP can be used to solve separate parts of the problem via "hybrid" algorithms. Eremin and Wallace (2001) presented a general framework for hybridization based on *Benders' decomposition* and *Dantzig-Wolfe decomposition*. Benders' decomposition is a *row-generation* approach. A properly structured problem is split into a linear *master model* and a *subproblem* that returns a number of *Benders cuts*. In Dantzig-Wolfe decomposition a suitable problem is split into a linear *master model* and a more complicated *subproblem* that, e.g., can be solved by CSP. Since a complete formulation of the master model for many relevant problems becomes exponentially large, *delayed column-generation* is used to write up the master model gradually, hoping that only a small part of it needs to be considered to prove optimality. The subproblem of finding a column with largest negative reduced cost to enter the master model is known as the *pricing problem*. Expressing the negativeness of the reduced costs as a constraint, the pricing problem may be converted into a CSP. Delayed column-generation only solves the master problem to LP-optimality, so to find an IP-optimal solution the approach needs to be combined with branch-and-bound, known as a *branch-and-price* algorithm.

Practical implementations of hybrid algorithms using CSP for solving the subproblems have been reported in Junker et al. (1999) (crew scheduling) and Eremin and Wallace (2001) (minimal perturbation scheduling problem).

We present a hybrid algorithm for solving the two-dimensional bin-packing problem (2DBPP). The 2DBPP seeks to pack a set $\mathcal{R}$ of n rectangles with dimensions $w_i \times h_i$ into identical larger rectangular bins with dimensions $W \times H$ using the fewest bins possible. The rectangles may not be rotated, they may not overlap, and they must be parallel to the sides of the bin. The 2DBPP is a straightforward generalization of the one-dimensional bin-packing problem (1DBPP) in which a set of weights have to be packed into the minimum number of bins. Both problems are known to be $\mathcal{NP}$-hard in the strong sense and are also in practice very difficult to solve (Martello and Vigo 1998).

Numerous heuristics for 2DBPP have been presented. Greedy heuristics based on concepts like *next-fit*, *first-fit*, *best-fit*, and their performance ratios are surveyed in Lodi et al. (2002). The first *local-search* algorithm was presented by Bengtsson (1982). Starting with a packing of a subset of the rectangles, the remaining rectangles are iteratively packed into the bin having maximum unused space. Metaheuristics based on *tabu search* were presented in Lodi et al. (1999), while Faroe et al. (2003) used the concept of *guided local search* (Voudouris and Tsang 1999). Dowsland (1993) presented a *simulated-annealing* algorithm for the strip-packing problem in two dimensions, which tries to pack the rectangles into one containing rectangle. When a feasible packing has been found, the height of the containing rectangle is reduced and a new feasible packing is sought. Monaci and Toth (2006) presented a heuristic based on *column-generation* and *set-covering*. The algorithm works in two phases: first a large number of feasible single bin-packings are generated by four different heuristics. Then the set-covering problem involving the generated packings is solved by a heuristic algorithm. Boschetti and Mingozzi (2003b) presented an *iterative greedy* heuristic. Each rectangle is assigned a price that is used to define the order of the rectangles filled into the bins. After each iteration the prices are adjusted and the algorithm is repeated.

For a comprehensive overview of models and algorithms for two-dimensional cutting problems see Belov (2003). Specialized algorithms for the 2DBPP are in Martello and Vigo (1998) and Fekete and Schepers (2001), which present several lower bounds on the solution value using, respectively, partitioning of rectangles in various classes and dual feasible functions. Boschetti and Mingozzi (2003a) presented a new lower bound that dominates the bounds of Martello and Vigo (1998) and Fekete and Schepers (2001). Boschetti and Mingozzi (2003b) generalized these bounds to the case where rectangles may be rotated 90 degrees. Considering the same variant of the problem, Dell'Amico et al. (2002) presented a lower bound and an exact branch-and-bound algorithm. Exact algorithms based on integer programming were presented in Chen et al. (1995) for the more general three-dimensional case. Padberg (2000) present an extended formulation and subjects it to polyhedral analysis, reaching a tighter LP relaxation.

A framework for deriving lower bounds through column-generation was implicitly presented in Gilmore and Gomory (1961, 1963) through a straightforward generalization of the techniques developed for the 1DBPP. However, few column-generation algorithms for the 2DBPP have been presented, perhaps because the pricing problem for 1DBPP is a simple knapsack problem for which several good algorithms are available (Kellerer et al. 2004), while the pricing problem of 2DBPP becomes a two-dimensional knapsack problem (2DKP) that is much harder to solve due to the geometrical structure. Hadjiconstantinou and Christofides (1995) studied the 2DKP but were able to solve instances of only moderate size. Fekete and Schepers (2007) presented a two-phase algorithm that first selects a subset of rectangles with large profits, and then tests feasibility through an enumerative algorithm based on isomorphic packing classes. Caprara and Monaci (2004) presented a $(\frac{1}{3} - \epsilon)$-approximation algorithm for the 2DKP and developed four exact algorithms based on various enumeration schemes. Belov (2003) chose a different direction, restricting the 2DBPP to two-stage guillotine patterns that can be solved through dynamic programming (see e.g. Hifi 2001). For this simpler problem he was able to solve various problems from the literature, using a branch-and-price framework.

We present a hybrid algorithm for the 2DBPP based on column-generation where the associated pricing problem is solved through CSP. Following the idea of Fekete and Schepers (2007), the associated 2DKP is split into a one-dimensional multi-constrained knapsack problem (1DMCKP) that selects a subset of rectangles, and a feasibility test checking whether the set of rectangles fits into a single bin. The latter problem is solved through CSP, which not only leads to a well-performing algorithm, but also adds flexibility to the packing patterns returned. Using CSP we are able to model constraints as guillotine-cutting requirements, relative positions, fixed positions, and irregular bins. Handling such constraints is necessary to solve real two-dimensional packing and cutting problems.

An important property of lower bounds from column-generation is their ability to handle additional constraints from the branching process, resulting in tighter bounds as constraints are added. Except for the bounds of Fekete and Schepers (2007), most bounds return the same lower bound throughout the branching process except if some special auxiliary properties (like the closing of a bin) can be used to strengthen the bound.

We also present a framework in which additional constraints detected by the CSP subproblem can be brought back to the ancestor problem: unsuccessful attempts to pack sets of rectangles into a single bin are added to the 1DMCKP as valid inequalities, resulting in a *branch-and-cut* system. Finally, we show how *Lagrangian relaxation* can be seen as a third way of decomposing the problem into a linear master problem and a subproblem solved through CSP, which makes it possible to terminate the column-generation earlier in a branch-and-bound algorithm, avoiding some tailing-off problems shown by Lasdon (1970).

In the following section we formulate the problem formally as an integer-linear model. Unfortunately, the LP-relaxation of the model leads to a lower bound that is quite weak, so the model is difficult to solve by general MIP solvers. A tighter formulation based on *Dantzig-Wolfe decomposition* is hence proposed in §2.1. The master problem is a set-covering problem, solved through *delayed column-generation* as described in §3, while the pricing problem becomes a two-dimensional knapsack problem. In §4 the pricing problem is split into a 1DMCKP selecting the most profitable rectangles to pack, and a two-dimensional packing problem in the decision form, and §5 shows how the decision problem may be solved through a combination of constraint programming and branch-and-cut. Section 6 describes a heuristic algorithm for solving the pricing problem, and §7 shows how we avoid tailing-off problems in the column-generation. Section 8 describes how an initial feasible solution may be obtained through heuristic methods, and how the primal bound is improved during the search. Section 9 describes the branch-and-price algorithm. We conclude with an extensive computational study in §10 where the developed branch-and-price algorithm is tested on a set of randomly generated test instances from Berkey and Wang (1987) and Martello and Vigo (1998). The bounds obtained through column-generation are tighter than any previous bounds and the developed algorithm can also solve larger problems to optimality than previously reported.

2. Problem Formulation

Assume that a set $\mathcal{R} = \{1, \ldots, n\}$ of rectangles is given, rectangle i having width w_i and height h_i. Infinitely many bins are available, each having width W and height H. Using the modeling technique of Onodera et al. (1991) and Chen et al. (1995) we formulate the 2DBPP in ILP form as follows: the binary variable l_{ij} is 1 iff rectangle i is located left of rectangle j, and similarly b_{ij} is 1 iff rectangle i is located below rectangle j. The lower left coordinates of rectangle i are (x_i, y_i) while m_i is the number of the bin containing rectangle i, and v is the number of bins used. Finally, the binary variable p_{ij} attains the value 1 iff $m_i < m_j$.

To ensure that no two rectangles overlap, the following inequalities must hold

$$l_{ij} + l_{ji} + b_{ij} + b_{ji} + p_{ij} + p_{ji} \geq 1 \quad i, j \in \mathcal{R},\ i < j \tag{1}$$

$$\begin{gathered} l_{ij} = 1 \Rightarrow x_i + w_i \leq x_j; \qquad b_{ij} = 1 \Rightarrow y_i + h_i \leq y_j; \\ p_{ij} = 1 \Rightarrow m_i + 1 \leq m_j. \end{gathered} \tag{2}$$

No part of the rectangles may exceed the bin, hence $0 \leq x_i \leq W - w_i$ and $0 \leq y_i \leq H - h_i$. As v is the number of bins used we have $1 \leq m_i \leq v$.

The 2DBPP can now be formulated as the following MIP:

$$\begin{aligned} \min\ & v \\ \text{s.t. } & l_{ij} + l_{ji} + b_{ij} + b_{ji} + p_{ij} + p_{ji} \geq 1 \quad i, j \in \mathcal{R},\ i < j \\ & x_i - x_j + W l_{ij} \leq W - w_i \quad i, j \in \mathcal{R} \\ & y_i - y_j + H b_{ij} \leq H - h_i \quad i, j \in \mathcal{R} \\ & m_i - m_j + n p_{ij} \leq n - 1 \quad i, j \in \mathcal{R} \\ & 0 \leq x_i \leq W - w_i \quad i \in \mathcal{R} \\ & 0 \leq y_i \leq H - h_i \quad i \in \mathcal{R} \\ & 1 \leq m_i \leq v \quad i \in \mathcal{R} \\ & l_{ij}, b_{ij}, p_{ij} \in \{0, 1\} \quad i, j \in \mathcal{R} \\ & x_i, y_i \in \mathbb{R} \quad i \in \mathcal{R} \\ & m_i, v \in \mathbb{N} \quad i \in \mathcal{R}. \end{aligned} \tag{3}$$

The model has $3n^2$ binary variables and $3n$ continuous variables. The number of constraints is $7n^2/2 + 6n$. Unfortunately the formulation contains several symmetric solutions so it is very difficult to solve by ordinary MIP solvers. To break some of the symmetries we add the additional constraints $m_i \le i$, $i \in \mathcal{R}$ demanding that rectangle 1 is placed in the first bin, rectangle 2 is placed in one of the two first bins, etc. However, the model remains difficult to solve.

