Multi-dimensional Bin Packing Problems with Guillotine Constraints

Rasmus R. Amossen* David Pisinger†

February 2010

Abstract

The problem addressed in this paper is the decision problem of determining if a set of multi-dimensional rectangular boxes can be orthogonally packed into a rectangular bin while satisfying the requirement that the packing should be *guillotine cuttable*. That is, there should exist a series of face parallel straight cuts that can recursively cut the bin into pieces so that each piece contains a box and no box has been intersected by a cut. The unrestricted problem is known to be NP-hard. In this paper we present a generalization of a constructive algorithm for the multi-dimensional bin packing problem, with and without the guillotine constraint, based on constraint programming.

1 Introduction

Arranging boxes into bins is a problem that occurs in a variety of situations: in the industry, one goal is to minimize transport and storage costs by packing as many items as possible per space unit. Another goal may be to maximize the utilization of material plates (e.g. glass or wood) for cutting by arranging the elements to be cut out cleverly. At higher dimensions the problem of packing becomes relevant in, e.g., scheduling tasks with multiple resource demands.

In the following we consider the general problem of packing d-dimensional boxes into d-dimensional bins. Let therefore V be a set of boxes and let $w : V \rightarrow \mathbb{R}_0^{+d}$ be a size function describing the width of each box in all dimensions $x_1, x_2, \ldots, x_d$. All bins are restricted to have the same size $W \in \mathbb{R}_0^{+d}$. Now we can loosely define a feasible *packing* as an arrangement of the boxes V into one or more bins so that no boxes overlap and no box exceeds the boundaries of the bin in which it is placed. Rotations are not allowed and box edges must be parallel to bin edges. Several problems can be formulated from the task of packing boxes into bins. Some essential ones include:

*IT University of Copenhagen, Rued Langgaards Vej 7, DK-2300 Copenhagen S
†DTU Management Engineering, Produktionstorvet 424, DK-2800 Kgs. Lyngby

Orthogonal Packing Problem (OPP-d) Given a set of boxes V, can V be packed into a single bin of size W? OPP-d is NP-complete [FS97].

Orthogonal Bin Packing Problem (OBPP-d) Given the set V, how few bins of size W is required for packing all the boxes in V? Since OPP-d is the corresponding decision problem for OBPP-d and OPP-d is NP-complete, OBPP-d is NP-hard.

Orthogonal Knapsack Problem (OKP-d) Given a single bin of size W and a value function $v : V \to \mathbb{R}_0^+$, choose a subset $V' \subseteq V$ that can be packed into the bin so the sum of values for V' is maximized. Again OPP-d is the corresponding decision problem so OKP-d is NP-hard as well.

Due the theoretical and practical hardness, OPP-d has been approached in several ways. Chen, Lee and Shen [CLS95] approached the problem with integer-programming and Fekete and Schepers [FS97, FS04b] presented a general multi-dimensional model based on graph theory, where bounds based on conservative scales [FSdV07] (dual feasible functions) are used to prune the search. In a more recent paper, Clautiaux et al. [CAdC08], give a survey of dual feasible functions and their use for bounding the search tree in cutting problems. Pisinger and Sigurd [PS07] presented an algorithm for $d = 2$ based on constraint programming (CSP) in which a packing was constructed by assigning appropriate relations to each pair of boxes $u, v \in V$. Clautiaux et al. [CJCM08] modeled the OPP-2 as a scheduling problem, making it possible to use powerful constraint-based scheduling propagation techniques. Martello et al. [MPV00] generalized the CSP technique to the case $d = 3$. Note, that contour building approaches as proposed in [Sch92] for $d = 2$ cannot be generalized to $d \geq 3$ as shown in [MPV$^+$07]. For a recent survey of relaxations of OPP-d see Belov et al. [BKRS09].

When cutting specific materials like glass it may be required that the rectangles can be cut out of the bin by a number of guillotine cuts which can be thought of as edge-to-edge cuts (see Figure 1). As noted in [Bel03], these constraints are very difficult to formulate in an IP-model, and also in practice the problem is difficult to solve as reported in Parada et al. [PPSG00]. Pisinger and Sigurd [PS07] presented an algorithm that handled the guillotine constraint in the CSP framework by testing the guillotine criteria in each step. In [Amo05] Amossen extended the multi-dimensional graph based model by Fekete and Schepers to also handled the guillotine constraint. The present paper will generalize the CSP technique for guillotine cuttings to arbitrary values of d as well.

None of the algorithms in the above papers used data structures that automatically ensures the guillotine property. Instead a series of tests were performed ongoingly. A guillotine ensuring data structure was used in work by Wong and Liu [WL86] who modeled guillotine cuts as a tree structure and represented the tree structure by *normalized polish expressions*. Also Christofides and Whitlock [CW77] used tree structures to model guillotine cuts. Both [WL86] and [CW77] considered guillotine cuttings from a top-down point of view where different cut

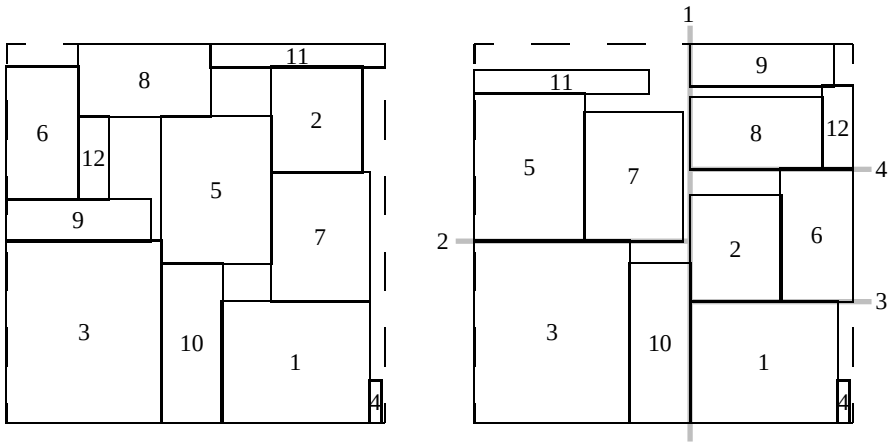


Figure 1: A non-guillotine pattern (left) and a guillotine pattern (right) with the first four cuts marked.

configurations were applied to a bin until a cut structure that could hold all boxes was found. Reversely, Wang [Wan83] iteratively assembled the boxes in a bottom-up approach.

