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

David Pisinger and Mikkel Sigurd
Dept. of Computer Science, University of Copenhagen,
Universitetsparken 1, DK-2100 Copenhagen, Denmark

Abstract

The two-dimensional bin packing problem is the problem of orthogonally packing a given set of rectangles into the 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 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), where forward propagation is used to prune inferior arrangements of rectangles. Unsuccessful attempts to pack rectangles into a bin are canalized back to the master model as valid inequalities. Hence, CSP is not only used 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 in the literature. In the computational study, quite large instances are solved to optimality through the developed branch-and-price-and-cut algorithm.

The work can be seen as a practical study on merging techniques from MIP and CSP. Each solution technique has its benefits and drawbacks, but decomposition techniques make it possible to use the techniques where most appropriate.

1 Introduction

During the last two decades mathematical optimization techniques have helped the industry in solving very complex planning problems. In particular two techniques have demonstrated

their advantages: Integer linear programming problems (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-inferior techniques [15], but ILP solvers use bounds from linear programming to prune the search tree while CSP algorithms rely on advanced constraint propagation techniques and graph theoretical results for pruning the search tree.

Despite progress in both fields, several industrial problems cannot be solved by present techniques. This typically includes problems which cannot be formulated efficiently in linear form, or where the number of decision variables are 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 at a shipyard).

The appearance of journals like *Constraints* indicates a growing interest in using CSP techniques for solving combinatorial problems, and several research projects have worked on merging techniques for CSP and ILP [3]. In the present paper we focus on decomposition techniques, where ILP and CSP may be combined in solving the very difficult 2D bin packing problem (2DBPP). ILP is the tool of choice when dealing with large-sized models in linear form, while CSP on the other hand is ideal for modeling and solving complicated restrictions. *Dantzig-Wolfe decomposition* [28] is a recognized technique for splitting a suitable problem into a linear master model and a more complicated subproblem where the two approaches, ILP and CSP, can be used at their best premises. The decomposed model leads to tighter bounds in a branch-and-bound algorithm, and delayed column generation may be used to limit the number of subproblems solved. The resulting *branch-and-price* approach is used in this paper to solve the two-dimensional knapsack problem.

The dual approach, based on the so-called *branch-and-cut* algorithms, may also benefit from combining ILP and CSP. In a classical branch-and-cut algorithm additional, globally valid, constraints are generated during the search. The constraints are redundant to the original model but they tighten the bounds obtained through continuous relaxation. The problem of separating a valid constraint may in our situation be formulated as a CSP problem.

The 2DBPP is the problem of packing 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 fewest bins possible, so that the rectangles do not overlap. The rectangles are required to be parallel to the sides of the bin and they may not be rotated. 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 and are also in practice very difficult to solve [19].

Numerous heuristic algorithms for 2DBPP have been presented during the last decades. Greedy heuristics based on concepts like *next-fit*, *first-fit*, *best-fit*, and their performance ratios are surveyed in Lodi, Martello, Vigo [18]. The first method that may be characterized

as a local search algorithm was given by Bengtson [2]. Starting with a packing of a subset of the pieces (boxes in 3D), the remaining pieces are iteratively packed into the bin having maximum unused space. Metaheuristics based on tabu search were presented in Lodi, Martello, Vigo [17], while Faroe, Pisinger, Zachariasen [8] used the concept of *guided local search*. Dowsland [7] presented a *simulated annealing* algorithm for the strip packing problem in 2D. The algorithm tries to pack the pieces 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. Recently, Monaci [20] 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 the heuristic algorithm proposed in [4]. Both the set covering heuristic and the guided local search algorithm yield very good results.

Exact algorithms for the 2DBPP have recently been presented in Martello and Vigo [19] and Fekete and Schepers [9]. Both papers present several lower bounds on the solution value. Although the bounds from many respects are good — having polynomial running times and providing quite tight bounds — their major disadvantage is that they cannot use information from the branching process to tighten the bounds. This means that the same lower bound is obtained throughout the whole branch-and-bound tree, except if some special auxiliary properties (like the closing of a bin) can be used to strengthen the bound. The column generation lower bound described in this paper does not share this disadvantage which makes it well suited to tackle the challenging problems where the lower bound in the root node of the branch-and-bound tree does not equal an initial upper bound calculated by a heuristic algorithm.

Lower bounds based on column generation were presented in [11, 12] through a straightforward generalization of the techniques developed for the 1DBPP problem. To the best of our knowledge, no practical algorithms using these bounds have been presented. A possible explanation is, that the pricing problem for 1DBPP is a simple knapsack problem for which several good algorithms are available [23], while the pricing problem of 2DBPP becomes a two-dimensional knapsack problem which is much harder to solve due to the geometrical structure. Hadjiconstantinou and Christophides [13] studied the two-dimensional knapsack problem, but were only able to solve instances of moderate size.

In this work we tackle the pricing problem in a new way by splitting the problem into a multi-constrained 1D knapsack problem and a simple decision problem of two-dimensional packing. The latter problem is solved through constraint programming, based on the successful results of Pisinger et.al. [22] for solving three-dimensional packing problems in the decision form.

The paper is organized as follows: In the following section we formulate the problem in integer-linear form using $O(n^2)$ binary variables. Unfortunately the proposed formulation contains several symmetric solutions, and hence bounds from LP-relaxation are generally weak. A tighter formulation based on *Dantzig-Wolfe decomposition* is proposed in Sec-

tion 2.1. The master problem is a set covering problem, solved through *delayed column generation*, while the pricing problem becomes a two-dimensional knapsack problem. In Section 4 it is shown how the pricing problem may be solved through a combination of constraint programming and branch-and-cut. The two-dimensional knapsack problem is split into a multi-constrained 1D knapsack problem selecting the most profitable rectangles to pack, and a two-dimensional packing problem in the decision form. Section 5 describes a heuristic algorithm for solving the pricing problem, which is used to speed up the column generation. Finally, Section 6 describes how an initial feasible solution may be obtained through heuristic methods, and how the primal bound is improved during the search, while section 7 describes the branching strategy. The paper concludes with an extensive computational study in Section 8 where the developed branch-and-price algorithm is tested on a set of randomly generated test instances originating from [16]. The bounds obtained through column generation are tighter than any bounds previously presented in the literature, and hence the developed algorithm is also able to solve moderately large problems to optimality.

2 Problem Formulation

Assume that a set $\mathcal{R} = \{1, \ldots, n\}$ of rectangles are given, rectangle i having width w_i and height h_i. An infinite number of bins are available, each having width W and height H. We use the modeling technique proposed by Onodera e.a. [21] and Chen e.a. [5] to formulate the 2DBPP as an IP model. The following decision variables are used: The binary variable ℓ_{ij} is 1 iff rectangle i is located left to j, and similarly b_{ij} is 1 iff rectangle i is located below j. The binary variable p_{ij} attains the value 1 iff rectangle i is located in a bin preceding the bin holding rectangle j. Finally, (x_i, y_i) are the lower left coordinates of rectangle i, m_i is the bin number where rectangle i is packed, and v is the number of bins used.