2.1. Set-Covering Model

If $\mathcal{P}$ is the set of all feasible packings of a single bin, we may reformulate the 2DBPP as a set-covering problem. For every feasible packing $p \in \mathcal{P}$, we use the binary variable x_p to indicate whether packing p is chosen in the solution of the set-covering model. For every rectangle $i \in \mathcal{R}$ and every feasible packing $p \in \mathcal{P}$ we set $\delta_i^p = 1$ iff packing p contains rectangle i. In this way the set-covering formulation of the 2DBPP becomes

$$\min\left\{\sum_{p\in\mathcal{P}} x_p \;\middle|\; \sum_{p\in\mathcal{P}} x_p\delta_i^p \ge 1,\ \forall\, i\in\mathcal{R};\ x_p\in\{0,1\},\ \forall\, p\in\mathcal{P}\right\}. \quad (4)$$

We minimize the number of bins used ensuring that every rectangle is included in some bin. The LP relaxation of the set-covering model is

$$\min\left\{\sum_{p\in\mathcal{P}} x_p \;\middle|\; \sum_{p\in\mathcal{P}} x_p\delta_i^p \ge 1,\ \forall\, i\in\mathcal{R};\ x_p\ge 0,\ \forall\, p\in\mathcal{P}\right\}. \quad (5)$$

This gives a fairly tight lower bound on the IP solution. In particular, (5) yields a tighter lower bound than does the LP relaxation of (3) since every solution to (5) is also a solution to the LP relaxation of (3), while the opposite is not true in general.

Since $|\mathcal{P}|$ may be exponentially large in the input size, we will use *delayed column-generation* for solving the LP-relaxed set-covering model.

3. Delayed Column-Generation

In delayed column-generation we solve the LP relaxed set-covering model (5) for a subset $\mathcal{P}'$ of the feasible packings, gradually adding new packings $p \in \mathcal{P}\backslash\mathcal{P}'$ with minimum reduced cost according to the well-known Dantzig rule. The *restricted master problem* (RMP) is

$$\min\left\{\sum_{p\in\mathcal{P}'} x_p \;\middle|\; \sum_{p\in\mathcal{P}'} x_p\delta_i^p \ge 1,\ \forall\, i\in\mathcal{R};\ x_p\ge 0,\ \forall\, p\in\mathcal{P}'\right\}. \quad (6)$$

Here $\mathcal{P}'$ is a small set of variables such that a feasible solution exists for RMP. Initially we choose $\mathcal{P}'$ as those packings found by a greedy heuristic (see §8) for the 2DBPP. Since we will only add packings to the RMP in the following iterations, the LP will always have a feasible solution (branching may render the RMP infeasible, but infeasible branch-and-bound nodes can safely be pruned).

In every iteration of the column-generation we will solve the RMP. Let π_i, $i \in \mathcal{R}$ be the dual variables associated with the current solution x_p to the RMP. The dual linear program of (5) is

$$\max\left\{\sum_{r\in\mathcal{R}} \pi_i \;\middle|\; \sum_{r\in\mathcal{R}} \pi_i\delta_i^p \le 1,\ \forall\, p\in\mathcal{P};\ \pi_i\ge 0,\ \forall\, i\in\mathcal{R}\right\}. \quad (7)$$

If the dual variables π_i associated with (5) satisfy feasibility in (7) we know from the weak-duality theorem that x_p is an optimal solution to (5). Otherwise the dual solution π_i must violate some constraint of (7). The violation of a constraint of type $\sum_{r\in\mathcal{R}} \pi_i\delta_i^p \le 1$ corresponding to a packing $p \in \mathcal{P}$ is $c_p^\pi = 1 - \sum_{i\in\mathcal{R}} \delta_i^p \pi_i$, which is called the *reduced cost* of packing p with respect to the dual variables π_i. Clearly, if $c_p^\pi \ge 0$ for all $p \in \mathcal{P}$, π_i is a feasible solution to the dual LP, which means that x_p is an optimal solution to (5).

In every iteration of the column-generation procedure we solve the *pricing problem*, which is to find a feasible packing $p \in \mathcal{P}$ with smallest reduced cost c_p^π. If $c_p^\pi \ge 0$ the LP has been solved to optimality. Otherwise we add the packing to the RMP. This is the same as adding the most violated constraint of (7) to the dual LP of the RMP, improving feasibility of (7). An exact and a heuristic algorithm for this problem will be described in §4.

The next iteration of the column-generation procedure starts by solving the slightly larger RMP, obtaining new dual variables. The procedure continues until at some point no packing with negative reduced cost can be found. Clearly, the column-generation procedure will terminate at some point, since there are only finitely many feasible packings; since packings already in the RMP have nonnegative reduced cost, no packings are added more than once. In practice, the column-generation procedure adds only a small fraction of the feasible packings before terminating, so it has much better average performance than would be expected from the worst-case complexity.

4. Solving the Pricing Problem

The *pricing problem* finds a feasible packing p of a single bin with the smallest reduced cost c_p^π. If we define the profit p_i of every rectangle $i \in \mathcal{R}$: $p_i = -\pi_i$ then the pricing problem is a 2DKP with respect to the profits p_i, the rectangle sizes (w_i, h_i) and the knapsack size (W, H). The 2DKP can be modeled using the following variables: u_i is binary variable with value 1 iff rectangle i is packed in the bin, $l_{ij} = 1$ iff rectangle i is located left of rectangle j, and $b_{ij} = 1$ iff rectangle i is located below j. Finally, (x_i, y_i) are the lower left coordinates of rectangle i. The problem can now be formulated as

$$\begin{aligned}
\max\ & \sum_{i=1}^{n} p_i u_i \\
\text{s.t. } & l_{ij} + l_{ji} + b_{ij} + b_{ji} + (1-u_i) + (1-u_j) \ge 1 \quad i, j \in \mathcal{R},\ i<j \\
& x_i - x_j + W l_{ij} \le W - w_i \quad i, j \in \mathcal{R} \\
& y_i - y_j + H b_{ij} \le H - h_i \quad i, j \in \mathcal{R} \\
& 0 \le x_i \le W - w_i \quad i \in \mathcal{R} \\
& 0 \le y_i \le H - h_i \quad i \in \mathcal{R} \\
& l_{ij}, b_{ij} \in \{0,1\} \quad i, j \in \mathcal{R} \\
& u_i \in \{0,1\} \quad i \in \mathcal{R}.
\end{aligned} \tag{8}$$

If $u_i = u_j = 1$ for two rectangles $i \neq j$, i.e. both of the rectangles are packed in the bin, then the first constraints ensure that rectangle i is placed left, right, below, or above rectangle j. If at least one of the variables u_i, u_j is zero, then the first constraints have no effect. The following four constraints ensure a non-overlapping packing within the bin dimensions as stated in (2). The IP model (8) is difficult to solve in practice through general IP solvers as it has the same adverse properties as model (3).

Using the technique from Fekete and Schepers (2001) we solve the pricing problem by splitting it into a one-dimensional optimization problem and a two-dimensional packing-decision problem. The one-dimensional optimization, considered in §5.2, chooses a subset of the rectangles with the largest profit sum subject to a restriction on the available area. The two-dimensional decision problem, considered in §5, answers whether it is possible to arrange a set of rectangles into a single bin.

5. Two-Dimensional Packing-Decision Problem

Martello et al. (2007) showed that constraint programming may be used for solving the decision problem of packing a given set of three-dimensional boxes into a single bin of fixed dimensions. The same approach may be used in two dimensions, answering the question whether a given subset of rectangles can be arranged into a single bin. Where Martello et al. (2007) used CSP for performance reasons, we also use CSP for reasons of flexibility. As seen below, several additional constraints can easily be handled by the CSP framework.

To formulate the decision problem as a CSP we associate with each pair of rectangles $i, j \in \mathcal{R}$ the domain $M_{ij} = \{\texttt{left}, \texttt{right}, \texttt{below}, \texttt{above}\}$ of possible relative placements (*relations*) among which we should choose at least one. To avoid mirror symmetric solutions, rectangles 1, 2 have the restricted domain $M_{12} = \{\texttt{left}, \texttt{below}\}$. This leads to the formulation

$$\begin{aligned}
& r_{ij} \in M_{ij} \quad i, j \in \mathcal{R},\ i \neq j \\
& r_{ij} = \texttt{left} \Rightarrow x_i + w_i \le x_j \quad i, j \in \mathcal{R},\ i \neq j \\
& r_{ij} = \texttt{right} \Rightarrow x_j + w_j \le x_i \quad i, j \in \mathcal{R},\ i \neq j \\
& r_{ij} = \texttt{below} \Rightarrow y_i + h_i \le y_j \quad i, j \in \mathcal{R},\ i \neq j \\
& r_{ij} = \texttt{above} \Rightarrow y_j + h_j \le y_i \quad i, j \in \mathcal{R},\ i \neq j \\
& 0 \le x_i \le W - w_i \quad i \in \mathcal{R} \\
& 0 \le y_i \le H - h_i \quad i \in \mathcal{R}.
\end{aligned} \tag{9}$$

Problem (9) is solved recursively by a specialized algorithm CSP2D. Initially all relations r_{ij} are set to `undefined`. In each iteration two rectangles i and j are considered and a value is assigned to r_{ij} from the associated domain M_{ij}. It is then checked whether a feasible assignment of coordinates exists respecting the current relations r_{ij}. If the coordinate check fails, the algorithm backtracks. Otherwise the algorithm calls itself recursively. If all relations r_{ij} have been assigned a value and the coordinate check was feasible, the CSP2D algorithm terminates with a positive answer.

The coordinate check is performed as follows. Considering the horizontal coordinates first, the rectangles are ordered in topological order according to the partial ordering $r_{ij} = \texttt{left} \Rightarrow i \preceq j$ and $r_{ij} = \texttt{right} \Rightarrow j \preceq i$. Then, running through the rectangles in topological order, the x_j coordinate of rectangle j is calculated as

$$x_j = \max_{i \preceq j} \{x_i + w_i \colon r_{ij} = \texttt{left} \text{ or } r_{ji} = \texttt{right}\}. \tag{10}$$

If $x_j + w_j > W$ for some rectangle j the coordinate check returns false. The same approach is used for the vertical coordinates. Here, the rectangles are ordered according to $r_{ij} = \texttt{below} \Rightarrow i \preceq j$ and $r_{ij} = \texttt{above} \Rightarrow j \preceq i$, and y_j is calculated as

$$y_j = \max_{i \preceq j} \{y_i + h_i \colon r_{ij} = \texttt{below} \text{ or } r_{ji} = \texttt{above}\}. \tag{11}$$

If $y_j + h_j > H$ for some rectangle j then the procedure returns false. The constraint graph has size $O(n^2)$, and the topological ordering can be found in linear time, so the total running time of checking coordinates is $O(n^2)$.