For the bin packing variant OBPP-d, two techniques are frequently used, both making use of OPP-d as subproblem. One method is to distribute boxes into bins with a branch-and-bound algorithm like Martello et al. [MPV00], who used an outer branch-and-bound algorithm distributing boxes to bins, and an inner OPP-3 algorithm test the feasibility of each such distribution. The other method is to formulate the problem as a mixed integer model. Pisinger and Sigurd [PS07] concerned OBPP-2 and used the idea from Dantzig-Wolfe [DW60] to decompose the model into a restricted master problem and a number of subproblems. Each of the subproblems were then split into a one-dimensional pricing problem and a OPP-2 decision problem. Clautiaux et al. [CAdC09] give a very recent survey on lower bounds for OPP-d.

When solving the knapsack variant OKP-d, it is common to use a two-stage approach as first proposed by Fekete and Schepers [FS04a]: First, a relaxed one-dimensional knapsack problem is solved to find a subset of most profitable items that fit within the area or volume, and then an OPP-d subproblem is used to check feasibility. In [FS04a] the OPP-d was solved through an enumerative algorithm based on isomorphic packing classes. Pisinger and Sigurd [PS07] used constraint programming to solve th OPP-2, while Baldacci and Boschetti [BB07] used a cutting-plane approach using a number of knapsack-, dominance- and incompatibility constraints.

Viswanathan and Bagchi [VB93] presented a best-first branch-and-bound algorithm for the orthogonal guillotine constrained OKP-2. The algorithm is based on the bottom-up strategy of Wang [Wan83]. Hifi [Hif97] improved the algorithm of Viswanathan and Bagchi by introducing improved upper and lower bounds based on the solution of one-dimensional bounded knapsack problems. Cung et

al. [CHC97] further improved the algorithm by adding symmetry detection for duplicate patterns, and using more eficient data structures. Hifi [Hif98] presented an exact algorithm for the strip packing problem with guillotine constraints.

Bediceanu and Carlsson [BC01] generalized a sweep algorithm to prune constraints having at least two variables in common by utilizing the concepts of forbidden and safe regions, and only knowing the minimum and maximum number of constraints to be satisfied. They specialized the technique to an extension of the non-overlapping rectangles constraint. Later, Bediceanu, Carlsson and Thiel [BCT06] extended the swap algorithm so that adjusting the bounds of a variable according to a set of conjunctions was done in a synchronized way instead of by independent sweeps.

This paper is organized as follows: Section 2 introduces the terminology used throughout this paper and Section 3 presents a constraint programming solution for solving the guillotine packing problem. Computational results for the developed algorithm are reported in Section 4 where the algorithm is compared to three-dimensional solvers for solving general and robot packable variants of the problems.

The present paper may be seen as an extension of [PS05, PS07, MPV$^+$07] where we generalize the packing algorithms to an arbitrary number of dimension, and develope efficient techniques for testing guillotine cuttability. Moreover, we experimentally compare three packing methods (unrestricted, robot packable, and guillotine cuttable) with respect to solution time and solution quality. For the considered instances, it turns out that the solution value of bin packing problems hardly is affected by adding constraints on the packing method.

2 Terminology

We aim to formalize the idea of packings by considering the positions of all boxes in a Cartesian coordinate system. For that, define the map $p : V \to \mathbb{R}_0^{+d}$ as the coordinate of the corner closest to the origin of each box. The box positions are then uniquely defined by p. Also define

$$\begin{aligned} I_i : V &\to \mathbb{R}_0^+ \times \mathbb{R}^+ \\ v &\mapsto [p_i(v), p_i(v) + w_i(v)) \end{aligned}$$

as the interval occupied by box v on the x_i-axis. With p and I in hand we can now define a packing formally:

Definition 2.1 (Packing) *A* feasible packing *of the tuple (V, w, W) is a function $p : V \to \mathbb{R}_0^{+d}$ that satisfies*

$$\forall v \in V : \qquad p(v) + w(v) \leq W \tag{1}$$

$$\forall u, v \in V, u \neq v, \exists i \in \{1, \ldots, d\} : \qquad I_i(u) \cap I_i(v) = \emptyset \tag{2}$$

Constraint (1) ensures that no box exceeds the bin boundaries and (2) ensures that no two boxes overlap.

If a packing p arranges the boxes so they are either placed at origin or at the edge of another box we call it gapless. More formally:

Definition 2.2 (Gapless packing) *A packing is said to be* gapless *if for all* $i = 1, \ldots, d$ *and* $v \in V$

$$p_i(v) = 0 \text{ or } \exists u \in V : p_i(v) = p_i(u) + w_i(u)$$

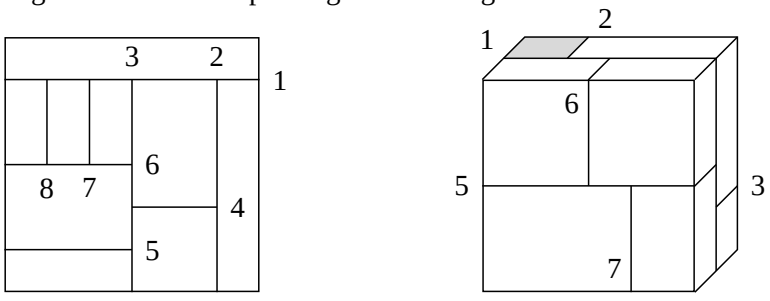
In this paper we only consider gapless packings, and in fact only gapless *guillotine* packings:

Definition 2.3 (Guillotine packing) *Let* p *be a* d*-dimensional packing of* V*. A* $(d-1)$*-dimensional axis parallel hyper plane* $\mathscr{P}$ *is called a* guillotine cut *if it divides* V *into two disjoint nonempty subsets* V_1 *and* V_2 *such that no box* $v \in V$ *is intersected by* $\mathscr{P}$ *with respect to* p*.*

Two subsets $U_1, U_2 \subseteq U \subseteq V$ *are called* cut slices *if and only if they are a result of a guillotine cut of* U*.*

The cutting of V *is done recursively in stages. In each stage all cuts splitting a cut slice must be parallel. A cut slice* U *is in the* k*'*th stage *if it has depth* k *in the recursion tree. A packing* p *of the set* V *is a* k*-stage guillotine packing if and only if it can be split into* $|V|$ *singleton sets in* k *stages. If there are no restrictions on* k *we just say that* p *is a* guillotine packing*.*

Figure 2 shows examples of guillotine cuts for $d = 2$ and $d = 3$ and Figure 1 compare a guillotine cuttable packing with a non-guillotine cuttable one.



(a) 4 stage guillotine cut for $d = 2$

(b) 3 stage guillotine cut for $d = 3$

Figure 2: Example of guillotine cutable packings. The numbers indicate one out of many feasible orderings of the cut slices. Cut number 4 in 2(b) is hidden in the back. It splits the grey cut slice.