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

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

If i is located left to j, i.e. $\ell_{ij} = 1$ then we have that $x_i + w_i \leq x_j$. In general we have the constraints

$$\begin{array}{rcl} \ell_{ij} = 1 & \Rightarrow & x_i + w_i \leq x_j \\ b_{ij} = 1 & \Rightarrow & y_i + h_i \leq y_j \\ p_{ij} = 1 & \Rightarrow & m_i + 1 \leq m_j \end{array} \tag{2}$$

No part of the rectangles may exceed the bin, hence $0 \leq x_i \leq W - w_i$ and $0 \leq y_i \leq H - h_i$. Moreover, no more than v bins may be used in total so $1 \leq m_i \leq v$.

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

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

The model has $3n^2$ binary variables and $3n$ continuous variables. The number of constraints is $7/2n^2 + 6n$. Unfortunately the formulation contains several symmetric solutions hence being very difficult to solve by ordinary MIP-solvers. To break some of the symmetries we may add the additional constraints

$$m_i \leq i \qquad i \in \mathcal{R} \tag{4}$$

demanding that rectangle 1 is placed in the first bin, rectangle 2 is placed in one of the two first bins, etc.

2.1 Set Covering Model

Assuming that $\mathcal{P}$ 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 $r \in \mathcal{R}$ and every feasible packing $p \in \mathcal{P}$ we set $\delta_r^p = 1$ iff packing p contains rectangle r. In this way the set covering formulation of the 2DBPP becomes

$$\text{minimize} \quad \sum_{p \in \mathcal{P}} x_p \tag{5a}$$

$$\text{subject to} \quad \sum_{p \in \mathcal{P}} x_p \delta_r^p \geq 1 \quad \forall r \in \mathcal{R} \tag{5b}$$

$$x_p \in \{0,1\} \quad \forall p \in \mathcal{P}. \tag{5c}$$

The object function minimizes the number of bins used in the solution while the constraints (5b) state that every rectangle must be included in some bin in the solution.

The LP relaxation of the set covering model is

$$\text{minimize} \quad \sum_{p \in \mathcal{P}} x_p \tag{6a}$$

$$\text{subject to} \quad \sum_{p \in \mathcal{P}} x_p \delta_r^p \geq 1 \quad \forall r \in \mathcal{R} \tag{6b}$$

$$x_p \geq 0 \qquad \forall p \in \mathcal{P}. \tag{6c}$$

The set covering model is very simple and the LP relaxation gives a fairly tight lower bound on the IP solution. In particular (6) yields a tighter lower bound than the LP relaxation of the first model proposed (3). This is the case since every solution to (6) is also a solution to the LP relaxation of (3) while the opposite is not true in general.

However, the set covering model contains a variable for every feasible packing of a single bin. In general there exists an exponential number of feasible single bin packings, and even for a small number of rectangles, model (6) would be too big to solve. Fortunately, by using *delayed column generation* we can solve the linear program without considering a majority of the variables explicitly.

3 Delayed Column Generation

In delayed column generation we solve the linear problem (6) for a subset $\mathcal{P}'$ of the feasible packings, gradually adding new packings $p \in \mathcal{P} \setminus \mathcal{P}'$ based on the Dantzig rule for solving the linear problem. The *restricted master problem* (RMP) is

$$\text{minimize} \quad \sum_{p \in \mathcal{P}'} x_p \tag{7a}$$

$$\text{subject to} \quad \sum_{p \in \mathcal{P}'} x_p \delta_r^p \geq 1 \quad \forall r \in \mathcal{R} \tag{7b}$$

$$x_p \geq 0 \qquad \forall p \in \mathcal{P}'. \tag{7c}$$

Here $\mathcal{P}'$ is a small set of variables so that a feasible solution exists for RMP. Initially we choose $\mathcal{P}'$ as those packings found by a greedy heuristic (see Section 6) for the 2DBPP. Since we will only add packings to the RMP in the following iterations, the linear program 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_r, $r \in \mathcal{R}$ be the dual variables of the optimal dual solution to the RMP. The dual linear program of (6) is given by:

$$\text{maximize} \quad \sum_{r \in \mathcal{R}} \pi_r \tag{8a}$$

$$\text{subject to} \quad \sum_{r \in \mathcal{R}} \pi_r \delta_r^p \leq 1 \quad \forall p \in \mathcal{P} \tag{8b}$$

$$\pi_r \geq 0 \qquad \forall r \in \mathcal{R}. \tag{8c}$$

If the dual variables π_r to (6) is a feasible solution to (8) we know from the weak duality theorem that x_p is an optimal solution to (6). Otherwise the dual solution π_r must violate

some constraint of (8). The violation of a constraint of type (8b) corresponding to a packing $p \in \mathcal{P}$ by the dual variables π_r is

$$c_p^\pi = 1 - \sum_{r \in \mathcal{R}} \delta_r^p \pi_r$$

which is called the *reduced cost* of packing p with respect to the dual variables π_r. Clearly, if $c_p^\pi \geq 0$ for all $p \in \mathcal{P}$, π_r is a feasible solution to the dual LP, which means that x_p is an optimal solution to (6).

In every iteration of the column generation procedure we find a feasible packing $p \in \mathcal{P}$ with smallest reduced cost c_p^π. If $c_p^\pi \geq 0$ the LP problem has been solved to optimality. Otherwise we add the packing to the RMP. This is the same as adding the most violated constraint of (8) to the dual LP of the RMP, improving feasibility of (8). Finding a packing with smallest reduced cost is known as the pricing problem of the column generation procedure. An exact and a heuristic algorithm for this problem will be described in Section 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 a finite number of feasible packings, and since packings that are already in the RMP have reduced cost ≥ 0, no packings are added more than once. In practice the column generation procedure adds only a small fraction of the feasible packings before terminating, hence the procedure has a much better average performance than should be expected from the worst-case complexity.

4 Solving the Pricing Problem

The *pricing problem* is the problem of finding 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 2D knapsack problem with respect to the profits p_i, the rectangle sizes w_i and h_i and the knapsack size W and H. The 2DKPP can be modeled using the following variables: u_i is a binary variable having the value 1 iff rectangle i is packed in the bin. The variable $\ell_{ij} = 1$ iff rectangle i is located left to 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 follows:

$$
\begin{array}{lll}
\text{maximize} & \sum_{i=1}^{n} p_i u_i & \\
\text{s.t.} & \ell_{ij} + \ell_{ji} + b_{ij} + b_{ji} + (1 - u_i) + (1 - u_j) \geq 1 & i, j \in \mathcal{R}, i < j \\
& x_i - x_j + W\ell_{ij} \leq W - w_i & i, j \in \mathcal{R} \\
& y_i - y_j + Hb_{ij} \leq H - h_i & i, j \in \mathcal{R} \\
& 0 \leq x_i \leq W - w_i & i \in \mathcal{R} \\
& 0 \leq y_i \leq H - h_i & i \in \mathcal{R} \\
& \ell_{ij}, b_{ij} \in \{0, 1\} & i, j \in \mathcal{R} \\
& u_i \in \{0, 1\} & i \in \mathcal{R}
\end{array}
\tag{9}
$$

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 type of 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 type of constraints have no effect. The following four types of constraints ensure a non-overlapping packing within the bin dimensions as earlier stated in (2). The IP-model (9) is difficult to solve in practice through general IP-solvers as it has the same adverse properties as model (3).