To speed up the solution algorithm further, constraint propagation and domain reduction is applied using the rules

$$\text{if } r_{ij} = v \quad \text{then } r_{ji} = \neg v \quad \forall\, i, j \tag{12}$$

$$\text{if } r_{ij} = v \text{ and } r_{jk} = v \quad \text{then } r_{ik} = v \quad \forall\, i, j, k \tag{13}$$

$$\text{if } r_{ij} = v \text{ and } r_{ki} = v \quad \text{then } r_{kj} = v \quad \forall\, i, j, k \tag{14}$$

where $v \in \{\texttt{left}, \texttt{right}, \texttt{below}, \texttt{above}\}$. The three propagation rules are applied whenever a realtion r_{ij} has been selected, i.e., for a fixed pair of indices i, j. This means that (12) can be evaluated in constant time, and (13)–(14) can be evaluated in $O(n)$ time. All relations and domain reductions obtained by the forward propagation are pushed to a stack, such that the domains quickly can be restored upon backtracking from CSP2D.

The best performance of CSP2D was obtained sorting the rectangles according to non-increasing area, and then considering pairs of rectangles as $(1,2)$, $(1,3), (2,3), (1,4), (2,4), (3,4), (1,5) \ldots$. In this way the largest rectangles were placed relatively to each other at an early stage of the algorithm while the smaller rectangles gradually were added to the problem. This complies with the *first-fail principle* (see e.g. Apt 2003) as we investigate the most difficult alternatives first.

5.1. Handling of Additional Constraints

To solve real two-dimensional packing and cutting problems, a number of additional constraints need to be handled. Several of these constraints are easily handled in the CSP model proposed, without affecting the worst-case size of the search tree or the worst-case time of each iteration.

- **Weight Considerations:** Many two-dimensional bin-packing problems are actually three-dimensional packing problems where the depth of the boxes is fixed and hence can be ignored. To ensure stability of the packing, and to avoid breaking items, it is frequently demanded that heavy boxes are placed below the light boxes. In general we may demand that if box i is heavier than box j then box i should be placed left of, right of, or below box j. This constraint is easily handled by initially setting $M_{ij} = \{\texttt{left}, \texttt{right}, \texttt{below}\}$.
- **Guillotine Patterns:** When cutting material like glass it may be demanded that the rectangles can be cut out of the bin only through guillotine-cuts, where each object is cut end-to-end in parallel to an edge (see Figure 1). As noted in Belov (2003), these constraints are very difficult to model as an IP, and also in practice the problem is difficult to solve (Parada et al. 2000).

Formally, a pattern is *guillotine-cuttable* if it satisfies at least one of the following

—The pattern contains at most three rectangles.

—There exists an x with $0 < x < W$ so that we can separate the set of rectangles into two nonempty sets S_1 and S_2 such that $x_i + w_i \leq x$ for all $i \in S_1$ and $x_i \geq x$ for all $i \in S_2$, and both sets S_1 and S_1 are guillotine-cuttable.

—A symmetric property holds for the y coordinates.

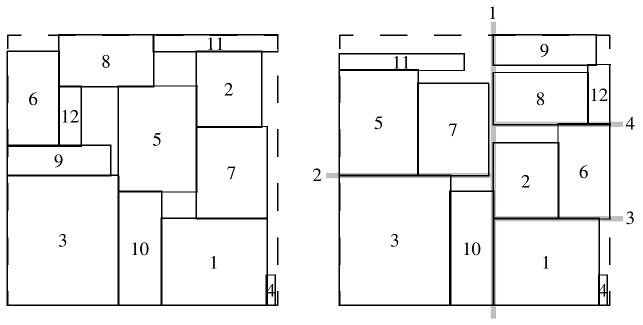

Figure 1 Left: A Non-Guillotine Pattern. Right: A Guillotine Pattern with First Four Cuts Marked

A pattern is *strongly-guillotine-cuttable* if it satisfies the property that we can separate the set S of rectangles into two subsets S_1 and S_2 such that $r_{ij} = \texttt{left}$ for all $i \in S_1$, $j \in S_2$ or such that $r_{ij} = \texttt{below}$ for all $i \in S_1$, $j \in S_2$, and both sets S_1 and S_2 are strongly-guillotine-cuttable.

Note that some patterns may be guillotine-cuttable even if they are not strongly-guillotine-cuttable. However, for every guillotine-cuttable pattern, there exists an equivalent strongly-guillotine-cuttable representation.

Guillotine constraints are easily handled by our framework. In each step of the CSP2D algorithm we check whether the current pattern is strongly-guillotine-cuttable or whether it may become strongly guillotine-cuttable. The test is performed using a *greedy approach*. Find any strong guillotine-cut, separate the problem into two subsets S_1 and S_2 and call the testing algorithm recursively. The greedy approach works since choosing a specific cut will not block any alternative cuts.

The algorithm is implemented as follows. Assume that coordinates of the rectangles have been calculated using (10) and (11). For a *vertical cut*, order the rectangles according to increasing x coordinate. For each distinct x coordinate assigned to the rectangles, partition the boxes into the appropriate two sets S_1 and S_2 and test whether all relations r_{ij}, $i \in S_1$, $j \in S_2$ are the same. If yes, call the algorithm recursively for both sets S_1 and S_2. The scan for a guillotine-cut can be implemented in $O(n^2)$ time by noting that only one rectangle will be moved from set S_2 to S_1 in each step. A *horizontal cut* is implemented in a symmetric way. Since there will be at most $O(n)$ guillotine-cuts in total, and no backtracking is necessary, the whole test runs in $O(n^3)$ time.

- ***n*-Stage Guillotine Patterns:** Technological limitations may demand that the guillotine-cuts are performed in at most n stages, alternating the direction of the cuts in each stage. Gilmore and Gomory (1965) presented an elegant algorithm for the two-stage guillotine-cutting problem based on dynamic programming, which unfortunately is difficult to generalize to more stages.

To handle this constraint we maintain a counter for each subset S, saying how many times the direction has been changed before reaching the current subset. The greedy approach for choosing guillotine-cuts also works for this variant of the problem, provided that the same direction as used at the father node is investigated before alternating direction.

• **Fixed Rectangles:** If the material of the bin is irregular, or if the rectangles need to be connected to specific places, as in VLSI design, one may demand that a rectangle j has a fixed position (x'_j, y'_j). This constraint is handled as follows: the moment we determine the x coordinate of rectangle j using (10) we check whether $x_j > x'_j$, and return false in this case. Otherwise we set $x_j := x'_j$ and continue evaluating (10) for the succeeding rectangles. A similar approach is used for the y coordinates.

• **Irregularities in the Bins:** One or more of the bins available may have some part of the area defective. This is easily handled by placing a dummy rectangle with fixed position at the defective position. If more than one bin is defective, additional constraints should be added to the model such that the two dummy rectangles are not placed in the same bin. If more than one area of the bin is defective, an additional constraint is added to ensure that the corresponding dummy rectangles are placed in the same bin.

• **Border Positions:** It may be demanded that a given rectangle j be placed at the border of a bin without specifying the actual position. The application may demand that the rectangle is easily accessible, or that it should interact outwardly (e.g., in VLSI design where some modules must be wired to the surroundings). This is handled by the additional constraint

$$\begin{gathered}\forall\, i \in \mathcal{R}\colon\ r_{ij} \neq \texttt{left} \quad \text{or}\\ \forall\, i \in \mathcal{R}\colon\ r_{ij} \neq \texttt{right} \quad \text{or}\\ \forall\, i \in \mathcal{R}\colon\ r_{ij} \neq \texttt{below} \quad \text{or}\\ \forall\, i \in \mathcal{R}\colon\ r_{ij} \neq \texttt{above}\end{gathered}$$

the constraint is checked in each iteration using time $O(n)$ for each rectangle j.

5.2. Solving the Pricing Problem Through CSP and Branch-and-Cut

As mentioned in §4 we split the pricing problem (8) into a one-dimensional optimization problem and a two-dimensional packing-decision problem. The one-dimensional optimization chooses a subset of the rectangles with the largest profit sum subject to a restriction on the available area. This model is recognized as a one-dimensional 0-1 knapsack problem (1DKPP) of the form

$$\max\left\{\sum_{i\in\mathcal{R}} p_i x_i \,\middle|\, \sum_{i\in\mathcal{R}} w_i h_i x_i \le WH;\ x_i \in \{0,1\},\ i\in\mathcal{R}\right\}. \tag{15}$$

Let $D = \{i \in \mathcal{R}\colon x_i = 1\}$ be those rectangles selected in (15). The two-dimensional packing problem now attempts to arrange the rectangles D into a bin of size $W \times H$. This decision problem is solved through the CSP2D algorithm.

If the decision problem cannot be satisfied, we may add the inequality

$$\sum_{i\in D} x_i \le |D| - 1, \tag{16}$$

to the knapsack problem (15). The inequality cuts off the present solution to (15) and it can easily be shown that repeating the above process will lead to an integer optimal solution to the pricing problem in the end. The set of all generated valid inequalities is denoted $\mathscr{C}$. In this way we get the one-dimensional multi-constrained knapsack problem (1DMCKP) of the form

$$\begin{aligned} \max\ & \sum_{i\in\mathcal{R}} p_i x_i\\ \text{s.t.}\ & \sum_{i\in\mathcal{R}} w_i h_i x_i \le WH\\ & \sum_{i\in C} x_i \le |C| - 1 \quad C \in \mathscr{C}\\ & x_i \in \{0,1\} \quad i \in \mathcal{R}. \end{aligned} \tag{17}$$

This problem may be solved through a general MIP solver but computational experiments showed that this was inefficient when the number of constraints $|\mathscr{C}|$ became very large. Hence a specialized branch-and-bound algorithm was developed, which sorts the items according to non-increasing efficiencies $p_i/(w_i h_i)$, and repeatedly branches on the most efficient free variable x_i. Upper bounds are derived from the LP relaxation of (15) in which both the integrality constraints and the cardinality constraints have been relaxed. Backtracking occurs whenever the upper bound does not exceed the current incumbent solution, or when some of the constraints are violated.