The literature sometimes distinguish between if *trimming* is allowed after the recursive guillotine cutting or not. If trimming is allowed, the recursive cutting may result in pieces being larger than the box they contain. If trimming is not

allowed, the recursive cutting must result in pieces with the exact size of the box they contain. In this text we always allow trimming.

We will sometimes refer to a subset of guillotine cuttable packings named *robot packing*:

Definition 2.4 (Robot packing) *A* robot packing *is a packing which can be achieved by placing the first box v_0 in $p_i(v) = 0$ for all i, and successively all other boxes v' so that there exists an i for which all previous boxes v satisfy $p_i(v) < p_i(v')$.*

Robots used for packing boxes in the industry are equipped with a rectangular "hand" covered with vacuum cells for lifting the boxes. To avoid collisions, it is demanded that no already packed box is positioned in front of, right of, or above the destination of the current box. Each guillotine cuttable packing is also a robot packing: consider indeed a guillotine cuttable packing and a feasible sequence (tree) of cuts. By first packing the items in the bottom, left or back part (depending on the cutting direction) of each cut, a feasible robot packing is obtained.

The subproblem of OPP-2 where packings are required to be guillotine cuttable is strongly NP-complete. This is easily seen by a reduction from the strongly NP-hard [GJ75] 3-partition problem: Let $S = \{w_1, \ldots, w_n\}$ be the set of n weights summarizing to $W = \sum w_i$. The 3-partition problem is the task of finding three disjoint subsets $S_1, S_2, S_3 \subset S$ with $S_1 \cup S_2 \cup S_3 = S$ where

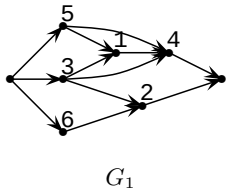
$$\sum_{w_i \in S_1} w_i = \sum_{w_j \in S_2} w_j = \sum_{w_l \in S_3} w_l = \frac{W}{3}.$$

Consider an OPP-2 instance with n boxes of size $(w_1, 1), \ldots, (w_n, 1)$ and a bin of size $(\frac{1}{3}W, 3)$ where $W = \sum w_i$. If an OPP-2 solution can be found for this instance it will be a filling of the bin by three rows, each being 1 unit high and $\frac{1}{3}W$ units wide. The packing is clearly guillotine cuttable as the two rows can be separated in the first cut stage and each of them sliced up afterwards. Solving the OPP with the guillotine restriction for this instance also solves the 3-partition problem for the instance mentioned above. Similarly if the 3-partition problem is solved, the solution is equivalent to a guillotine restricted OPP solution similar to the one described above. As the reduction is polynomial, OPP-2 is strongly NP-complete.

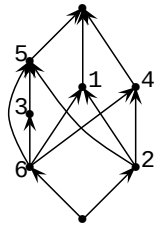
3 Constraint-programming based approach

Pisinger and Sigurd [PS05, PS07] showed how to solve two-dimensional packing problems with guillotine constraints through constraint programming. In the following we will generalize the developed techniques to more dimensions, and present some more general approaches for testing whether a packing is guillotine cuttable.

r_{uv}	1	2	3	4	5	6
1		A	R	L	R	A
2			R	U	U	R
3				L	U	A
4					R	A
5						A
6						



G_1



G_2

Figure 3: A 2-dimensional packing given by relation r, and the corresponding graphs G_1 and G_2.

The algorithm operates with relations $r : V \times V \mapsto \{0, \pm 1, \ldots, \pm d\}$, defining the relative position of two boxes $u, v \in V$. The relation matrix r is skew-symmetric, i.e. $r_{uv} = -r_{vu}$ and hence $r_{uu} = 0$. If $r_{uv} = -k$ it means that box u is located "left" to box v in coordinate k. If $r_{uv} = k$ it means that box u is located "right" to box v in coordinate k. If $r_{uv} = 0$ for $u \neq v$ then the relative position of boxes u, v has not been settled yet. More formally we have

$$\begin{array}{lcl} r_{uv} = -k & \Rightarrow & p_k(u) + w_k(u) \leq p_k(v) \\ r_{uv} = k & \Rightarrow & p_k(v) + w_k(v) \leq p_k(u) \end{array} \tag{3}$$

Now the task of finding a packing is to assign values $\{\pm 1, \ldots, \pm d\}$ to the relations $r_{uv}, u, v \in V, u \neq v$ such that equation (3) is satisfied, and the coordinate function p is a packing, i.e. satisfies (1) and (2). For problems with dimension $d \leq 3$ we may interpret the values $\{\pm 1, \ldots, \pm d\}$ as

-1	'left' (L)	-2	'under' (U)	-3	'in front' (F)
1	'right' (R)	2	'above' (A)	3	'behind' (B)

Given a relation r we may find a corresponding coordinate function p as follows. For each dimension $k = 1, \ldots, d$ construct an oriented graph $G_k = (V, E_k)$ defined by the relation

$$\begin{array}{lcl} (u, v) \in E_k & \Leftrightarrow & r_{uv} = -k \\ (v, u) \in E_k & \Leftrightarrow & r_{uv} = k \end{array} \tag{4}$$

We will call these graphs *relation graphs* (see Figure 3 for an example). If a graph G_k contains a cycle, then obviously r cannot define a feasible packing. For each graph G_k we now solve a critical path problem with w_k as edge lenth, finding the earliest start d_u^k of each node u. The coordinates of each node u are then set to $p_k(u) = d_u^k$. If some box u by these calculations exceeds the bin size, i.e. in some dimension k, $p_k(u) + w_k(u) > W$, then the relation r does not define a feasible packing.

The framework by Pisinger and Sigurd [PS05] is based on a depth-first search, where in each iteration a single relation r_{uv} is assigned a value in $\{\pm 1, \ldots, \pm d\}$. The boxes are initially ordered according to decreasing volume, such that relations between the largest boxes are fixed first before considering the smaller boxes.