4.1 Constraint Programming

Pisinger et. al. [22] 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.

In order to formulate the problem as a CSP we associate with each pair of rectangles i, j the set $M_{ij} = \{\ell, r, b, a\}$ of possible relative placements (*relations*) among which we should choose at least one. To avoid symmetric solutions, the rectangles 1 and 2 have the restricted domain $M_{12} = \{\ell, b\}$. This leads to the formulation:

$$
\begin{array}{lcll}
|M_{ij}| \geq 1 & & & i, j \in \mathcal{R}, i < j \\
M_{ij} = \{\ell\} & \Rightarrow & x_i + w_i \leq x_j & i, j \in \mathcal{R} \\
M_{ij} = \{r\} & \Rightarrow & x_j + w_j \leq x_i & i, j \in \mathcal{R} \\
M_{ij} = \{b\} & \Rightarrow & y_i + h_i \leq y_j & i, j \in \mathcal{R} \\
M_{ij} = \{a\} & \Rightarrow & y_j + h_j \leq y_i & i, j \in \mathcal{R}
\end{array}
\tag{10}
$$

Problem (10) is solved recursively by an algorithm CSPBIN, where two rectangles i and j are considered in each iteration and one of the values in M_{ij} is selected, removing all other values from the set. The feasibility of the imposed relations is then checked, and if it can be proved that no solution exists, the algorithm backtracks. Otherwise the algorithm calls itself recursively. If all sets have cardinality $|M_{ij}| = 1$ (i.e. the relative placement of all rectangles have been fixed) and a feasible assignment of coordinates to the rectangles exists, the algorithm terminates with a positive answer.

The feasibility of a problem with a subset of the relations imposed, is checked the following way: Considering the horizontal coordinates first, the rectangles are ordered in topological order according to the partial ordering

$$\begin{aligned} M_{ij} = \{\ell\} &\Rightarrow i \preceq j \\ M_{ij} = \{r\} &\Rightarrow j \preceq i \end{aligned}$$

Then, running through the rectangles from left to right, x_j is defined as

$$x_j = \max_{i<j} \Big\{ x_i + w_i : M_{ij} = \{r\} \text{ or } M_{ji} = \{\ell\} \Big\}$$

If $x_j + w_j > W$ for some rectangle j then the relations cannot be satisfied and the procedure returns a false answer. The constraint graph has size $O(n^2)$, and the topological ordering can be found in linear time, hence the total running time is $O(n^2)$. The same approach is used for the vertical coordinates using a topological ordering based on values b and a in M_{ij}.

To further speed up the solution algorithm, domain restriction and constraint propagation is applied. Domain restriction is obtained by the FC (forward check) approach proposed by [14], which means that each time we choose a relation in M_{ij} we immediately discard all other relations.

We use the MAC (maintaining rc consistency) approach proposed by [10] to propagate the chosen constraints. Although this means that the time complexity of each node in the search tree grows considerably, it has been shown in [24] that the additional effort pays off when dealing with hard problems. The MAC strategy is implemented as follows: For each recursive call of CSPBIN (i.e. for each new assignment of a relation) we run through all pairs of rectangles i and j and temporarily select a relation in M_{ij} by deleting all other relations in the set. If the problem becomes infeasible with the additional relation imposed, the relation is removed from M_{ij}. If M_{ij} in this way becomes empty, we may conclude that no feasible solution can be obtained by following this branch, and thus we may backtrack in CSPBIN. If only one relation is left in a domain M_{ij} then this relation is fixed. All reductions in the domain achieved by the forward propagation are pushed to a stack, such that the domains can be restored quickly upon backtracking from CSPBIN.

The best performance of algorithm CSPBIN was obtained sorting the rectangles according to non-increasing volume, 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 and infeasibility could quickly be detected. This complies with the *fail first strategy* proposed by [1].

4.2 Solving the Pricing Problem Through CSP and Branch-and-Cut

The CSPBIN algorithm solves the decision problem of arranging a set of rectangles into a single bin. The pricing problem (9) is however an optimization problem. Using the

technique from Fekete and Schepers [9] we split the problem into a 1D optimization problem and a 2D packing decision problem. The 1D optimization asks to choose 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:

$$\begin{array}{lll} \max & \sum_{i\in\mathcal{R}} p_i x_i & \\ \text{s.t.} & \sum_{i\in\mathcal{R}} w_i h_i x_i \leq WH & \\ & x_i \in \{0,1\} & i \in \mathcal{R} \end{array} \tag{11}$$

Let $D = \{i \in \mathcal{R} : x_i = 1\}$ be those rectangles selected in (11). The 2D packing problem now attempts to arrange the rectangles D into a bin of size $W \times H$. This decision problem is solved through the CSPBIN algorithm.

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

$$\sum_{i\in D} x_i \leq |D| - 1 \tag{12}$$

to the knapsack problem (11). The inequality cuts off the present solution to (11) and it can easily be proved that repeating the above process will lead to an integer optimal solution to the pricing problem. The set of all generated valid inequalities is denoted $\mathcal{C}$.

Unfortunately, the branch-and-cut process may converge very slowly if the generated inequalities in $\mathcal{C}$ are weak. Hence, instead of adding inequalities of the form (12) whenever the rectangles D do not fit into the a bin, we would like to identify the smallest subset $D' \subseteq D$ which do not fit into the bin, giving rise to the tighter inequality $\sum_{i\in D'} x_i \leq |D'|-1$. 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, hence 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, hence in case of ONEBIN returning false, knowing 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, hence a compromise technique 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 which do not fit into the bin.

To further improve 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 which fit into a single bin. Still, a subset of D' may have the property

that they do not fit into a single bin, hence we prune D' by repeatedly testing whether $D' \setminus \{i\}$ 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.

4.3 Multi-Constrained Knapsack Problem

Having extended the model (11) with valid inequalities (12) we obtain a one-dimensional multi-constrained knapsack problem of the form

$$\begin{array}{lll} \max & \sum_{i \in \mathcal{R}} p_i x_i & \\ \text{s.t.} & \sum_{i \in \mathcal{R}} w_i h_i x_i \leq WH & \\ & \sum_{i \in C_j} x_i \leq |C_j| - 1 & C_j \in \mathcal{C} \\ & x_i \in \{0, 1\} & i \in \mathcal{R} \end{array} \tag{13}$$

This problem may be solved through a general IP-solver but as the number of constraints may grow very large, this approach soon becomes inappropriate. Hence, a simple branch-and-bound algorithm was developed for this purpose. The algorithm 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 (11) 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.