Unfortunately, the branch-and-cut process may converge very slowly if the generated inequalities in $\mathscr{C}$ are weak. Hence, instead of adding inequalities of the form (16) whenever the rectangles D do not fit into the bin, we would like to identify the smallest subset $D' \subseteq D$ that does not fit into the bin, giving rise to the tighter inequality $\sum_{i\in D'} x_i \le |D'| - 1$. The latter separation problem may be solved through a modification of the CSP algorithm to avoid aggressive forward propagation. The disadvantage of forward propagation is that information from all rectangles $i \in D$ is used to deduce that the rectangles do not fit into a single bin, so leaving no information about the "core of the problem" D'. If, on the other hand, we used the CSP algorithm without forward propagation, we could register the maximum depth $\hat{d}$ of the search tree, so if CSP2D returns false, we know that rectangles $D' = \{1, 2, \ldots, \hat{d}\}$ do not fit into a single bin.

Omitting the forward propagation also makes the search less efficient, so a compromise was developed, using *limited-depth forward propagation*. Initially, no propagation is made, but as the CSP search develops we register the current maximum depth $\hat{d}$ of the search tree. At any node of the CSP algorithm, we only use forward propagation involving rectangles $\{1, \ldots, \hat{d}\}$. In this way, when the algorithm terminates with a negative answer, a subset $D' = \{1, 2, \ldots, \hat{d}\}$ of rectangles has been identified that do not fit into the bin.

To improve further the efficiency of the above approach, we initially sort the rectangles according to decreasing area. Then the above CSP algorithm is run, returning the set $D' = \{1, 2, \ldots, \hat{d}\}$. D' is minimal in the sense that removing the last rectangle will leave a subset of rectangles that fit into a single bin. Still, a subset of D' may have the property that they do not fit into a single bin, so we prune D' by repeatedly testing whether $D' \backslash \{i\}$ still cannot be packed. The tests are performed for $i = 1, \ldots, \hat{d} - 1$ in a recursive way, until a minimal subset has been identified.

5.3. Adding Multiple Cuts

Whenever the CSP2D algorithm returns a negative result we obtain a valid inequality of the form (16), which says that the rectangles in D cannot be packed in the same bin. By lifting the inequality we can derive additional valid constraints that tighten the 1DKPP further without solving the computationally expensive CSP2D.

Assuming that a call to CSP2D has produced a valid inequality $\sum_{i \in C} x_i \leq |C| - 1$, we may add another valid inequality for every substitution of a rectangle $i \in C$ with a larger or equal rectangle j (i.e. $w_j \geq w_i$ and $h_j \geq h_i$), $j \in \mathcal{R} \backslash C$. If a rectangle j is larger than or equal to all rectangles $i \in C$ we can add the valid inequality $\sum_{i \in C} x_i + x_j \leq |C| - 1$.

As an example consider the instance in Figure 2. If we call CSP2D on the rectangles in the figure, we get a negative answer and the equation $x_1 + x_2 + x_3 \leq 2$ is returned saying that rectangles 1, 2, and 3 cannot be packed in a single bin. But since rectangle 4 is larger than or equal to rectangle 3 (i.e. $w_4 \geq w_3$ and $h_4 \geq h_3$), clearly rectangles 1, 2, and 4 cannot be packed in the same bin either, revealing a new valid inequality $x_1 + x_2 + x_4 \leq 2$, which may be added to the 1DMCKP.

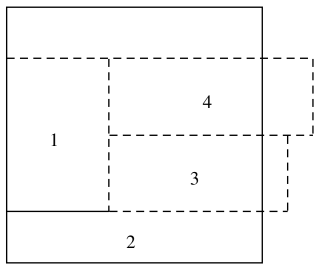

Figure 2 **Adding Multiple Cuts for 1DMKP**

5.4. Dominance

Whenever we add a valid inequality, it could be dominated by an existing valid inequality, or it could dominate one or more of the existing valid inequalities. For example, assume we have added the inequality $x_1 + x_6 \leq 1$ to the set of valid inequalities and we find out that $x_1 + x_6 + x_9 \leq 1$ is also a valid inequality. Clearly the second inequality dominates the first and we can therefore delete the first inequality. The dominance criterion for inequalities is stated below. To keep the set of valid inequalities as small as possible, we delete all dominated inequalities by this criterion.

Theorem 1 (Dominance). *An inequality X:*

$$\sum_{i \in \mathcal{I}} x_i \leq d_X, \quad \forall\, i \in \mathcal{I}\colon x_i \in \{0, 1\}$$

dominates another inequality Y:

$$\sum_{k \in \mathcal{K}} x_k \leq d_Y, \quad \forall\, k \in \mathcal{K}\colon x_k \in \{0, 1\}$$

iff $d_X \leq d_Y - |\mathcal{K} \backslash (\mathcal{I} \cap \mathcal{K})|$.

Proof. Let $\mathcal{M} = \mathcal{I} \cap \mathcal{K}$. Assume $d_X \leq d_Y - |\mathcal{K} \backslash \mathcal{M}|$. We want to prove that if inequality X is satisfied then inequality Y is also satisfied. Assume inequality X is satisfied, i.e. $\sum_{i \in \mathcal{I}} x_i \leq d_X$. Since $\mathcal{M} \subseteq \mathcal{I}$ we have $\sum_{i \in \mathcal{M}} x_i \leq d_X \Rightarrow \sum_{k \in \mathcal{M}} x_k + \sum_{k \in \mathcal{K} \backslash \mathcal{M}} x_k \leq d_X + |\mathcal{K} \backslash \mathcal{M}| \leq d_Y$, since $\forall\, k \in \mathcal{K}\colon x_k \leq 1$. This proves that inequality Y is also satisfied.

The other implication is proved by contradiction. Assume $d_X > d_Y - |\mathcal{K} \backslash \mathcal{M}|$. We now want to show that we can find an example where inequality X is satisfied without inequality Y being satisfied. Assume that $\mathcal{K} \neq \varnothing$ and that $M = \mathcal{I} \cap \mathcal{K} = \varnothing$. If $d_X = d_Y = 0$ then assumption $d_X > d_Y - |\mathcal{K} \backslash \mathcal{M}|$ holds. But at the point where $\forall\, i \in \mathcal{I}\colon x_i = 0$ and $\forall\, k \in \mathcal{K}\colon x_k = 1$, inequality X is satisfied but inequality Y is not. Thus, inequality X does not dominate inequality Y, which concludes the proof by contradiction. □

6. A Heuristic Pricing Algorithm

The 2DKP described in the previous section is $\mathcal{NP}$-hard, so to speed up the column-generation the pricing problem is solved heuristically whenever possible. We use a heuristic that solves the 2DKP in polynomial time, but without giving a guarantee of finding a packing with lowest reduced cost. We will use this algorithm in every iteration of the column-generation procedure, and only when it fails to find a single bin packing with negative reduced cost will we apply the exact 2DKP algorithm from §4.

The heuristic pricing algorithm is based on the greedy paradigm. It starts with an empty bin and greedily places rectangles in the bin until no more rectangles can be placed. The algorithm keeps track of all the *free-space rectangles* in the bin. The free-space rectangles comprise a collection of *maximal* rectangles f satisfying $f \cap i = \varnothing$ for all $i \in \mathcal{R}$. *Maximal* means that a free-space rectangle is bounded on all sides by rectangles or the bin sides. Notice that the free-space rectangles may overlap and that $\bigcup f = W \times H \setminus \bigcup i$. In every iteration a rectangle is placed in one of the free-space rectangles where it fits and the set of new free-space rectangles is determined for the bin. Two different strategies for selecting a rectangle to place and two different strategies for selecting a free-space have been considered.

6.1. Choosing a Rectangle

The objective of the heuristic pricing algorithm is to determine a two-dimensional packing that minimizes the corresponding reduced cost. Since we are interested only in packings with negative reduced cost, we can discard all rectangles with nonnegative reduced price. A natural, greedy strategy is thus to add the rectangle with lowest reduced price/area ratio in every iteration, since this gives us the lowest reduced cost compared to the area. The above strategy may tend to add smaller rectangles before larger ones, since the small rectangles may often have a better price/area ratio. This may, however, prevent the larger rectangles from being added, since there may not be room for them later on. Another strategy is to disregard the area of the rectangles and simply choose the rectangle with the lowest reduced price in every iteration. Randomized versions of both strategies are also considered. In the randomized versions, the probability of choosing a rectangle is proportional to the reduced price/area ratio, or the reduced price of the rectangle.

6.2. Choosing a Free-Space Rectangle

Having chosen a rectangle i to place into the bin, we need to choose a free-space rectangle to place it in. Two strategies have been considered: (1) *Smallest Rectangle*. Choose the smallest free-space rectangle f that can hold i. (2) *Maximize Largest*. Choose the free-space rectangle f that maximizes the largest free-space rectangle in the bin after placing i in f.

If no free-space rectangle exists that can hold the chosen rectangle i, we discard i and choose another rectangle. When there are no more rectangles left, the heuristic terminates.

Combining the strategies for choosing a rectangle and free-space rectangle we obtain eight variants of the heuristic pricing algorithm. In every step of the column-generation procedure, we successively run one of the eight variants until a single bin-packing with negative reduced cost is found. Only if none of the eight variants produce such a packing is the exact pricing algorithm applied. The randomized heuristic pricing algorithms may be run several times before applying the exact pricing algorithm.

7. Early Termination of Column Generation

Delayed column-generation has the disadvantage that the last pricing problems may have an objective value close to 0. This is known as the tailing-off problem, which results in a huge number of columns being generated without changing the optimal value of the restricted master problem significantly. In general, if the column-generation is terminated before completion, this will result in a non-valid lower bound. Lasdon (1970) presents a bound on the optimal objective value of the full master problem, which can be computed in constant time given the optimal objective value of the current master LP and an optimal solution to the pricing problem.

If m is an upper bound on the optimal number of bins, and z' is the optimal objective value of the current restricted master LP and c^π is the reduced cost of the minimum reduced-cost column, we get the following lower bound on the optimal objective value of (5):

$$a = z' + mc^\pi. \tag{18}$$

If $\lceil a \rceil = m$ we may terminate the column-generation, knowing that the present upper bound equals the lower bound. The objective value of the restricted master problem is available in every iteration, but the minimum reduced cost is available only when we solve the pricing problem with the exact algorithm. Hence the lower bound can only be computed in iterations where the exact pricing algorithm is used.

8. The Primal Bound

Column-generation provides fairly tight lower bounds for the optimal solution to 2DBPP. To bound the optimal solution from above we start out by running a heuristic algorithm that finds very high-quality solutions. Although the column-generation in principle can start with the identity matrix as initial columns, our computational experiments showed that the computational effort could be decreased considerably by feeding the algorithm with a near-optimal set of columns. Hence a state-of-the-art heuristic from Faroe et al. (2003) was chosen to provide the initial columns as well as an initial upper bound.