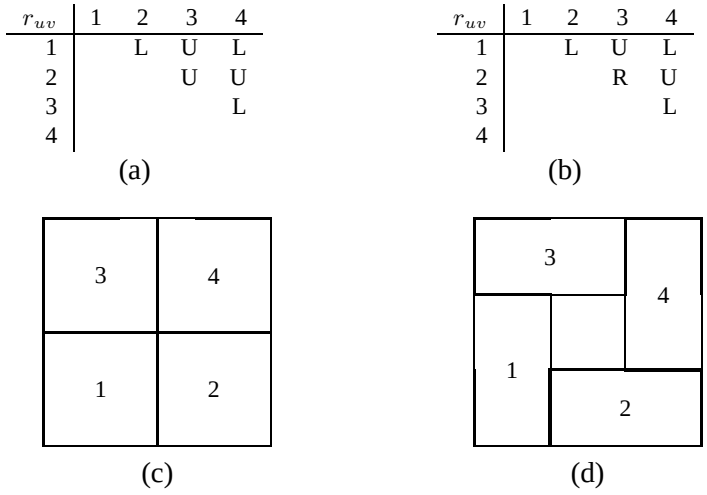


Figure 4: Two 2-dimensional relations r_{uv} shown in (a) and (b), and two packings shown in (c) and (d). Relation r_{uv} in (a) can be realized by packing (c) and (d) depending on the dimensions of the rectangles. The first packing is guillotine cuttable, while the second is not. Note that the relation r_{uv} in (a) does not define a strongly guillotine cuttable packing. Relation r_{uv} in (b) is strongly guillotine cuttable, and it can only be realized by packing (c).

With each pair of boxes u, v we associate a set M_{uv} of feasible values of r_{uv}. Initially $M_{uv} = \{\pm 1, \ldots, \pm d\}$. If $M_{uv} = \emptyset$ then we backtrack. If M_{uv} contains only one value then r_{uv} is fixed to this value. If all relations $r_{uv} \neq 0$ and the corresponding packing p is feasible, we terminate.

In each step of the search constraint propagation is used to fathom infeasible values r_{uv} from M_{uv}. This is done by repeatedly fixing r_{uv} to each value in M_{uv} and testing whether the corresponding packing p is feasible. If the packing is infeasible, we remove the value from M_{uv}.

In order to extend the framework to handle guillotine packings, we need to ensure that the defined relation r at any step defines a guillotine packing, and that all currently undefined relations can be extended to a feasible guillotine packing.

Definition 3.1 (Strongly guillotine cuttable) *A relation function r represents a* strongly guillotine cuttable *packing if either:*

- *The set of boxes V contains one box only.*
- *We can separate V into two subsets V_1 and V_2 and there exists a $k \in \{\pm 1, \ldots, \pm d\}$ such that*

$$r_{uv} = k \text{ for all } u \in V_1, v \in V_2 \tag{5}$$

and both sets V_1 and V_2 are again strongly guillotine cuttable.

Note that the relation r may represent a guillotine cuttable packing even if it is not strongly guillotine cuttable (see Figure 4). However, for every guillotine cuttable pattern, there exists an equivalent strongly guillotine cuttable representation.

Definition 3.2 (Candidate strongly guillotine cuttable) *A set M of feasible relations represents a* candidate strongly guillotine cuttable *packing if either:*

- *The set of boxes V contains one box only.*
- *We can separate V into two subsets V_1 and V_2 and there exists a $k \in \{\pm 1, \ldots, \pm d\}$ such that*

$$k \in M_{uv} \text{ for all } u \in V_1, v \in V_2 \tag{6}$$

and both sets V_1 and V_2 are again candidate strongly guillotine cuttable.

Testing whether a relation r represents a strongly guillotine cuttable packing may be done by using a *greedy approach*. Find any strong guillotine cut, separate the problem into two subsets V_1 and V_2 and call the testing algorithm recursively. The greedy approach works since choosing a specific cut will not block for any alternative cuts.

3.1 Finding a strong guillotine cut in a complete graph

Assume that no relations r_{uv} are undefined, meaning that

$$G = (V, E_1 \cup \cdots \cup E_d)$$

is a complete graph. Finding a strong guillotine cut in dimension k is done as follows.

Algorithm 3.3 *Construct the directed graph $G_k = (V, E_k)$ defined by relation* (4)*. Since G_k is acyclic, we may sort the nodes V in topological order (this order is not necessarily unique). Assuming that vertex v is the first node in the topological order, start with $V_1 = \{v\}$ and $V_2 = V \setminus \{v\}$. If there exists a k for which the two sets define a strong guillotine cut, we are done. Otherwise move the next node in the topological order from V_2 to V_1, and test. Repeat until $V_2 = \emptyset$ in which case no strong guillotine cut exists.*

Theorem 3.4 *V_1 and $V_2 = V \setminus V_1$ define a strong guillotine cut of V if and only if $V_2 \neq \emptyset$ upon termination of Algorithm 3.3.*

PROOF ($\Rightarrow$) Assume that V_1 and $V_2 = V \setminus V_1$ define a strong guillotine cut. That is, $r_{uv} = k$ for all $u \in V_1, v \in V_2$. The topological sorting will therefore order all vertices in V_1 before all vertices in V_2 (else there would be an edge from V_2 to V_1) and at some point, the algorithm will thus stop when V has been split into V_1 and V_2.

($\Leftarrow$) Assume that $V_2 \neq \emptyset$ upon algorithm termination. V_1 and V_2 will then each be strongly guillotine cuttable so we need to show that V_1 and V_2 define a strong guillotine cut of V. That is, we have to show that there exists a k for which $r_{uv} = k$ for all $u \in V_1$ and $v \in V_2$. But this is trivially true if the algorithm terminated before $V_2 = \emptyset$. □

The running time of the algorithm is $O(n^3)$ since we consider n subsets V_1, V_2, and each time check criteria (5) in time $O(n^2)$.

We can decrease the running time to $O(n^2)$ by using Algorithm 3.5, which tests for a strong guillotine cut in dimension k.

Algorithm 3.5 *Set $m = 1$ and let V_1 and V_2 define a disjoint division of V with $V_1 = \{1, \ldots, m\}$ and $V_2 = \{m+1, \ldots, n\}$. Perform the following actions for each $u \in \{1, \ldots, n-1\}$: First, for $v \in \{m+1, \ldots, n\}$, set $m = v$ iff $r_{uv} \neq k$. Next, if $m = u$ then the pair (V_1, V_2) defines a strong guillotine cut. Else, if $m = n$ then no strong guillotine cut exists.*

Initially we have $V_1 = \{1\}$ and $V_2 = \{2, \ldots, n\}$. If $r_{uv} = k$ for all $u \in V_1$ and $v \in V_2$ we have a strong guillotine cut. Otherwise $r_{uv} \neq k$ for some $v \in V_2$ and hence no separation where $u \in V_1$ and $v \in V_2$ will be strongly guillotine cuttable. So we proceed to the case where $u, v \in V_1$, i.e. testing $V_1 = \{1, \ldots, v\}$ and $V_2 = \{v+1, \ldots, n\}$.