4.4 Adding Multiple Cuts

Whenever the CSPBIN algorithm returns a negative result we obtain a valid inequality on the form (12), which says that the rectangles in D cannot be packed in the same bin. By lifting the inequality we can derive additional valid constraints which tighten the 1DKPP further without solving the computationally expensive CSPBIN

Assuming a call to CSPBIN 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 r_i, $i \in C$ with a larger or equal rectangle r_j (i.e. $w_j \geq w_i$ and $h_j \geq h_i$), $j \in \mathcal{R} \setminus C$. An example of this is shown in Figure 1. If a rectangle r_{k_0} is larger or equal than all rectangles r_i, $i \in C$ we can add the valid inequality $\sum_{i \in C} x_i + x_{k_0} \leq |C| - 1$ instead of adding $|C|$ inequalities of the previous type. Clearly, this means adding several extra inequalities to the constrained 1D knapsack problem making it more time consuming to solve, but since calls to CSPBIN are quite time consuming we prefer to add as many valid inequalities as possible to reduce the number of times we need to solve CSPBIN.

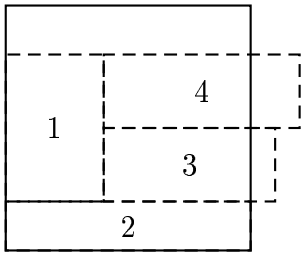

Figure 1: If we call CSPBIN on the rectangles in the figure, we get a negative answer and the equation $x_1 + x_2 + x_3 \leq 2$ is returned saying that rectangles 1, 2, and 3 can not be packed in a single bin. But since rectangle 4 is larger or equal to 3 (i.e. $w_4 \geq w_3$ and $h_4 \geq h_3$), clearly rectangles 1, 2, 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 multi-constrained 1D knapsack problem.

4.5 Dominance

Whenever we add a valid inequality it may happen that this inequality is dominated by an existing valid inequality or that it dominates 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.

Proposition 1 (Dominance) *An inequality* $X : \sum_{i\in\mathcal{I}} x_i \leq d_X, \forall i \in \mathcal{I} : x_i \in \{0,1\}$ *dominates another inequality* $Y : \sum_{k\in\mathcal{K}} x_k \leq d_Y, \forall k \in \mathcal{K} : x_k \in \{0,1\}$ *iff* $d_X \leq d_Y - |\mathcal{K} \setminus (\mathcal{I} \cap \mathcal{K})|$.

Proof: Let $\mathcal{M} = \mathcal{I}\cap\mathcal{K}$. Assume $d_X \leq d_Y - |\mathcal{K}\setminus\mathcal{M}|$. 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 $\sum_{i\in\mathcal{M}} x_i \leq d_X \Longrightarrow \sum_{k\in\mathcal{M}} x_k + \sum_{k\in\mathcal{K}\setminus\mathcal{M}} x_k \leq d_X + |\mathcal{K} \setminus \mathcal{M}| \leq d_Y$, since $\forall k \in \mathcal{K} : x_k \leq 1$. This proves that inequality Y is also satisfied.

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

4.6 Early Termination of the 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 generated without changing the optimal value of the restricted master problem significantly. If the column generation is terminated before completion, this will result in a non-valid lower bound.

Lagrangian relaxation on the other hand has the advantage that one gets a valid lower bound for any set of nonnegative Lagrangian multipliers. Hence we will use Lagrangian relaxation to obtain a valid lower bound based on the present restricted master problem [28]. For this purpose, we write up the problem (3) in an alternative way. The set of bins is $K = \{1, \ldots, m\}$ where m is an upper bound on the number of bins used. The binary variable u_{ik} is 1 iff rectangle i is placed in bin k, binary variable t_{ij} is 1 iff rectangle i and j are placed in the same bin, and finally v_k is 1 iff bin k is used. This leads to the MIP-model

$$
\begin{array}{lll}
\min & \sum_{k \in K} v_k & \\
\text{s.t.} & \sum_{k \in K} u_{ik} = 1 & i \in \mathcal{R} \\
& \sum_{i \in \mathcal{R}} u_{ik} \leq n v_k & k \in K \\
& u_{ik} + u_{jk} - t_{ij} \leq 1 & i, j \in \mathcal{R}, i < j, k \in K \\
& \ell_{ij} + \ell_{ji} + b_{ij} + b_{ji} \geq t_{ij} & i, j \in \mathcal{R}, i < j \\
& x_i - x_j + W \ell_{ij} \leq W - w_i & i, j \in \mathcal{R} \\
& y_i - y_j + H b_{ij} \leq H - h_i & i, j \in \mathcal{R} \\
& 0 \leq x_i \leq W - w_i & i \in \mathcal{R} \\
& 0 \leq y_i \leq H - h_i & i \in \mathcal{R} \\
& \ell_{ij}, b_{ij}, u_{ij}, t_{ij} \in \{0, 1\} & i, j \in \mathcal{R} \\
& v_k \in \{0, 1\} & i \in \mathcal{R} \\
& x_i, y_i \geq 0 & i \in \mathcal{R}
\end{array}
\tag{14}
$$

The first type of constraints say that every rectangle should be placed in *some* bin. The second type of constraints pushes v_k to 1 if any rectangles are placed in bin k. The third type of constraints say that if $u_{ik} = 1$ and $u_{jk} = 1$ for two rectangles i, j and a bin k then $t_{ij} = 1$, hence t_{ij} becomes one only if the two rectangles are placed in the same bin. The next type of constraints demand that $\ell_{ij} + \ell_{ji} + b_{ij} + b_{ji} \geq 1$ if $t_{ij} = 1$. I.e. if two rectangles are placed in the same bin, then they must be located left, right, below, or above each other. The remaining constraints are similar to those in (3).

Relaxing the first type of constraints in (14) in a Lagrangian way using multipliers

$\lambda_i \geq 0$ leads to the model

$$\begin{array}{lll}
\min & \sum_{k\in K} v_k - \sum_{i\in\mathcal{R}} \lambda_i \left(\sum_{k\in K} u_{ik} - 1\right) & \\
\text{s.t.} & \sum_{i\in\mathcal{R}} u_{ik} \leq n v_k & k \in K \\
& u_{ik} + u_{jk} - t_{ij} \leq 1 & i,j \in \mathcal{R}, k \in K \\
& \ell_{ij} + \ell_{ji} + b_{ij} + b_{ji} \geq t_{ij} & i,j \in \mathcal{R}, i < j \\
& x_i - x_j + W\ell_{ij} \leq W - w_i & i,j \in \mathcal{R} \\
& y_i - y_j + Hb_{ij} \leq H - h_i & i,j \in \mathcal{R} \\
& 0 \leq x_i \leq W - w_i & i \in \mathcal{R} \\
& 0 \leq y_i \leq H - h_i & i \in \mathcal{R} \\
& \ell_{ij}, b_{ij}, u_{ij}, t_{ij} \in \{0,1\} & i,j \in \mathcal{R} \\
& v_k \in \{0,1\} & i \in \mathcal{R} \\
& x_i, y_i \geq 0 & i \in \mathcal{R}
\end{array} \tag{15}$$