The heuristic, based on guided local search (GLS) as described in Voudouris (1997); Voudouris and Tsang (1999), attempt to pack all the rectangles into a fixed

Table 1 Computational Results for 2DBPP

Class[1]	No. of items[2]	No. of problems solved optimally[3]	Avg. CPU (seconds)[4]	Avg. no. of col. gen.[5]	Avg. no. of cuts gen.[6]	Avg. no. of B&B nodes[7]	Avg. CPU low. bound (seconds)[8]	No. of cols. gen. GLS + slack[9]	No. of cols. gen. det. heur.[10]	No. of cols. gen. rand. heur.[11]	No. of cols. gen. 1DMCKP + CSP2D[12]	No. of opt. cols. GLS[13]	No. of opt. cols. heuristic[14]	No. of opt. cols. 1DMCKP + CSP2D[15]
I	20	10	30.4	41.8	65.5	12.4	0.1	27.2	7.4	0.1	7.1	1.3	1	0.1
I	40	10	45.2	111.3	474.4	5	0.5	53.5	37.6	7.3	12.9	0.8	1.1	0.4
I	60	10	202.2	144.7	2,345	2.8	171.1	80.2	47.3	15.7	1.5	1.8	4.3	0
I	80	10	20.7	124.2	1,390.6	1.2	7.2	107.6	13.4	3.2	0	2.1	0.7	0
I	100	10	119.9	491.8	2,310.5	1.2	55.0	132.4	283.2	76.2	0	4.2	13.9	0
II	20	10	0.0	21	0	1	0.0	21	0	0	0	0	0	0
II	40	10	161.0	217.2	2.3	1	151.0	42	146.6	28.5	0.1	0	0.2	0
II	60	10	0.6	62.5	0	1	0.0	62.5	0	0	0	0	0	0
II	80	9	361.4	301.6	704	1.2	353.3	83.2	154	64.4	0	0.4	0	0
II	100	10	0.3	97.8	0	1	0.0	97.8	0	0	0	0	0	0
III	20	10	39.9	47.4	47.7	1	0.2	25.1	18.6	3.7	0	0.1	0.6	0
III	40	10	306.0	151.3	1,359.3	4.6	197.2	49.4	57.8	33.2	10.9	0.8	0.5	0.2
III	60	10	517.6	282	2,847.6	1	467.4	74	139.6	67.3	1.1	0.2	3.7	0
III	80	10	394.7	353.7	2,456.1	1	343.5	99.1	192.1	61.4	1.1	0.8	6.4	0
III	100	9	481.8	853.1	2,874.9	1	406.9	122.7	549.4	181	0	4.1	11	0
IV	20	10	0.0	21	0	1	0.0	21	0	0	0	0	0	0
IV	40	10	0.1	41.9	0	1	0.0	41.9	0	0	0	0	0	0
IV	60	8	720.2	388.5	968.3	1	705.1	62.5	208.5	117.5	0	0	0.6	0
IV	80	7	1,080.1	778.5	2,100.6	1.2	1,065.3	83.3	491.9	203.3	0	0.4	0.8	0
IV	100	9	365.5	462.8	510.8	1	352.8	103.8	253.2	105.8	0	0.4	0	0
V	20	10	50.0	38.9	50.1	1	0.1	26.5	11.1	1.3	0	0.2	0.4	0
V	40	10	37.5	114.1	609.3	1	7.4	51.9	47	14.9	0.3	1.5	2.2	0
V	60	9	402.9	205.5	2,005.9	1	351.0	78.1	99.1	28.1	0.2	2.2	2.6	0
V	80	8	801.6	309	6,028.9	1	729.1	104.7	146.2	57.8	0.3	2.8	4.5	0.2
V	100	7	1,614.5	698.1	10,930.6	8.6	1,191.0	128.6	382.7	181.5	5.3	2.6	15.5	0.4
VI	20	10	0.0	21	0	1	0.0	21	0	0	0	0	0	0
VI	40	10	968.2	503.1	9.7	1	932.4	41.8	382.9	78	0.4	0	0.6	0
VI	60	9	360.4	219.6	400.3	1.2	351.6	62.2	95.8	61.6	0	0	0.3	0
VI	80	10	0.5	83	0	1	0.0	83	0	0	0	0	0	0
VI	100	8	723.0	723.5	1,075.2	1	719.6	103.4	428.5	191.6	0	0.8	0	0
VII	20	10	20.5	36.9	120.2	1	0.6	25.5	7	4.4	0	0	0.6	0
VII	40	8	1,099.6	130.7	2,832.4	2.2	1,051.6	51.3	41.7	36.5	1.2	0.4	2.9	0.2
VII	60	7	1,460.8	297.9	6,910.4	1.4	1,402.1	76.1	125.3	96.5	0	3.3	6.4	0
VII	80	2	2,892.3	493.4	15,263.1	1.4	2,807.3	103.3	198.3	191.8	0	5.5	12.7	0
VII	100	7	1,119.3	427.1	4,588.2	1.2	1,064.7	127.5	181.5	118.1	0	4.9	11.5	0
VIII	20	10	30.2	36.6	74.1	1	0.3	25.8	7.2	3.6	0	0.1	0.4	0
VIII	40	9	370.7	79.9	773.1	1	350.2	51.4	16.2	12.3	0	0.6	1.9	0
VIII	60	7	1,090.8	198.4	5,549.3	1.4	1,053.5	76.3	55.4	66.7	0	1	5.7	0
VIII	80	9	387.2	219.7	2,585.9	1	352.0	102.6	75.6	41.5	0	1.1	3.4	0
VIII	100	6	1,825.4	471.7	12,339.8	1.4	1,758.6	128.1	194.7	148.9	0	5.4	11.9	0
IX	20	10	0.0	34.3	128.7	1	0.0	34.3	0	0	0	0	0	0
IX	40	10	30.4	70	529.7	1	0.2	67.8	1.9	0.3	0	0	0	0
IX	60	10	21.6	105.1	1,258.2	1	0.4	103.7	1.3	0.1	0	0	0	0
IX	80	10	34.5	140.4	2,204	1	1.6	137.7	2.7	0	0	0	0	0
IX	100	10	30.1	173	3,360.7	1.4	2.3	169.5	3.1	0.2	0.2	0	0	0
X	20	10	46.4	61.3	63.6	1	22.9	24.2	18.3	18.6	0.2	0	0.3	0
X	40	9	381.2	177.6	637	1	352.3	47.4	92.7	37.4	0.1	0	1.4	0
X	60	7	1,104.4	543.6	3,172.5	1.2	1,055.4	70.2	318.4	155	0	0.5	4.8	0
X	80	3	2,521.0	1,166	11,097.2	1.6	2,454.6	93	725.9	347.1	0	0.2	9.3	0
X	100	2	2,888.6	1,798.5	13,928.1	1.6	2,810.8	116.1	1,166.1	516.3	0	5	8.2	0
Avg.		8.8	543.2	291.4	2,579.7	1.7	502.0	74.5	148.5	67.6	0.9	1.1	3.0	0.0

[1]Problem class; [2]no. of items; [3]no. of problems solved to optimality; [4]avg. CPU time used; [5]avg. no. of columns generated; [6]avg. no. of cuts generated; [7]avg. no. of branch-and-bound nodes; [8]avg. CPU time used for computing the lower bound in the root node; [9]avg. no. of columns generated by the initial GLS algorithm + slack columns; [10]no. of columns generated by the deterministic heuristic pricing algorithm; [11]no. of columns generated by the randomized heuristic pricing algorithm; [12]no. of columns generated by the exact pricing algorithm; [13]no. of columns in the optimal solution generated by the initial guided local search algorithm; [14]no. of columns in the optimal solution generated by the heuristic pricing algorithm; [15]no. of columns in the optimal solution generated by the exact pricing algorithm.

number m of bins. A feasible packing will have no overlap between rectangles, so the objective function minimizes the sum of overlaps between each pair of rectangles. The neighborhood of the local search algorithm is based on selecting a rectangle and sliding it either horizontally, vertically or moving it to another bin. The GLS framework is based on a "hill-climbing" approach that repeatedly performs an improving local search iteration until the algorithm is trapped in a local minimum. GLS then selects a pair of rectangles with the overlap feature, assigns a punishment to this feature, and re-optimizes the problem with respect to a modified objective function. To minimize the number of bins used, GLS is called for decreasing values of m until a feasible packing cannot be found within a given number of iterations.

Table 2 Summarized Results for Different Problem Sizes

Size	No. of problems solved optimally	Avg. CPU (seconds)	Avg. no. of col. gen.	Avg. no. of cuts gen.	Avg. no. of B&B nodes	Avg. CPU low. bound (seconds)	No. of cols. gen. GLS + slack	No. of cols. gen. det. heur.	No. of cols. gen. rand. heur.	No. of cols. gen. 1DMCKP + CSP2D	No. of opt. cols. GLS	No. of opt. cols. heuristic	No. of opt. cols. 1DMCKP + CSP2D
20	100	22	36	55	2.1	2	25.2	7.0	3.2	0.7	0.2	0.3	0.0
40	96	340	160	723	1.9	304	49.8	82.4	24.8	2.6	0.4	1.1	0.1
60	87	588	245	2,546	1.3	556	74.6	109.1	60.9	0.3	0.9	2.8	0.0
80	78	849	397	4,383	1.2	811	99.8	200.0	97.1	0.1	1.3	3.8	0.0
100	78	917	620	5,192	1.9	836	123.0	344.2	152.0	0.6	2.7	7.2	0.0

Note. See Table 1 for column definitions.

During the execution of the branch-and-price algorithm we may improve the primal bound. During column-generation numerous instances of the restricted master problem (6) are solved to LP-optimality. Any IP solution to this problem provides a valid solution for the 2DBPP and may be used for tightening the upper bound. The IP variant of (6) may be solved by CPLEX (ILOG 2000), but since it is quite time consuming, we only solve the problem to integer optimality every M_3 iterations. Experimentally we found that $M_3 = 50$ works well.

9. Branching

We have shown how to compute a lower bound for the 2DBPP using LP column-generation and solving the pricing problem with constraint programming. The lower bound is the optimal solution to the LP relaxation of the 2DBPP. The optimal solution to the LP may incidently have all integer values, in which case it is an optimal solution to our IP. Otherwise, an optimal IP solution must be determined through branch and bound.