3.2 Nonexistence of strong guillotine cuts in arbitrary graphs

If some of the relations r_{uv} are undefined we cannot use the approach from the previous section, as a topological ordering of the nodes is not sufficient to define the strong guillotine cut. Instead, we may look for an efficient algorithm which detects non-existence of a strong guillotine cut in arbitrary constraint graphs.

Assume that each pair of boxes has associated the set M_{uv} of feasible relative placements. Testing whether a, by r induced, packing is guillotine cuttable in dimension $k \in \{1, \ldots, d\}$ is done as follows:

1. Calculate new sets $M'_{uv} := M_{uv} \cap \{-k, k\}$.
2. If for two nodes u and v we have that $M'_{uv} = \emptyset$ then merge the two nodes into a node u'. For every other node w let $M_{u'w} = M_{uw} \cap M_{vw}$.
3. Repeat the process until no more nodes can be merged.

If all nodes have been merged into one node, then the current solution will never be strongly guillotine cuttable, as shown in the following theorem:

Theorem 3.6 *A sufficient criterion for a strong guillotine cut* not *to exist in dimension k is that the above process results in a singleton node set.*

PROOF We show that if a strong guillotine cut V_1, V_2 exists in dimension k then the above process will result in a node set of cardinality at least 2. For a strong guillotine cut V_1, V_2 in dimension k we have, per definition, $r_{uv} = -k$ or $r_{uv} = k$ for $u \in V_1, v \in V_2$. Therefore either $-k \in M_{uv}$ for all $u \in V_1, v \in V_2$ or $k \in M_{uv}$ for all $u \in V_1, v \in V_2$. Consequently $M'_{uv} = M_{uv} \cap \{-k, k\} \neq \emptyset$ for all $u \in V_1, v \in V_2$.

We now have to show, that merging any two nodes in V_1 or V_2 will not make $M'_{uv} = \emptyset$ for any $u \in V_1, v \in V_2$. Without loss of generality, we may assume that $u \in V_1$ and $z, w \in V_2$ and that both $k \in M_{uw}$ and $k \in M_{uz}$. Merging z and w results in a new node z' with $M_{uz'} = M_{uw} \cap M_{uz}$ and thus also $k \in M_{uz'}$. Trivially $M'_{uz'} = M_{uz} \cap \{-k, k\} \neq \emptyset$.

No node $u \in V_1$ will therefore ever be merged to a node $v \in V_2$ and thus the resulting cardinality of the node set will be at least 2. □

Since there are $n(n-1)/2$ uv-pairs and the cardinality of M_{uv} is $O(1)$, step 1 in the above test can be done in $O(n^2)$ time. If merging two sets has complexity $O(n)$ each iteration takes $O(n^3)$ time. At most n iterations are needed so the total complexity is $O(n^4)$.

3.3 Finding strong guillotine cuts in almost complete graphs

As we fix the relations between the boxes $1, \ldots, i-1$ before proceeding to the next box i, a typical situation will be as follows: For a given box i we have

$$
\begin{aligned}
&r_{uv} \in \{\pm 1, \ldots, \pm d\} \text{ for } u, v = 1, \ldots, i-1 \\
&r_{ui} \in \{0, \pm 1, \ldots, \pm d\} \text{ for } u = 1, \ldots, i-1 \\
&r_{uv} = 0 \text{ for } u = 1, \ldots, i-1, v = i+1, \ldots, n
\end{aligned}
$$

In other words, all relations between boxes $1, \ldots, i-1$ are fully fixed, while relations involving boxes $i+1, \ldots, n$ are unsettled. Relations involving box i are partially settled. The constraint graph G_k defined on nodes $1, \ldots, i$ is in other words almost complete, as illustrated in Figure 5.

In this situation we may run Algorithm 3.3 for boxes $1, \ldots, i-1$ testing whether a *candidate* strong guillotine cut exists. Each time we test whether the partitioning of the boxes into (V_1, V_2) is candidate strongly guillotine cuttable, we try to add box i to each of the sets in turn, i.e. we test if $(V_1 \cup \{i\}, V_2)$ is candidate strongly guillotine cuttable, or if $(V_1, V_2 \cup \{i\})$ is candidate strongly guillotine cuttable. The running time will be increased by a factor two, which is a constant factor.

An alternative approach for testing strong guillotine cuts in almost complete graphs is to ignore undefined relations with $r_{uv} = 0$, and run Algorithm 3.3 for boxes $1, \ldots, i$. To see correctness of this approach, assume that a strong guillotine cut exists in dimension k, meaning that we can separate the set of boxes $\{1, \ldots, i\}$ into V_1, V_2, such that

$$
(r_{uv} = k \vee r_{uv} = 0) \text{ for all } u \in V_1, v \in V_2 \tag{7}
$$

M_{uv}	1	2	3	4	5
1		{A}	{L}	{L}	{L}
2			{L}	{L}	{L,R,A}
3				{A}	{L,U}
4					{R,U,A}
5					

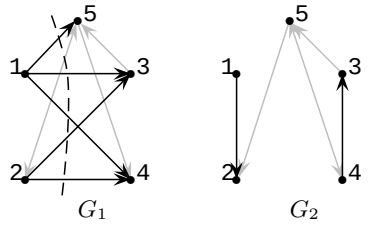


Figure 5: A 2-dimensional packing given by the sets M and the corresponding graphs G_1 and G_2. Dark edges (u, v) indicate that only one relation is possible in M_{uv} while light edges indicate alternative relations. $G = (V, E_1 \cup E_2)$ is complete for the node subset $\{1, 2, 3, 4\}$. A topological ordering of the nodes in G_1 according to relation r may lead to orderings $1, 2, 5, 3, 4$ or $1, 5, 2, 3, 4$. The first-mentioned will be found candidate strongly guillotine cuttable by Algorithm 3.3, while the latter will not.

Now assume that Algorithm 3.3 fails finding the separation V_1, V_2. If it was not for the last box, i, we know that Algorithm 3.3 would have found the two sets. Assume (for symmetry reasons) that i was placed in V_1 but a node v exists in V_2 such that $r_{vi} = k$. But this means that the topological ordering of the nodes in Algorithm 3.3 was not correct.

4 Computational results

The experiments presented in this section aim at demonstrating three things: 1) our implementation works properly, 2) guillotine constraints can be handled effectively, and 3) the quality of solutions generally is not affected by requiring guillotine cuttability. To the best of our knowledge, this is the first comparison of OBPP-3 displaying the costs of the addional constraints.

We only cover results for $d = 3$ as displaying results for higher-dimensional problems would not contribute with additional information. Furthermore, to the best of our knowledge, there are very few OBPP-3 solvers publicly available, and most such solvers are built on top of an OPP-3 algorithm, making them difficult to use for comparison. Therefore, the results below do not compare our solver with other solvers.