The objective function may be rewritten

$$L(\lambda) = \min \quad \sum_{k\in K} \left(v_k - \sum_{i\in\mathcal{R}} \lambda_i u_{ik}\right) + \sum_{i\in\mathcal{R}} \lambda_i \tag{16}$$

We wish to maximize the lower bound, hence the Lagrangian dual problem should be solved

$$\max_{\lambda\geq 0} L(\lambda) \tag{17}$$

The Lagrangian dual problem may be quite time-consuming to solve, so instead we "guess" some appropriate values of the Lagrangian multipliers. Choosing λ_i as the dual variables π_i corresponding to the present restricted master problem, we solve the problem

$$\min \quad \sum_{k\in K} \left(v_k - \sum_{i\in\mathcal{R}} \pi_i u_{ik}\right) + \sum_{i\in\mathcal{R}} \pi_i \tag{18}$$

subject to constraints (15) where the first part of the objective function has a solution equal to $|K| = m$ times the pricing problem (9). This leads to the following lower bound

$$a = mr + \sum_{i\in\mathcal{R}} \pi_i \tag{19}$$

where m is an upper bound on the number of bins used (typically obtained through a heuristic or the incumbent solution in the branch-and-price algorithm), r is the reduced cost of the last pricing problem solved, and $\pi_i \geq 0$, $i \in \mathcal{R}$ are the dual variables of the present restricted master problem. If $\lceil a \rceil = m$ we may terminate the column generation, knowing that the present upper bound equals the lower bound.

5 A Heuristic Pricing Algorithm

The 2D knapsack problem described in the previous section is $\mathcal{NP}$-hard, hence to speed up the column generation the pricing problem is solved heuristically whenever possible. We

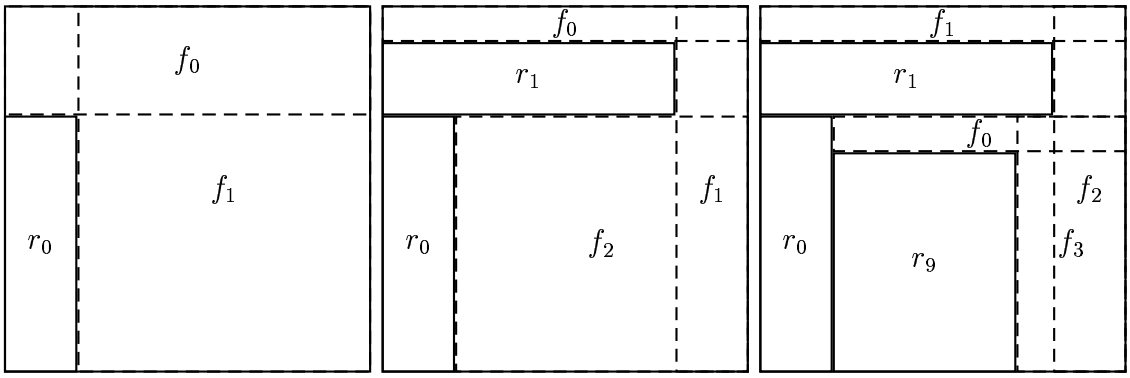

Figure 2: An example of the greedy filling by the heuristic pricing algorithm. The r_i's are the rectangles placed and the f_i's are the *free space rectangles* in the bin.

use a heuristic that solves the 2D knapsack problem in polynomial time, however 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 2D knapsack algorithm from Section 4.

The heuristic pricing algorithm is based on the greedy paradigm. It starts out 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 is a collection of *maximal* rectangles f satisfying $f \cap r = \emptyset$ for all $r \in \mathcal{R}$. *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 $\bigcup f = W \times H \setminus \bigcup r$. In every iteration a rectangle is placed in one the free space rectangles where it fits and the set of new free space rectangles is determined for the bin. An example is shown in Figure 2.

In every iteration, the heuristic pricing algorithm selects a rectangle to place in the bin and a free space rectangle in which it should be placed. Two different strategies for selecting a rectangle and two different strategies for selecting a free space have been considered.

Choosing a Rectangle

The objective of the heuristic pricing algorithm is to determine a 2D packing which minimizes the corresponding reduced cost. Since we are only interested 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

```
Heuristic_filling
    R ← {1,...,n}; F ← {(W × H)}
    while(R ≠ ∅)
        select a rectangle r from R
        let p_r be the reduced cost associated with r
        if(p_r < 0)
            select a free space rectangle f from F in which r fits
            if(f exists) place r at f; update F
        R ← R \ { r }
```

Figure 3: Pseudo-code for the heuristic pricing algorithm

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 respectively the reduced price/area ratio and the reduced price of the rectangle.

Choosing a Free Space Rectangle

Having chosen a rectangle r 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 r.
2. *Maximize Largest.* Choose the free space rectangle f, that maximizes the largest free space rectangle in the bin after placing r in f .

If no free space rectangle exists that can hold the chosen rectangle r, we discard r and choose another rectangle. When there are no more rectangles left, the heuristic terminates. Figure 3 shows the pseudo-code for the heuristic pricing algorithm.

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 produces such a packing, the exact pricing algorithm is applied. The randomized heuristic pricing algorithms may be run several times before applying the exact pricing algorithm.

6 The Primal Bound

The column generation procedure 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 which finds very high quality solutions. Although the column generation in principle can start up 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-art heuristic from Faroe, Pisinger, Zachariasen [8] was chosen to provide the initial columns as well as an initial upper bound.

The heuristic is based on Guided Local Search (GLS) as described in Voudouris and Tsang [26, 27] and attempts to pack all the rectangles into a fixed number m of bins. A feasible packing will have no overlap between rectangles, hence 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 in horizontal direction, vertical direction, or moving it to another bin. The GLS framework is based on a "hill-climbing" approach which repeatedly performs an improving local search iteration until the algorithm is trapped in a local minima. At this moment, the GLS algorithm selects a pair of rectangles which posses the overlap feature, assign a punishment to this feature, and reoptimize the problem with respect to a modified objective function. In order to minimize the number of bins used, the GLS algorithm is called for decreasing values of m, until a feasible packing cannot be found within a given number of iterations.

During the execution of the branch-and-price algorithm we may improve the primal bound. During the column generation numerous instances of the restricted master problem (7) are solved to LP-optimality. Any IP-solution to this problem provides a valid solution for the 2DBPP and hence may be used for tightening the upper bound. The IP variant of (7) may be solved by CPLEX, but since it is quite time consuming, we only solve the problem to integer optimality every M_3 iterations. A proper value of M_3 was experimentally found as $M_3 = 50$.

7 Branching

The LP-solutions to (6) found by column generation are not necessarily integral. In order to obtain optimal integral solutions we apply a branching rule which excludes fractional solutions. As always when designing a branch-and-bound algorithm we want a branching scheme which divides the solution space fairly even in order 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 the following branching rule: Choose two rectangles r_i and r_j, $i \neq j$. Divide the solution space into two branches. On one branch we demand that rectangles r_i and r_j

may not be in the same bin and on the other branch we demand that r_i and r_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 LP model, and the second branch corresponds to adding the two constraints

$$\sum_{p \in \mathcal{P}} x_p \delta_i^p (1 - \delta_j^p) = 0 \qquad \sum_{p \in \mathcal{P}} x_p (1 - \delta_i^p) \delta_j^p = 0$$

to the LP model.