In order to obtain optimal integral solutions we apply a branching rule that excludes fractional solutions. As always, when designing a branch-and-bound algorithm, we want a branching scheme that divides the solution space fairly evenly to ensure progress in the algorithm. But when designing a branching scheme for a branch-and-price algorithm, we also have to make sure that the branching scheme works well with the pricing algorithm.

We use Ryan-Foster branching (Ryan and Foster 1981): choose two rectangles i and j, $i \neq j$. Divide the solution space into two branches. On one branch we demand that rectangles i and j may not be in the same bin, and on the other branch we demand that i and j have to be in the same bin. By applying the branching rule we get two new subproblems in the branch-and-bound tree. The first branch corresponds to adding the constraint $\sum_{p\in\mathcal{P}} x_p \delta_i^p \delta_j^p = 0$ to the model, and the second branch corresponds to adding the two constraints $\sum_{p\in\mathcal{P}} x_p \delta_i^p (1-\delta_j^p) = 0$ and $\sum_{p\in\mathcal{P}} x_p (1-\delta_i^p)\delta_j^p = 0$ to the model.

When solving the pricing problem in a subproblem of the branch-and-bound tree, the packings produced by the pricing algorithm must comply with all the branching constraints of the subproblem. As described in §4, the pricing algorithm consists of relaxing the 2DKP to a 1DMCKP and checking whether the associated solution is valid in two dimensions.

If the branching constraint demands that rectangles i and j have to be in the same bin, we simply merge the two rectangles into one rectangle k when solving the 1DMCKP, while the two rectangles are considered as separate when solving the CSP2D problem. If, on the other hand, i and j have to be in different bins, we simply add the inequality $x_i + x_j \leq 1$ to the 1DMCKP. Observe that this inequality is only

Table 3 Summarized Results for Different Problem Classes

Class	No. of problems solved optimally	Avg. CPU (seconds)	Avg. no. of col. gen.	Avg. no. of cuts gen.	Avg. no. of B&B nodes	Avg. CPU low. bound (seconds)	No. of cols. gen. GLS + slack	No. of cols. gen. det. heur.	No. of cols. gen. rand. heur.	No. of cols. gen. 1DMCKP + CSP2D	No. of opt. cols. GLS	No. of opt. cols. heuristic	No. of opt. cols. 1DMCKP + CSP2D
I	50	84	183	1,317	4.5	47	80.2	77.8	20.5	4.3	2.0	4.2	0.1
II	49	78	179	1,304	2.2	47	78.9	76.3	20.5	2.9	1.8	4.0	0.1
III	49	101	200	1,210	1.4	77	76.6	98.1	24.7	0.3	1.6	3.8	0.0
IV	44	60	183	741	1.1	43	73.1	88.6	21.6	0.0	1.3	3.0	0.0
V	44	129	219	603	1.1	112	68.2	116.8	33.8	0.0	0.9	2.8	0.0
VI	47	105	140	141	1.0	101	61.3	60.1	18.6	0.0	0.1	0.0	0.0
VII	34	113	145	151	1.0	101	62.1	63.8	19.3	0.0	0.1	0.2	0.0
VIII	41	142	132	422	1.8	110	63.6	46.1	20.3	2.2	0.3	0.2	0.0
IX	50	245	176	992	1.8	204	65.9	74.0	33.7	2.4	0.3	1.0	0.0
X	31	252	186	1,342	1.7	202	69.1	81.6	33.1	2.6	0.4	2.2	0.0

Note. See Table 1 for column definitions.

Table 4 Comparison of Lower Bounds for 2DBPP

Class	Size	L_0	L_1	L_2	L_4	FS	BM	Col. gen.	UB[1]	Col. gen. guil.[2]	UB guil.[3]
I	20	6.4	5.9	6.7	6.7	6.9	7.0	7.1	7.1	7.1	7.1
I	40	12.0	11.7	12.8	12.8	13.0	13.1	13.4	13.4	13.4	13.4
I	60	18.5	17.8	19.3	19.3	19.7	19.7	20.0	20.0	20.0	20.0
I	80	25.3	24.7	26.9	26.9	27.3	27.4	27.5	27.5	27.5	27.6
I	100	30.5	28.6	31.4	31.4	31.7	31.7	31.7	31.7	31.7	31.9
II	20	1.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
II	40	1.9	0.0	1.9	1.9	1.9	1.9	1.9	1.9	2.0	2.0
II	60	2.5	0.0	2.5	2.5	2.5	2.5	2.5	2.5	2.5	2.6
II	80	3.1	0.0	3.1	3.1	3.1	3.1	3.1	3.2	3.1	3.3
II	100	3.8	0.0	3.8	3.8	3.9	3.9	3.9	3.9	3.9	4.0
III	20	4.4	4.4	4.6	4.6	4.7	4.9	5.1	5.1	5.1	5.1
III	40	8.2	8.2	8.8	8.8	9.2	9.2	9.3	9.3	9.3	9.4
III	60	12.5	12.7	13.3	13.3	13.4	13.6	13.9	13.9	13.9	13.9
III	80	17.3	17.6	18.4	18.4	18.7	18.7	18.9	18.9	18.9	19.0
III	100	20.5	21.0	21.7	21.7	21.9	22.1	22.2	22.3	22.2	22.4
IV	20	1.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
IV	40	1.9	0.0	1.9	1.9	1.9	1.9	1.9	1.9	1.9	1.9
IV	60	2.3	0.0	2.3	2.3	2.3	2.3	2.3	2.5	2.3	2.5
IV	80	3.0	0.0	3.0	3.0	3.0	3.0	3.0	3.3	3.0	3.3
IV	100	3.7	0.0	3.7	3.7	3.7	3.7	3.7	3.8	3.7	3.8
V	20	5.4	5.9	6.0	6.0	6.0	6.5	6.5	6.5	6.5	6.5
V	40	10.1	11.0	11.4	11.4	11.6	11.6	11.9	11.9	11.9	11.9
V	60	15.7	16.7	17.2	17.2	17.7	17.8	17.9	18.0	18.0	18.0
V	80	21.5	23.2	23.6	23.6	24.0	24.1	24.5	24.7	24.6	24.7
V	100	25.9	27.1	27.3	27.3	27.6	27.9	27.9	28.2	27.9	28.2
VI	20	1.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
VI	40	1.5	0.0	1.5	1.5	1.5	1.5	1.7	1.8	1.7	1.9
VI	60	2.1	0.0	2.1	2.1	2.1	2.1	2.1	2.2	2.1	2.2
VI	80	3.0	0.0	3.0	3.0	3.0	3.0	3.0	3.0	3.0	3.0
VI	100	3.2	0.0	3.2	3.2	3.2	3.2	3.2	3.4	3.2	3.4
VII	20	4.7	5.1	5.3	5.3	5.3	5.4	5.5	5.5	5.5	5.5
VII	40	9.7	10.4	10.8	10.8	10.8	10.9	10.9	11.1	10.9	11.1
VII	60	14.0	14.9	15.5	15.5	15.5	15.6	15.5	15.8	15.5	15.8
VII	80	19.7	21.7	22.3	22.3	22.3	22.4	22.4	23.2	22.4	23.2
VII	100	23.8	25.7	26.8	26.8	26.8	26.9	26.9	27.2	26.9	27.2
VIII	20	4.8	5.3	5.5	5.5	5.5	5.8	5.8	5.8	5.8	5.8
VIII	40	9.6	10.2	11.1	11.1	11.1	11.2	11.2	11.3	11.2	11.3
VIII	60	14.1	15.3	15.9	15.9	15.9	15.9	15.9	16.2	15.9	16.1
VIII	80	19.5	21.3	22.2	22.2	22.2	22.3	22.3	22.4	22.3	22.4
VIII	100	24.1	26.5	27.3	27.3	27.3	27.4	27.4	27.8	27.4	27.9
IX	20	9.4	14.3	14.3	14.3	14.3	14.3	14.3	14.3	14.3	14.3
IX	40	18.0	27.4	27.4	27.4	27.5	27.8	27.8	27.8	27.8	27.8
IX	60	27.6	43.3	43.3	43.3	43.5	43.7	43.7	43.7	43.7	43.7
IX	80	37.1	56.9	56.9	56.9	57.4	57.7	57.7	57.7	57.7	57.7
IX	100	45.0	68.9	68.9	68.9	69.3	69.5	69.5	69.5	69.5	69.5
X	20	3.8	3.4	4.0	4.0	4.0	4.0	4.2	4.2	4.2	4.2
X	40	6.9	6.1	7.1	7.1	7.2	7.2	7.3	7.4	7.3	7.4
X	60	9.4	7.8	9.7	9.7	9.7	9.7	9.8	10.1	9.8	10.1
X	80	12.2	9.8	12.3	12.3	12.3	12.3	12.3	13.0	12.3	13.1
X	100	15.3	11.9	15.3	15.3	15.3	15.3	15.3	16.1	15.5	16.5
Sum		597.9	642.7	706.3	706.3	711.7	715.7	718.8	725.0	719.1	726.6

Notes. Lower bounds: L_0, L_1, L_2, L_4 (Martello and Vigo 1998); FS (Fekete and Schepers 2001); BM (Boschetti and Mingozzi 2003a); and the Column-Generation Lower Bound in this paper.

[1]The best-known upper bound; [2]column-generation lower bounds for solutions with guillotine constraints; [3]the best-known upper bound for solutions with guillotine constraints.

Table 5 **Summarized Lower Bound Comparisons for the Ten Problem Classes**

Class	L_0	L_1	L_2	L_4	FS	BM	Col. gen.	UB[1]
I	92.7	88.7	97.1	97.1	98.6	98.9	99.7	99.7
II	12.3	0.0	12.3	12.3	12.4	12.4	12.4	12.5
III	62.9	63.9	66.8	66.8	67.9	68.5	69.4	69.5
IV	11.9	0.0	11.9	11.9	11.9	11.9	11.9	12.5
V	78.6	83.9	85.5	85.5	86.9	87.9	88.7	89.3
VI	10.8	0.0	10.8	10.8	10.8	10.8	11.0	11.4
VII	71.9	77.8	80.7	80.7	80.7	81.2	81.2	82.8
VIII	72.1	78.6	82.0	82.0	82.0	82.6	82.6	83.5
IX	137.1	210.8	210.8	210.8	212.0	213.0	213.0	213.0
X	47.6	39.0	48.4	48.4	48.5	48.5	48.9	50.8

Notes. Lower bounds: L_0, L_1, L_2, L_4 (Martello, Vigo); FS (Fekete, Schepers); BM (Boschetti, Mingozzi); and the Column-Generation Lower Bound in this paper.

[1]Best-known upper bound.

locally valid, in contrast to the other inequalities of the 1DMCKP, which are globally valid.