Based on the codes [PS05, PS07, MPV$^+$07] we have implemented all of the pruning techniques described in Section 3.1 to 3.3. After some preliminary tuning we have chosen to use the alternative approach, described in the last part of Section 3.3, for testing whether a partial solution is guillotine cuttable.

The code was implemented in ANSI-C, and the experiments were run on a Pentium 4 processor running at 3GHz. A time limit of 3600 seconds was imposed to each instance. We will test the developed d-dimensional guillotine packing algorithm on three-dimensional instances presented by Martello et al. [MPV00]. For the first five classes, the bin size is $W = H = D = 100$ and five types of boxes are

packing	n	Class 1	2	3	4	5	6	7	8	9
general	10	10	10	10	10	10	10	10	10	10
	15	10	10	10	10	10	10	10	10	10
	20	10	10	10	10	10	10	10	10	10
	25	10	10	10	10	10	10	10	10	10
	30	10	10	10	10	10	10	10	9	9
	35	10	10	10	10	10	10	8	9	9
	40	10	9	8	10	9	10	7	9	3
	45	8	6	4	10	6	9	5	9	0
	50	7	3	7	10	5	10	6	9	0
total: 723		85	78	79	90	80	89	76	85	61
robot	10	10	10	10	10	10	10	10	10	10
	15	10	10	10	10	10	10	10	10	10
	20	10	10	10	10	10	10	10	10	10
	25	10	10	10	10	8	10	7	9	10
	30	10	10	10	10	9	10	7	7	10
	35	10	10	10	10	5	10	5	8	10
	40	10	9	8	10	4	10	2	10	9
	45	10	7	4	10	4	10	1	8	2
	50	7	4	9	10	3	10	3	5	0
total: 694		87	80	81	90	63	90	55	77	71
guillotine	10	10	10	10	10	10	10	10	10	10
	15	10	10	10	10	10	10	10	10	10
	20	10	10	10	10	10	10	10	10	10
	25	10	10	10	10	10	10	10	10	10
	30	10	10	10	10	10	10	10	10	10
	35	10	10	10	10	10	10	10	10	10
	40	10	9	8	10	10	10	10	10	10
	45	8	7	4	10	9	10	9	10	7
	50	7	3	7	10	10	10	10	10	1
total: 769		85	79	79	90	89	90	89	90	78

Table 1: Number of instances, out of 10, solved to proved optimality in 3600 seconds

packing	n	Class 1	2	3	4	5	6	7	8	9
general	10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	15	0.00	0.00	0.00	0.00	0.01	0.00	0.02	0.00	0.00
	20	0.01	0.01	0.01	0.00	0.01	0.01	0.16	0.00	0.01
	25	0.05	0.04	0.10	0.00	0.09	0.19	1.76	0.91	0.11
	30	15.18	1.09	3.50	0.00	1.41	7.34	93.73	360.35	468.81
	35	61.07	102.39	26.30	0.00	3.01	2.90	764.97	390.11	546.46
	40	217.87	614.06	861.15	0.00	448.32	15.79	1151.20	365.25	2859.09
	45	1049.05	1493.49	2163.92	0.01	1464.79	363.07	1952.55	450.54	3600.01
	50	1391.89	2696.83	1712.14	4.00	1971.17	4.22	1604.58	434.98	3600.02
robot	10	0.00	0.00	0.00	0.00	0.00	0.00	7.81	0.00	0.00
	15	0.00	0.00	0.00	0.00	0.04	0.01	4.85	0.02	0.00
	20	0.01	0.01	0.01	0.00	3.96	0.04	207.76	4.31	0.00
	25	0.11	0.03	0.18	0.00	843.37	0.09	1425.30	361.05	0.05
	30	10.41	1.01	1.90	0.00	381.42	5.00	1308.55	1080.11	19.78
	35	36.67	35.75	11.03	0.00	1819.74	1.53	2033.36	950.86	74.75
	40	114.53	449.87	766.76	0.00	2184.86	12.84	2881.79	136.94	1250.15
	45	748.34	1299.86	2162.28	0.01	2161.50	11.28	3240.01	751.35	3247.02
	50	1262.65	2523.43	1333.56	3.52	2532.84	5.49	2742.58	1875.95	3600.01
guillotine	10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	15	0.02	0.00	0.01	0.00	0.01	0.00	0.01	0.05	0.00
	20	0.01	0.02	0.01	0.00	0.01	0.09	0.26	0.02	0.01
	25	0.06	0.05	0.15	0.00	0.11	0.10	1.80	0.61	0.41
	30	13.33	0.96	3.28	0.00	1.67	10.23	5.00	1.46	1.47
	35	42.45	107.74	21.43	0.00	3.59	1.46	13.49	7.52	54.53
	40	152.28	664.45	844.57	0.01	89.86	3.21	27.43	29.37	762.25
	45	998.11	1473.31	2163.11	0.02	419.69	3.62	648.31	8.69	2024.80
	50	1331.28	2663.01	1727.20	3.91	45.06	13.46	15.24	51.58	3283.91

Table 2: Average solution times in seconds, as average of 10 instances using a time limit of 3600 seconds

packing	n	Class 1	2	3	4	5	6	7	8	9
general	10	3.3	3.5	3.6	6.6	2.5	2.9	2.5	2.8	3.0
	15	4.7	4.5	4.8	8.7	2.9	4.4	3.0	3.7	3.0
	20	6.0	6.6	6.0	12.3	3.9	5.4	3.8	4.9	3.0
	25	7.4	7.0	7.1	15.4	4.6	5.9	4.3	5.5	3.0
	30	8.6	8.0	8.6	17.2	5.4	6.7	4.0	6.1	3.1
	35	9.3	9.5	10.3	21.1	6.3	7.7	5.4	7.1	3.2
	40	11.0	10.7	11.5	24.3	6.9	8.8	6.5	7.7	4.7
	45	12.3	12.6	12.2	27.6	7.9	9.8	7.0	8.3	5.0
	50	13.5	13.8	13.4	29.4	9.2	9.8	7.9	9.4	6.2
robot	10	3.3	3.5	3.6	6.6	2.5	2.9	2.5	2.8	3.0
	15	4.7	4.5	4.8	8.7	2.9	4.4	3.0	3.7	3.0
	20	6.0	6.6	6.0	12.3	3.9	5.4	3.8	4.9	3.0
	25	7.4	7.0	7.1	15.4	4.8	5.9	4.7	5.6	3.0
	30	8.6	8.0	8.6	17.2	5.5	6.7	4.6	6.2	3.0
	35	9.3	9.5	10.3	21.1	7.2	7.7	6.0	7.2	3.0
	40	11.0	10.7	11.5	24.3	7.6	8.8	7.7	7.7	3.3
	45	12.2	12.6	12.2	27.6	8.2	9.6	8.1	8.5	4.7
	50	13.5	13.8	13.3	29.4	9.6	9.8	8.8	10.3	6.0
guilllotine	10	3.3	3.5	3.6	6.6	2.5	2.9	2.5	2.8	3.0
	15	4.7	4.5	4.8	8.7	2.9	4.4	3.1	3.7	3.0
	20	6.0	6.6	6.0	12.3	3.9	5.4	3.8	4.9	3.0
	25	7.4	7.0	7.1	15.4	4.6	5.9	4.3	5.5	3.0
	30	8.6	8.0	8.6	17.2	5.4	6.8	4.0	6.0	3.0
	35	9.3	9.5	10.3	21.1	6.3	7.8	5.2	7.0	3.0
	40	11.0	10.7	11.5	24.3	6.6	8.8	6.0	7.8	3.0
	45	12.3	12.6	12.2	27.6	11.0	9.7	6.4	8.4	7.5
	50	13.5	13.8	13.4	29.4	8.3	9.9	7.4	9.3	22.7