When solving the pricing problem in some 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 Section 4, the pricing algorithm consists of relaxing the 2D knapsack problem to a constrained 1D knapsack problem and checking if the solution from the 1D knapsack problem is valid. If a branching constraint saying that rectangles r_i and r_j have to be in the same bin is present in the subproblem, we simply merge the two rectangles into one r_{ij}. The area of r_{ij} is thus $w_i h_i + w_j h_j$, which is used when the constrained 1D knapsack problems are solved during the column generation in this subproblem. If on the other hand r_i and r_j have to be in different bins, we simply add the inequality $x_i + x_j \leq 1$ to the constrained 1D knapsack problem. Observe that this inequality is only locally valid in contrast to the other inequalities of the constrained 1D knapsack problem which are globally valid.

As mentioned the branching scheme should divide the solution space evenly. Unfortunately our branching rule does not, since it is a stronger constraint to demand that two rectangles have to be in the same bin, than demanding that two rectangles have to be in different bins. This means that the solution space of the second branch is bigger than on the first branch. However, if we branch on two rectangles from a bin with high fractional value in the LP solution, we are given two rectangles which probably should be in the same bin in an optimal solution. This makes the second type of branching constraint stronger, balancing the division of the solution space.

8 Computational Results

We have implemented a branch-and-price algorithm to solve 2DBPP as described in the previous sections. For this purpose we used ABACUS ("A Branch-And-CUt System") [25] 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 [6] 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

have implemented the lower bounds of Martello and Vigo [19] and of Fekete and Schepers [9]. All tests have been carried out on an Intel Pentium III-933 with 1 GB of memory.

We have tested our algorithm on the test instances provided by Lodi, Martello and Vigo in [16]. These instances consist 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 class I-VI have been proposed by Berkey and Wang while the last four classes have been proposed by Lodi, Martello and Vigo in [16]. The first 6 problem classes have the following characteristics:

Class I: w_r and h_r uniformly random in $[1, 10]$, $W = H = 10$.

Class II: w_r and h_r uniformly random in $[1, 10]$, $W = H = 30$.

Class III: w_r and h_r uniformly random in $[1, 35]$, $W = H = 40$.

Class IV: w_r and h_r uniformly random in $[1, 35]$, $W = H = 100$.

Class V: w_r and h_r uniformly random in $[1, 100]$, $W = H = 100$.

Class VI: w_r and h_r uniformly random in $[1, 100]$, $W = H = 300$.

In the last four problem classes four types of rectangles are used in the problems:

Type I: w_r uniformly random in $[\frac{2}{3}W, W]$, h_r uniformly random in $[1, \frac{1}{2}H]$.

Type II: w_r uniformly random in $[1, \frac{1}{2}W]$, h_r uniformly random in $[\frac{2}{3}H, H]$.

Type III: w_r uniformly random in $[\frac{1}{2}W, W]$, h_r uniformly random in $[\frac{1}{2}H, H]$.

Type IV: w_r uniformly random in $[1, \frac{1}{2}W]$, h_r uniformly random in $[1, \frac{1}{2}H]$.

The last four problem classes all have $W = H = 100$ and the rectangles are as follows:

Class VII: type 1 with probability 70%, type 2, 3, 4 with probability 10% each.

Class VIII: type 2 with probability 70%, type 1, 3, 4 with probability 10% each.

Class IX: type 3 with probability 70%, type 1, 2, 4 with probability 10% each.

Class X: type 4 with probability 70%, type 1, 2, 3 with probability 10% each.

8.1 Results

In tables 1 and 2 we have shown the results of our branch-and-price algorithm on the 10 problem classes. Each row in the tables is summation of 10 problems of the specified type. We present the number of improved lower bounds in the root node of the branch-and-bound tree compared to the lower bound by Fekete and Schepers, the number of problems solved to optimality, the average CPU time used, the average number of columns generated, the average number of cuts generated, the average number of branch-and-bound nodes, and the average CPU time used for the obtaining the lower bound in the root node.

In tables 3 and 4 we have summarized the results for different classes and different problem sizes. From Table 3 we see that the hardest problem classes to solve are classes

Class	# Items	# Improved Low. Bounds	# Solved Optimally	Avg. CPU (seconds)	Avg. # Col. Gen.	Avg. # Cuts Gen.	Avg. # B&B nodes	Avg. CPU Low. Bound
I	20	2	10	13	6	17	1	0
I	40	3	10	19	27	166	1	1
I	60	1	10	345	49	5552	4	318
I	80	0	10	51	0	0	1	0
I	100	0	10	836	288	20502	22	583
II	20	0	10	1	0	0	1	0
II	40	0	10	275	160	3	1	360
II	60	0	10	1357	698	81	18	1797
II	80	0	9	2523	1319	214	204	3225
II	100	0	9	2256	1399	295	108	2857
III	20	4	10	16	37	83	1	1
III	40	1	10	270	161	4565	7	173
III	60	4	10	646	339	14672	7	600
III	80	2	10	1201	645	26204	67	1156
III	100	3	8	2336	963	32500	110	2371
IV	20	0	10	1	0	0	1	0
IV	40	0	10	193	252	9	5	230
IV	60	0	8	2085	1033	94	28	2490
IV	80	0	7	2970	1592	191	197	3488
IV	100	1	9	2537	1834	183	55	3013
V	20	5	10	17	9	29	1	0
V	40	3	10	50	55	2004	1	26
V	60	3	10	415	208	5289	7	365
V	80	6	9	892	344	17940	8	842
V	100	4	7	1853	793	34112	35	1584
VI	20	0	10	0	0	0	1	0
VI	40	0	8	1304	883	20	1	1455
VI	60	0	9	2384	1093	89	28	2211
VI	80	0	10	2690	1206	33	228	204
VI	100	0	8	2880	206	0	65	0

Table 1: Results for problem classes I - VI

Class	# Items	# Improved Low. Bounds	# Solved Optimally	Avg. CPU (seconds)	Avg. # Col. Gen.	Avg. # Cuts Gen.	Avg. # B&B nodes	Avg. CPU Low. Bound
VII	20	2	10	11	11	286	1	1
VII	40	0	8	744	74	4371	2	716
VII	60	0	7	1317	164	18265	23	1239
VII	80	0	2	2909	357	34520	111	2742
VII	100	0	8	1484	342	22107	34	1200
VIII	20	3	10	14	10	150	1	0
VIII	40	0	9	388	19	2601	1	358
VIII	60	0	8	871	90	10887	83	789
VIII	80	0	9	584	69	7983	8	432
VIII	100	0	7	1823	363	32881	81	1440
IX	20	0	10	0	0	0	1	0
IX	40	3	10	13	4	113	1	0
IX	60	2	10	19	1	190	1	0
IX	80	3	10	40	2	472	1	2
IX	100	2	10	66	3	629	1	4
X	20	1	9	371	49	235	1	365
X	40	2	9	389	121	798	1	365
X	60	1	7	1722	673	22972	11	1625
X	80	1	5	2719	1088	24738	28	2675
X	100	3	1	3510	1605	30830	31	3447

Table 2: Results for problem classes VII - X

Class	# Improved Low. Bounds	# Solved Optimally
I	6	50
II	0	48
III	14	48
IV	1	44
V	21	46
VI	0	45
VII	2	35
VIII	3	43
IX	10	50
X	8	31
Sum	65	440

Table 3: Summarized results for different problem classes

# Items	# Improved Low. Bounds	# Solved Optimally
20	17	99
40	12	94
60	11	89
80	12	81
100	13	77
Sum	65	440

Table 4: Summarized results for different problem sizes

VII, VIII, and X which all contain many medium sized items. For problems with many small items, the continuous lower bound is tight and these problems are therefore quite easy to solve. For problems with many large items each bin only has room for 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 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 sized and large items (classes I, III, and V). For these problem classes the present algorithm is performing very well.