To divide the solution space evenly we branch on two rectangles from a bin with high fractional value in the LP solution. Thus, we are given two rectangles that probably should be in the same bin in an optimal solution.

10. Computational Results

We have implemented a branch-and-price algorithm to solve 2DBPP as described above. We used ABACUS ("A Branch-And-CUt System") (Junger and Thienel 2000), which is a collection of C++ classes that significantly reduces the work of implementing branch-and-bound-like algorithms. ABACUS provides an interface to CPLEX 7.0 (ILOG 2000), which we have used to solve the linear programs resulting from the column-generation. To evaluate the quality of the lower bounds obtained by the column-generation procedure, we implemented the lower bounds of Martello and Vigo (1998) and of Fekete and Schepers (2001) and we compare our lower bounds with the bounds of Boschetti and Mingozzi (2003a) as well. All tests were carried out on an Intel Pentium IV-3.0 with 2 GB of memory. A time limit of 3,600 seconds was assigned to each instance, where at most 100 seconds were used for the initial GLS heuristic.

We tested our algorithm on the test instances provided by Berkey and Wang (1987) and Martello and Vigo (1998), consisting of ten classes of problems. In each problem class there are 50 instances: 10 with 20 rectangles, 10 with 40 rectangles, 10 with 60 rectangles, 10 with 80 rectangles, and 10 with 100 rectangles. Problem classes I–VI were proposed by Berkey and Wang (1987), while classes VII–X are due to Martello and Vigo (1998).

Table 6 **Summarized Lower Bound Calculations for the Five Problem Sizes**

Size	L_0	L_1	L_2	L_4	FS	BM	Col. gen.	UB[1]
20	41.9	44.3	49.4	49.4	49.7	50.9	51.5	51.5
40	79.8	85.0	94.7	94.7	95.7	96.3	97.3	97.8
60	118.7	128.5	141.1	141.1	142.3	142.9	143.6	144.9
80	161.7	175.2	191.7	191.7	193.3	194.0	194.7	196.9
100	195.8	209.7	229.4	229.4	230.7	231.6	231.7	233.9

Notes. Lower bounds: L_0, L_1, L_2, L_4 (Martello, Vigo); FS (Fekete, Schepers); BM (Boschetti, Mingozzi); and the Column-Generation Lower Bound in this paper.

[1]Best-known upper bound.

10.1. Results

Table 1 shows the results of our branch-and-price algorithm on the ten problem classes. Each row in the tables is a summary of ten problems of the specified type. Tables 2 and 3 summarize the results for different classes and different problem sizes. Table 2 shows how the run time increases with problem size. Most columns are generated by the heuristic pricing algorithms and only very few are generated by the exact pricing algorithm. Thus, the exact algorithm is primarily used to guarantee exactness of the overall algorithm. The exact pricing algorithm accounts for more than 90% of the CPU time, i.e., guaranteeing optimal solutions is computationally expensive!

From Table 3 we see that the hardest problem classes to solve are VII, VIII, and X, which all contain many medium-sized items. For problems with many small items, the continuous lower bound is quite tight and these problems are therefore relatively easy to solve. For problems with many large items, each bin has room for only a relatively small number of items, which means that the number of different single bin-packings is relatively small. It makes sense that our algorithm solves these types of problem more easily since we explicitly generate single bin-packings to obtain an optimal packing of all items.

The branch-and-bound algorithm by Martello and Vigo (1998) was also able to solve many of the problems with either many small items or many large items. However, the algorithm did not perform well on problems with an even distribution of small, medium, and large items (classes I, III, and V). For these problem classes the present algorithm is performing very well.

In Tables 4, 5, and 6 we show the lower-bound calculations in greater detail. The column-generation lower bounds in the root node are compared to L_0, L_1, L_2, L_4 (Martello and Vigo 1998), *FS* (Fekete and Schepers 2007), and *BM* (Boschetti and Mingozzi 2003a). Bound *BM* dominates *FS* which again domintes L_0 to L_4. In the table we also show the best known upper bounds for comparison. These values are found as the minimum of the bounds found by

Table 7 Computational Results for Guillotine-Cuttable Instances

Class	No. of items	No. of problems solved optimally	Avg. CPU (seconds)	Avg. no. of col. gen.	Avg. no. of cuts gen.	Avg. no. of B&B nodes	Avg. CPU low. bound (seconds)	No. of cols. gen. GLS + slack	No. of cols. gen. det. heur.	No. of cols. gen. rand. heur.	No. of cols. gen. 1DMCKP + CSP2D	No. of opt. cols. GLS	No. of opt. cols. heuristic	No. of opt. cols. 1DMCKP + CSP2D
I	20	10	2.5	93.2	77.1	16.2	0.3	29.9	46.1	5.3	11.9	1.2	5.9	0
I	40	10	77.9	225.2	1,157.7	5.2	67.3	59.2	130.5	26.6	8.9	3.4	9.6	0.4
I	60	10	422.2	376.4	4,564.3	5.2	374.5	90.2	234.2	47.5	4.5	2.7	16.6	0.7
I	80	9	394.0	648.6	3,150	1,290.4	15.0	120.9	257.3	21.9	248.5	3.9	20.8	2.9
I	100	8	936.5	865.7	6,979.9	688.6	584.6	149.2	493.5	89	134	2.2	27.9	1.8
II	20	10	0.8	21	0	1	0.0	21	0	0	0	0	0	0
II	40	10	291.8	145	0.8	1	265.9	42	79.4	23.3	0.3	0	0.2	0
II	60	6	1,460.2	594.7	812.3	1.8	1,424.9	62.9	338.8	193	0	0	1.2	0
II	80	1	3,241.8	1,869	2,903.1	2.8	3,164.1	84	1,194.4	590.6	0	0	3.6	0
II	100	3	2,529.3	1,945.1	2,582.6	2.4	2,430.1	104.6	1,292.5	548	0	1	2.4	0
III	20	10	4.2	109.8	62.1	2.8	0.4	26.9	69.4	12.6	0.9	0.9	4	0.2
III	40	9	383.6	325.9	1,899.5	6.4	72.2	53.7	194.3	69	8.9	0.4	8	1
III	60	9	699.2	724.8	4,458.6	287.8	318.0	81.6	333.7	75.4	234.1	0.8	11.3	1.9
III	80	9	584.9	690.7	4,673.5	7.2	183.6	110.8	469.3	103.6	7	0.2	17.8	1
III	100	7	1,173.0	1,161.6	7,138	4.2	777.2	136	799.1	224.9	1.6	0.6	21.5	0.4
IV	20	10	0.0	21	0	1	0.0	21	0	0	0	0	0	0
IV	40	10	55.4	155.4	0	1	0.9	42	107.9	5.5	0	0	0.1	0
IV	60	4	2,198.8	942.7	988.4	1.8	2,068.2	62.9	546.8	333	0	0	1.8	0
IV	80	0	3,600.0	2,242.5	2,620.5	2.8	3,382.9	84	1,447.4	711.1	0	0.8	3.2	0
IV	100	3	2,557.4	2,217.4	1,633	2.2	2,294.7	104.4	1,453.1	659.9	0	1.8	1.4	0
V	20	10	1.3	78.2	58.3	2.4	0.2	28.9	45.2	3.2	0.9	1.6	4.1	0.3
V	40	10	14.4	233.4	690.9	4	6.0	57.3	145.7	25.5	4.9	1.8	9.6	0.5
V	60	10	343.7	398.2	3,245.9	10	239.6	87.8	258.8	37.3	14.3	1.8	15.2	1
V	80	9	745.3	486.6	6,113.9	1.6	730.7	117.2	312.4	56.9	0.1	1.8	22.9	0
V	100	7	1,295.5	955.5	9,896.8	7.6	807.3	144.5	633.8	170.3	6.9	1.7	25.7	0.8
VI	20	10	0.1	21	0	1	0.0	21	0	0	0	0	0	0
VI	40	8	1,271.8	536.6	7.8	1	1,208.2	41.9	387.1	104.6	3	0	0.8	0
VI	60	6	1,813.7	960.2	551.9	1.6	1,630.7	52.5	649.4	258.2	0.1	0	1.7	0
VI	80	6	1,499.0	964.7	855.1	1.8	1,343.2	83.4	571.6	309.7	0	0	1.6	0
VI	100	0	3,600.0	3,054.6	1,577.4	2.4	3,210.4	104.2	2,041.2	909.2	0	1.6	2.6	0
VII	20	10	0.8	86	136.6	2.2	0.7	27.5	44.9	11.7	1.9	0.1	5.2	0.2
VII	40	8	1,080.8	240.6	2,781.5	1.4	1,080.6	56	119.3	65.3	0	0.7	10.4	0
VII	60	7	1,083.0	416.2	4,774.6	1.2	1,081.2	84.3	203.7	128.2	0	0.5	15.3	0
VII	80	2	2,881.3	604.7	15,699.1	1.4	2,880.9	114.5	281.3	208.9	0	0.5	22.7	0
VII	100	7	1,102.3	690	4,503.5	1.2	1,097.1	140.6	374.3	175.1	0	0	27.2	0
VIII	20	10	13.9	78.8	349.4	1.8	13.5	27	30.9	20.9	0	0.4	4.4	0
VIII	40	9	721.8	211.5	1,664.2	1.2	720.2	56.1	112.8	42.6	0	0.5	10.8	0
VIII	60	8	1,443.6	379.9	7,202.9	1.6	1,441.4	82.6	182.8	114.5	0	0.3	15.8	0
VIII	80	9	369.7	479.4	2,597.1	1	364.3	112.8	271.2	95.4	0	0.3	22.1	0
VIII	100	5	1,819.0	711.8	13,050.5	1.4	1,811.6	139.2	361.4	211.2	0	0.8	27.1	0
IX	20	10	0.1	39.2	128.7	1.6	0.1	35.6	3.6	0	0	5.9	1.8	0
IX	40	10	0.6	90.3	529.7	1.6	0.5	71.8	17.2	1.3	0	22.7	5.1	0
IX	60	10	2.2	127.6	1,258.2	2.4	2.0	108.2	18.1	1.3	0	38.1	5.6	0
IX	80	10	5.3	171.6	2,204	2	5.1	143.6	26.2	1.8	0	49.6	8.1	0
IX	100	10	16.0	225.5	3,464.7	3.6	10.8	179.3	43.9	1.5	0.8	58.4	11	0.1
X	20	10	8.5	115.1	64.2	3	3.2	25.4	62.4	25.5	1.8	0.3	3.3	0.2
X	40	9	383.0	426.6	960.2	1.2	360.0	50.7	294.1	81.7	0.1	0.1	7.3	0
X	60	7	1,139.1	927.5	3,808.3	1.2	1,079.6	74.6	614.1	238.8	0	0	10.1	0
X	80	2	2,921.7	1,626.4	12,621.3	1.8	2,831.6	99.6	1,101.2	425.6	0	0	13.1	0
X	100	0	3,600.0	2,236.6	17,622	2.4	3,483.7	124.7	1,497.6	614.3	0	0	16.5	0
Avg.		7.5	995.8	659.0	3,282.4	48.0	897.4	79.6	403.9	161.6	13.9	4.2	9.7	0.3

Note. See Table 1 for column definitions.