Table 3: Upper bound, as average of 10 instances using a time limit of 3600 seconds

packing	n	Class 1	2	3	4	5	6	7	8	9
general	10	3.3	3.5	3.6	6.6	2.5	2.9	2.5	2.8	3.0
	15	4.7	4.5	4.8	8.7	2.9	4.4	3.0	3.7	3.0
	20	6.0	6.6	6.0	12.3	3.9	5.4	3.8	4.9	3.0
	25	7.4	7.0	7.1	15.4	4.6	5.9	4.3	5.5	3.0
	30	8.6	8.0	8.6	17.2	5.4	6.7	4.0	5.8	3.0
	35	9.3	9.5	10.3	21.1	6.3	7.7	4.9	6.9	3.0
	40	11.0	10.5	11.3	24.3	6.6	8.8	5.5	7.6	3.0
	45	12.0	12.1	11.6	27.6	7.0	9.5	5.6	8.2	3.0
	50	13.1	12.9	13.1	29.4	7.9	9.8	7.0	9.1	3.0
robot	10	3.3	3.5	3.6	6.6	2.5	2.9	2.5	2.8	3.0
	15	4.7	4.5	4.8	8.7	2.9	4.4	3.0	3.7	3.0
	20	6.0	6.6	6.0	12.3	3.9	5.4	3.8	4.9	3.0
	25	7.4	7.0	7.1	15.4	4.6	5.9	4.0	5.4	3.0
	30	8.6	8.0	8.6	17.2	5.4	6.7	3.9	5.6	3.0
	35	9.3	9.5	10.3	21.1	6.0	7.7	4.7	6.8	3.0
	40	11.0	10.5	11.3	24.3	5.9	8.8	5.2	7.7	3.0
	45	12.2	12.3	11.6	27.6	6.8	9.6	5.1	8.1	3.0
	50	13.1	13.0	13.2	29.4	7.7	9.8	6.6	8.7	3.0
guillotine	10	3.3	3.5	3.6	6.6	2.5	2.9	2.5	2.8	3.0
	15	4.7	4.5	4.8	8.7	2.9	4.4	3.1	3.7	3.0
	20	6.0	6.6	6.0	12.3	3.9	5.4	3.8	4.9	3.0
	25	7.4	7.0	7.1	15.4	4.6	5.9	4.3	5.5	3.0
	30	8.6	8.0	8.6	17.2	5.4	6.8	4.0	6.0	3.0
	35	9.3	9.5	10.3	21.1	6.3	7.8	5.2	7.0	3.0
	40	11.0	10.5	11.3	24.3	6.6	8.8	6.0	7.8	3.0
	45	12.0	12.3	11.6	27.6	7.2	9.7	6.2	8.4	3.0
	50	13.1	12.9	13.1	29.4	8.3	9.9	7.4	9.3	3.0

Table 4: Lower bound, as average of 10 instances using a time limit of 3600 seconds

considered by uniformly randomly generating the box sizes in different intervals, namely:

Type 1: $w_j \in [1, \frac{1}{2}W]$, $h_j \in [\frac{2}{3}H, H]$, $d_j \in [\frac{2}{3}D, D]$;
Type 2: $w_j \in [\frac{2}{3}W, W]$, $h_j \in [1, \frac{1}{2}H]$, $d_j \in [\frac{2}{3}D, D]$;
Type 3: $w_j \in [\frac{2}{3}W, W]$, $h_j \in [\frac{2}{3}H, H]$, $d_j \in [1, \frac{1}{2}D]$;
Type 4: $w_j \in [\frac{1}{2}W, W]$, $h_j \in [\frac{1}{2}H, H]$, $d_j \in [\frac{1}{2}D, D]$;
Type 5: $w_j \in [1, \frac{1}{2}W]$, $h_j \in [1, \frac{1}{2}H]$, $d_j \in [1, \frac{1}{2}D]$.

Classes k ($k = 1, \ldots, 5$) have then been obtained by generating each box according to type k with probability 60%, and according to the other four types with probability 10% each. Classes 6 to 8 may be described as follows:

Class 6: $W = H = D = 10$; w_j, h_j, d_j uniformly random in $[1, 10]$;
Class 7: $W = H = D = 40$; w_j, h_j, d_j uniformly random in $[1, 35]$;
Class 8: $W = H = D = 100$; w_j, h_j, d_j uniformly random in $[1, 100]$.

Finally, *Class 9* consists of difficult *all-fill* instances having a known solution with three bins: the boxes are generated by randomly cutting the bins into smaller parts as follows. Bins 1 and 2 are cut into $\lfloor n/3 \rfloor$ boxes each, while bin 3 is cut into $n-2\lfloor n/3 \rfloor$ boxes. The cutting is made using a recursive approach which repeatedly cuts the bin into two parts using a guillotine cut until five boxes remain. These boxes are then cut using a non guillotine pattern. This means that if the number of boxes in a bin is not a multiple of 5, then then bin is guillotine-cuttable. Otherwise, it contains a non guillotine pattern, which *possibly* can be made guillotine cuttable by interchanging the boxes.

The best found solver for comparison was a solver by Martello et al. [MPV$^+$07] which solves the general and robot-packable version of the bin packing problem.