From Table 4 we see, as we would expect, that problems with more items are harder to solve than problems with fewer items. We also see that the number of lower bounds improved by our algorithm seems to be independent of the number of items in the problems. On the other hand we see from Table 3 that the number of improved lower bounds varies between different problem classes. Classes III, V, and IX have the largest numbers of improved lower bounds. These are the classes with mixed item sizes, on which the Martello and Vigo algorithm did not perform well.

In Table 5, 6, and 7 we show the lower bound calculations in greater detail. The values of the column generation lower bounds are compared to Martello and Vigo's L_0, L_1, L_2, L_4 and to Fekete and Schepers' lower bounds FS which have been proved to dominate Martello and Vigo's. In the table we have also shown the best known upper bounds for comparison. We see that the column generation lower bounds are greater or equal to Fekete and Schepers' in all cases, however, the improvement varies with the problem classes. For problem classes II, IV, and VI the two lower bounds are very similar. These are the classes with many small items and for these classes the simple L_0 lower bounds actually match Fekete and Schepers' lower bounds. The largest improvement of Fekete and Schepers'

Class	Size	L_0	L_1	L_2	L_4	FS	Col. gen.	UB
I	20	6.4	5.9	6.7	6.7	6.9	7.1	7.1
I	40	12.0	11.7	12.8	12.8	13.1	13.4	13.4
I	60	18.5	17.8	19.3	19.3	19.9	20.0	20.0
I	80	25.3	24.7	26.9	26.9	27.5	27.5	27.5
I	100	30.5	28.6	31.4	31.4	31.7	31.7	31.8
II	20	1.0	0.0	1.0	1.0	1.0	1.0	1.0
II	40	1.9	0.0	1.9	1.9	1.9	1.9	2.0
II	60	2.5	0.0	2.5	2.5	2.5	2.5	2.6
II	80	3.1	0.0	3.1	3.1	3.1	3.1	3.3
II	100	3.9	0.0	3.9	3.9	3.9	3.9	4.0
III	20	4.4	4.4	4.6	4.6	4.7	5.1	5.1
III	40	8.2	8.2	8.8	8.8	9.2	9.3	9.4
III	60	12.5	12.7	13.3	13.3	13.5	13.9	14.0
III	80	17.3	17.6	18.4	18.4	18.7	18.9	19.0
III	100	20.5	21.0	21.7	21.7	22.0	22.3	22.8
IV	20	1.0	0.0	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
IV	60	2.3	0.0	2.3	2.3	2.3	2.3	2.5
IV	80	3.0	0.0	3.0	3.0	3.0	3.0	3.3
IV	100	3.7	0.0	3.7	3.7	3.7	3.8	3.8
V	20	5.4	5.9	6.0	6.0	6.0	6.5	6.5
V	40	10.1	11.0	11.4	11.4	11.6	11.9	11.9
V	60	15.7	16.7	17.2	17.2	17.6	17.9	18.0
V	80	21.5	23.2	23.6	23.6	24.0	24.6	24.8
V	100	25.9	27.1	27.3	27.3	27.7	28.1	28.8
VI	20	1.0	0.0	1.0	1.0	1.0	1.0	1.0
VI	40	1.5	0.0	1.5	1.5	1.5	1.5	1.9
VI	60	2.1	0.0	2.1	2.1	2.1	2.1	2.2
VI	80	3.0	0.0	3.0	3.0	3.0	3.0	3.0
VI	100	3.2	0.0	3.2	3.2	3.2	3.2	3.4
VII	20	4.7	5.1	5.3	5.3	5.3	5.5	5.5
VII	40	9.7	10.4	10.8	10.8	10.9	10.9	11.1
VII	60	14.0	14.9	15.5	15.5	15.5	15.5	15.8
VII	80	19.7	21.7	22.3	22.3	22.4	22.4	23.2
VII	100	23.8	25.7	26.8	26.8	26.9	26.9	27.3
VIII	20	4.8	5.3	5.5	5.5	5.5	5.8	5.8
VIII	40	9.6	10.2	11.1	11.1	11.2	11.2	11.3
VIII	60	14.1	15.3	15.9	15.9	15.9	15.9	16.1
VIII	80	19.5	21.3	22.2	22.2	22.3	22.3	22.4
VIII	100	24.1	26.5	27.3	27.3	27.4	27.4	27.8
IX	20	9.4	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
IX	60	27.6	43.3	43.3	43.3	43.5	43.7	43.7
IX	80	37.1	56.9	56.9	56.9	57.4	57.7	57.7
IX	100	45.0	68.9	68.9	68.9	69.3	69.5	69.5
X	20	3.8	3.4	4.0	4.0	4.0	4.1	4.2
X	40	6.9	6.1	7.1	7.1	7.1	7.3	7.4
X	60	9.4	7.8	9.7	9.7	9.7	9.8	10.2
X	80	12.2	9.8	12.3	12.3	12.3	12.4	13.0
X	100	15.3	11.9	15.3	15.3	15.3	15.6	16.5
Sum		598.0	642.7	706.4	706.4	712.9	719.4	727.6

Table 5: Lower bounds: Martello and Vigo's L_0, L_1, L_2, L_4 and Fekete and Schepers' FS and the column generation lower bound calculated in this article

Class	L_0	L_1	L_2	L_4	FS	Col. gen.	UB
I	92.7	88.7	97.1	97.1	99.1	99.7	99.8
II	12.4	0.0	12.4	12.4	12.4	12.4	12.9
III	62.9	63.9	66.8	66.8	68.1	69.5	70.3
IV	11.9	0.0	11.9	11.9	11.9	12.0	12.5
V	78.6	83.9	85.5	85.5	86.9	89.0	90.0
VI	10.8	0.0	10.8	10.8	10.8	10.8	11.5
VII	71.9	77.8	80.7	80.7	81.0	81.2	82.9
VIII	72.1	78.6	82.0	82.0	82.3	82.6	83.4
IX	137.1	210.8	210.8	210.8	212.0	213.0	213.0
X	47.6	39.0	48.4	48.4	48.4	49.2	51.3

Table 6: Summarized lower bound calculations for the ten problem classes

Size	L_0	L_1	L_2	L_4	FS	Col. gen.	UB
20	41.9	44.3	49.4	49.4	49.7	51.4	51.5
40	79.8	85.0	94.7	94.7	95.9	97.1	98.1
60	118.7	128.5	141.1	141.1	142.5	143.6	145.1
80	161.7	175.2	191.7	191.7	193.7	194.9	197.2
100	195.9	209.7	229.5	229.5	231.1	232.4	235.7

Table 7: Summarized lower bound calculations for the five problem sizes

lower bounds is seen for classes III and V. These are the classes where we have an evenly distributed amount of small, medium sized and big items. It seems reasonable that the column generation lower bound has the greatest advantage over Fekete and Schepers' lower bounds for these problem classes since neither L_0 or bounding by considering the big items yields a tight bound in this case.