Martello and Vigo (1998), and by our approach. The column-generation lower bounds are greater than or equal to *BM* in all cases, but the improvement varies with the problem classes. For problem classes II, IV, VII, VIII, and IX the two lower bounds are very similar. The largest improvement of the *FS* lower bound is for classes I, III, and V with an evenly distributed amount of small, medium, and large items. It seems reasonable that the column-generation lower bound has the greatest advantage over *BM* for these problem classes since neither L_0 nor bounding by considering the large items yields a tight bound in this case.

From the summary of the lower bounds presented in the bottom line of Table 4 we get a rough overview of the quality of the different lower bounds. The column-generation lower bound is on average 33% closer to *UB* than the previously best lower bound *BM*. The improvement of the column-generation lower bound varies considerably with the type of problems. Since the column-generation lower bound is able to handle additional constraints added by the branch-and-bound process, the integrality gap will decrease during the branching.

10.2. Guillotine Constraints

To show flexibility of the CSP framework in combination with column-generation, we implemented the guillotine-cutting constraints from §5.1. The guided local search heuristic, which is used to find an initial upper bound, cannot be easily modified to allow guillotine constraints. Hence, we used a simpler initial heuristic, which does not find a solution of as high quality as the guided local search heuristic. The heuristics for the pricing problem were easily adapted to handle guillotine constraints, by checking the cutting constraint in each step of the greedy algorithm.

We tested our algorithm with guillotine-cutting constraints on the same 500 test instances as for the normal 2DBPP and Table 7 shows the results. Furthermore, the column-generation lower bound and the best known upper bound for guillotine-cuttable solutions are reported in the last two columns of Table 5.

Table 7 shows that in total, 75% of all instances were solved for instances with up to 100 items. Without the guillotine constraint, 88% of the 2DBPP instances were solved. This may be due to the simpler initial heuristic, which does not provide as tight an upper bound for the guillotine-cuttable instances as for the normal instances. Table 5 shows that the additional guillotine constraints only affect the number of bins in the optimal solution a little bit, as 725.0 bins are used on average for the normal 2DBPP instances whereas 726.6 bins are used on average for the guillotine-cuttable instances. However, the placement of the items in the bins in the guillotine-cuttable instances may be very different from the normal 2DBPP instances, as in Figure 1. As shown in Table 5, our lower bound based in column-generation and CSP is able to handle the guillotine-cutting constraints to provide a tighter lower bound, whereas the guillotine constraints cannot easily be incorporated in the lower bounds previously presented in literature.

The constraint-satisfaction problem is easier to solve with guillotine constraints, since the solution space is smaller. However, the 1D relaxation of the multi-constrained knapsack problem provides a worse bound, since the guillotine constraints are not included in the 1D relaxation. Thus, the pricing algorithm may require the solution of many more CSPs before a feasible guillotine-cuttable packing has been found. In practice, this makes the pricing problem harder to solve when guillotine-cutting constraints are included.

10.3. Conclusion

The present paper is, to the best of our knowledge, the first to report results on large instances using column-generation for the 2DBPP. The pricing problem in two dimensions is much more difficult to solve than in one dimension, so we propose a combination of constraint programming and branch-and-cut for solving the resulting 2DKP. The computational results show that the lower bounds obtained through column-generation are tighter than any bounds previously published. Due to these tight lower bounds we are able to solve quite large 2DBPP instances to optimality.

A second contribution of the paper is to try out new ways of combining ILP and CSP such that we obtain the flexibility of CSP while improving overall efficiency. Using CSP we are able to model constraints such as guillotine-cutting requirements, weight considerations, fixed positions, etc., which makes the resulting algorithm much more applicable in practice. The proposed framework can easily be extended to 2DBPP with variable bin sizes or variable bin costs. These generalizations are adressed in (Pisinger and Sigurd 2005).

Moreover, we have developed a framework in which additional constraints detected by the CSP subproblem can be brought back to the above problem, tightening the relaxed pricing problem formulation.

Finally, we are the first to report upper and lower bounds for the 2DBPP with guillotine-cutting constraints. Many instances with up to 100 items have been solved to optimality with guillotine-cutting constraints.

References

Apt, K. R. 2003. *Principles of Constraint Programming*. Cambridge University Press, Cambridge, UK.

Belov, G. 2003. Problems, models and algorithms in one- and two-dimensional cutting. Ph.D. thesis, Institute of Numerical Mathematics, Technischen Universität Dresden, Dresden, Germany.

Bengtsson, B. E. 1982. Packing rectangular pieces—A heuristic approach. *Comput. J.* **25** 353–357.

Berkey, J. O., P. Y. Wang. 1987. Two dimensional finite bin packing algorithms. *J. Oper. Res. Soc.* **38** 423–429.

Bockmayr, A., T. Kasper. 1998. Branch-and-infer: A unifying framework for integer and finite domain constraint programming. *INFORMS J. Comput.* **10** 287–300.

Boschetti, M. A., A. Mingozzi. 2003a. The two-dimensional finite bin packing problem. Part I: New lower bounds for the oriented case. *4OR* **1** 27–42.

Boschetti, M. A., A. Mingozzi. 2003b. The two-dimensional finite bin packing problem. Part II: New lower and upper bounds. *4OR* **2** 135–148.

Caprara, A., M. Monaci. 2004. On the two-dimensional knapsack problem. *Oper. Res. Lett.* **32** 5–14.

Chen, C. S., S. M. Lee, Q. S. Shen. 1995. An analytical model for the container loading problem. *Eur. J. Oper. Res.* **80** 68–76.

Dell'Amico, M., S. Martello, D. Vigo. 2002. A lower bound for the non-oriented two-dimensional bin packing problem. *Discrete Appl. Math.* **118** 13–24.

Dowsland, K. 1993. Some experiments with simulated annealing techniques for packing problems. *Eur. J. Oper. Res.* **68** 389–399.

Eremin, A., M. Wallace. 2001. Hybrid Benders decomposition algorithms in constraint logic programming. T. Walsh, ed. *CP 2001, Lecture Notes in Computer Science*, Vol. 2239. Springer, Berlin, Germany, 1–15.

Faroe, O., D. Pisinger, M. Zachariasen. 2003. Guided local search for the three-dimensional bin packing problem. *INFORMS J. Comput.* **15** 267–283.

Fekete, Sandor P., Joerg Schepers. 2001. New classes of lower bounds for the bin packing problem. *Math. Programming* **91** 11–31.

Fekete, Sandor P., Joerg Schepers, Jan C. van der Veen. 2007. An exact algorithm for higher-dimensional orthogonal packing. *Oper. Res.* Forthcoming.

Gilmore, P. C., R. E. Gomory. 1961. A linear programming approach to the cutting stock problem. *Oper. Res.* **9** 849–859.

Gilmore, P. C., R. E. Gomory. 1963. A linear programming approach to the cutting stock problem—Part II. *Oper. Res.* **13** 94–119.

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

Hadjiconstantinou, E., N. Christofides. 1995. An exact algorithm for general, orthogonal, two-dimensional knapsack problems. *Eur. J. Oper. Res.* **83** 39–56.

Hifi, M. 2001. Exact algorithms for large-scale unconstrained two and three staged unconstrained cutting problems. *Comput. Optim. Appl.* **18** 63–88.

ILOG. 2000. *ILOG CPLEX 7.0, Reference Manual*. ILOG SA, Gentilly, France.

Junger, Michael, Stafan Thienel. 2000. The ABACUS system for branch-and-cut-and-price algorithms in integer programming and combinatorial optimization. *Software Practice Experience* **30**(11) 1325–1352.

Junker, U., S. E. Karisch, N. Kohl, B. Vaaben, T. Fahle, M. Sellmann. 1999. Constraint programming based column generation for crew assignment. J. Jaffar, ed. *Proc. CP'99, Lecture Notes in Computer Science*, Vol. 1713. Springer, Berlin, Germany, 261–274.

Kellerer, H., U. Pferschy, D. Pisinger. 2004. *Knapsack Problems*. Springer, Berlin, Germany.

Lasdon, L. S. 1970. *Optimization Theory for Large Systems*. Macmillan, London, UK.

Lodi, A., S. Martello, D. Vigo. 1999. Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems. *INFORMS J. Comput.* **11** 345–357.

Lodi, A., S. Martello, D. Vigo. 2002. Recent advances on two-dimensional bin packing problems. *Discrete Appl. Math.* **123** 379–396.

Martello, S., D. Vigo. 1998. Exact solution of the two-dimensional finite bin packing problem. *Management Sci.* **44** 388–399.

Martello, S., D. Pisinger, D. Vigo, Edgar den Boef, Jan Korst. 2007. Algorithms for general and robot-packable variants of the three-dimensional bin packing problem. *ACM Trans. Math. Software* **33**(1).

Monaci, M., P. Toth 2006. A set-covering based heuristic approach for bin-packing problems. *INFORMS J. Comput.* **18**(1) 71–85.

Onodera, H., Y. Taniguchi, K. Tamaru. 1991. Branch-and-bound placement for building block layout. *Proc. 28th ACM/IEEE Design Automation Conf.*, ACM Press, New York, 433–439.

Padberg, M. 2000. Packing small boxes into a big box. *Math. Methods Oper. Res.* **52** 1–21.

Parada, V., R. Palma, D. Sales, A. Gómes. 2000. A comparative numerical analysis for the guillotine two-dimensional cutting problem. *Ann. Oper. Res.* **96** 245–254.

Pisinger, D., M. M. Sigurd. 2005. The two-dimensional bin packing problem with variable bin sizes and costs. *Discrete Optim.* **2** 154–167.

Ryan, D. M., B. A. Foster. 1981. An integer programming approach to scheduling. A. Wren, ed. *Computer Scheduling of Public Transport Urban Passenger Vehicle and Crew Scheduling*. North-Holland, Amsterdam, The Netherlands, 269–280.

Voudouris, C. 1997. Guided local search for combinatorial optimisation problems. Ph.D. thesis, Department of Computer Science, University of Essex, Colchester, UK.

Voudouris, C., E. Tsang. 1999. Guided local search and its application to the traveling salesman problem. *Eur. J. Oper. Res.* **113** 469–499.