From the summation of the lower bounds presented in the bottom line of Table 5 we get a rough overview of the quality of the different lower bounds. We can see that the Fekete and Schepers lower bound decreases the gap between L_4 and the best known upper bound considerably. The column generation lower bound further decreases this gap by 44% providing a very tight lower bound. But from the previous section we can also conclude, that the improvement of the column generation lower bound varies considerably with the types of problems it is applied to.

8.2 Conclusion

Although Gilmore and Gomory [11] already several decades ago presented the first formulation of 2DBPP based on column generation, the present paper is, to the best knowledge of the authors, the first to report results on large-sized instances using this approach. The pricing problem in two dimensions is much more difficult to solve than in one dimension, hence we proposed a combination of constraint programming and branch-and-cut for solving the resulting two-dimensional knapsack problem. The computational results show that the lower bounds obtained through column generation are tighter than any bounds previously published in the literature. Due to these tight lower bounds we are able to solve to optimality 2DBPP instances of quite large size.

A second contribution of the present paper is to investigate how the best properties of CSP and ILP can be used for solving large-scale industrial problems by use of decomposition techniques. Previous papers have focused on developing unified solution methods and unified programming languages for ILP and CSP. The present work has shown how the two techniques may be used at their best premises, using results from one approach to strengthen the formulation of the other approach.

Finally, several new polynomial IP-formulations of the 2DBPP and related packing problems have been presented, making it easier to solve small-sized packing problems through general IP-solvers.

References

[1] R. Barták. Online guide to constraint programming, 1998. `http://kti.mff.cuni.cz/~bartak/constraints/`.

[2] B.E. Bengtsson. Packing rectangular pieces – a heuristic approach. *The Computer Journal*, 25:353–357, 1982.

[3] S. C. Brailsford, C. N. Potts, and B. M. Smith. Constraint satisfaction problems: Algorithms and applications. *European Journal of Operational Research*, 119:557–581, 1999.

[4] A. Caprara, M. Fischetti, and P. Toth. A heuristic method for the set covering problem. *Operations Research*, 47(5):730–743, 1999.

[5] C.S. Chen, S.M. Lee, and Q.S. Shen. An analytical model for the container loading problem. *European Journal of Operational Research*, 80:68–76, 1995.

[6] Cplex. *Using the Cplex Callable Library.* Cplex Optimization, Inc., 1995.

[7] K. Dowsland. Some experiments with simulated annealing techniques for packing problems. *European Journal of Operational Research*, 68:389–399, 1993.

[8] O. Faroe, D. Pisinger, and M. Zachariasen. Guided local search for the three-dimensional bin packing problem. *INFORMS Journal on Computing*, 2000. to appear.

[9] S. P. Fekete and J. Schepers. New classes of lower bounds for bin packing problems. In R. E. Bixby, E. A. Boyd, and R. Z. Ríos-Mercado, editors, *Integer Programming and Combinatorial Optimization, Proc. 6th International IPCO Conference*, volume 1412 of *Lecture Notes in Computer Science*, pages 257–270. Springer-Verlag, 1998.

[10] J. Gaschnig. Performance measurement and analysis of certain search algorithms. Technical Report CMU-CS-79-124, Carnegie-Mellon University, 1979.

[11] P.C. Gilmore and R.E. Gomory. A linear programming approach to the cutting stock problem. *Operations Research*, 9:849–859, 1961.

[12] P.C. Gilmore and R.E. Gomory. A linear programming approach to the cutting stock problem – part II. *Operations Research*, 13:94–119, 1963.

[13] E. Hadjiconstantinou and N. Christofides. An exact algorithm for general, orthogonal, two-dimensional knapsack problems. *European Journal of Operational Research*, 83:39–56, 1995.

[14] R.M. Haralick and G.L. Elliot. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, pages 263–313, 1980.

[15] J. Hooker, G. Ottosson, E. S. Thorsteinsson, and H.-J. Kim. A scheme for unifying optimization and constraint satisfaction methods. *Knowledge Engineering Review, Special Issue on Artificial Intelligence and Operations Research*, 15, 2000.

[16] A. Lodi, S. Martello, and D. Vigo. Heuristic and metaheuristic approaches for a class of twodimensional bin packing problems.

[17] A. Lodi, S. Martello, and D. Vigo. Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems. *INFORMS Journal on Computing*, 11:345–357, 1999.

[18] A. Lodi, S. Martello, and D. Vigo. Recent advances on two-dimensional bin packing problems. *Discrete Applied Mathematics*, 123:379–396, 2002.

[19] S. Martello and D. Vigo. Exact solution of the two-dimensional finite bin packing problem. *Management Science*, 44:388–399, 1998.

[20] M. Monaci. *Algorithms for Packing and Scheduling Problems.* PhD thesis, University of Bologna, 2002.

[21] H. Onodera, Y. Taniguchi, and K. Tmaru. Branch-and-bound placement for building block layout. *28th ACM/IEEE design automation conference*, pages 433–439, 1991.

[22] D. Pisinger, E. den Boef, J. Korst, S. Martello, and D. Vigo. Robot packable and general variants of the three-dimensional bin packing problem. *submitted*, 2001.

[23] D. Pisinger and P. Toth. Knapsack problems. In D.Z. Du and P. Pardalos, editors, *Handbook of Combinatorial Optimization, vol 1.* Kluwer Academic Publishers, 1998.

[24] D. Sabin and E.C. Freuder. Contradicting conventional wisdom in constraint satisfaction. In A. Borning, editor, *Proceedings of the Second International Workshop on Principles and Practice of Constraint Programming, PPCP'94, Rosario, Orcas Island, Washington, USA*, volume 874, pages 10–20, 1994.

[25] S. Thienel. *ABACUS — A Branch-And-CUt System.* PhD thesis, Universität zu Köln, 1995.

[26] C. Voudouris. Guided local search for combinatorial optimisation problems. Ph.D. Thesis, Dept. of Computer Science, University of Essex, Colchester, England, 1997.

[27] C. Voudouris and E. Tsang. Guided local search and its application to the traveling salesman problem. *European Journal of Operational Research*, 113:469–499, 1999.

[28] L.A. Wolsey. *Integer Programming.* Wiley Interscience, 1998.