Table 1 shows the number of instances solved out of ten for each problem size and instance type. It is seen that in general we are able to solve more guillotine cuttable problems than general packing problems, while the robot-packable problems are the most difficult to solve. Indeed, using the guillotine algorithm, we are able to solve most problems to optimality of size up to $n = 50$.

Table 2 reports the corresponding solution times as average of 10 runs. The solution times are dominated by the instances that time out, but for the instances which are solved by all three algorithms there is no major difference in the times.

Finally Table 3 and 4 present upper and lower bounds for the considered instances. If a problem was solved to optimality the upper and lower bound is equal. Otherwise the upper bound denotes the best solution found when time-out was reached, and the lower bound denotes the corresponding best-known lower bound. It is seen, that imposing the guillotine cut constraint only in very few instances affects the optimal solution in comparison to a general packing.

Class 9 is particularly interesting to study since all optimal solutions are guillotine cuttable. Not surprising the guillotine algorithm is able to solve more instances than the other two approaches since it takes advantage of the smaller solution space.

However, for large instances, the upper bound becomes very bad for the guillotine algorithm. This can be the result of the bin-packing algorithm making infeasible assignments of items to bins. Since the instances in class 9 are trim-free, specialized algorithms can exploit this property to backtrack whenever a gap appears in the packing.

5 Conclusion

We have presented several guillotine tests, which can be used in a constraint programming approach for solving the packing problem. The results are quite promising, as problems with guillotine cut constraint tend to be easier than general packing problems (and robot packing problems). Moreover, it seems that for the considered bin-packing instances, the objective function is affected only very little by imposing the guillotine constraints.

Future challenges would be to develop solvers which automatically respects guillotine constraints in each step. Currently, we solve a general packing problem, where we in each step test for satisfaction of guillotine constraints. This can quite easily be done in the Fekete-Schepers representation [Amo05, CM] but the computational results seem to be disappointing. A similar approach for the constraint programming approach would be interesting.

References

[Amo05] R. R. Amossen. Constructive algorithms and lower bounds for guillotine cuttable orthogonal bin packing problems. Master's thesis, Dept. of Computer Science, University of Copenhagen, 2005.

[BB07] R. Baldacci and M.A. Boschetti. A cutting-plane approach for the two-dimensional orthogonal non-guillotine cutting problem. *European Journal of Operational Research*, 183:1136–1149, 2007.

[BC01] N. Bediceanu and M. Carlsson. Sweep as a generic pruning technique applied to the non-overlapping rectangles constraints. In *Principles and Practice of Constraint Programming (CP'2001)*, volume 2239, pages 377–391. Springer, 2001.

[BCT06] N. Bediceanu, M. Carlsson, and S. Thiel. Sweep synchronization as a global propagation mechanism. *Computers and Operations Research*, 33:2835–2851, 2006.

[Bel03] G. Belov. *Problems, Models and Algorithms in One- and Two-Dimensional Cutting*. PhD thesis, Technischen Universität Dresden, 2003.

[BKRS09] G. Belov, V. Kartak, H. Rohling, and G Scheithauer. One-dimensional relaxations and lp bounds for orthogonal packing. *International Transactions in Operational Research*, 16:745–766, 2009.

[CAdC08] F. Clautiaux, C. Alves, and J. Valerio de Carvalho. A survey of dual feasible and superadditive functions. *Annals of OPerations Research*, 2008.

[CAdC09] F. Clautiaux, C. Alves, and J. Valério de Carvalho. A survey of dual-feasible functions for bin-packing problems. *Annals of Operations Research*, pages 1–26, 2009. doi: 10.1007/s10479-008-0453-8.

[CHC97] V.-D. Cung, M. Hifi, and B. Le Cun. Constrained tow-dimensional cutting stock problems – a best-first branch-and-bound algorithm. Technical Report 97/020, Laboratorie PRISM, University of Versailles, France, 1997.

[CJCM08] F. Clautiaux, A. Jouglet, J. Carlier, and A. Moukrim. A new constraint programming approach for the orthogonal packing problem. *Computers and Operations Research*, 35:944–959, 2008.

[CLS95] 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.

[CM] F. Clautiaux and A. Moukrim. On graph-theoretical models for k-dimensional guillotine-cutting problems. 3rd ESICUP meeting, Porto, Portugal, March 16–18, 2006.

[CW77] N. Christofides and C. Whitlock. An algorithm for two dimensional cutting problems. *Operations Research*, 25:30–44, 1977.

[DW60] G.B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8:101–111, 1960.

[FS97] S. P. Fekete and J. Schepers. On higher-dimensional packing I: Modeling. Technical Report 97–288, University at zu Köln, 1997.

[FS04a] S. Fekete and J. Schepers. A combinatorial characterization of higher-dimensional orthogonal packings. *Mathematics of Operations Research*, 29:353–368, 2004.

[FS04b] S. P. Fekete and J. Schepers. A combinatorial characterization of higher-dimensional orthogonal packing. *Math. Oper. Res.*, 29(2):353–368, 2004.

[FSdV07] S. Fekete, J. Schepers, and J. Van der Veen. An exact algorithm for higher-dimensional orthogonal packings. *Operations Research*, 55:569–587, 2007.

[GJ75] M Garey and D Johnson. Complexity results for multiprocessor scheduling under resource constraints. *SIAM Journal on Computing*, Jan 1975.

[Hif97] M. Hifi. An improvement of Viswanathan and Bagchi's exact algorithm for constrained two-dimensional cutting stock. *Computers and Operations Research*, 24:727–736, 1997.

[Hif98] M. Hifi. Exact algorithms for the guillotine strip cutting/packing problem. *Computers and Operations Research*, 25:925–940, 1998.

[MPV00] S. Martello, D. Pisinger, and D. Vigo. The three-dimensional bin packing problem. *Operations Research*, 48:256–267, 2000.

[MPV+07] S. Martello, D. Pisinger, D. Vigo, E. den Boef, and J. Korst. Algorithm 864: General and robot-packable variants of the three-dimensional bin packing problem. *ACM Transactions on Mathematical Software*, 33:7 pp, 2007.

[PPSG00] V. Parada, R. Palma, D. Sales, and A. Gómes. A comparative numerical analysis for the guillotine two-dimensional cutting problem. *Annals of Operations Research*, 96:245–254, 2000.

[PS05] D. Pisinger and M.M. Sigurd. The two-dimensional variable-sized bin packing problem. *Discrete Optimization*, 2:154–167, 2005.

[PS07] D. Pisinger and M.M. Sigurd. Using decomposition techniques and constraint programming for solving the two-dimensional bin packing problem. *INFORMS Journal on Computing*, 19:36–51, 2007.

[Sch92] G. Scheithauer. Algorithms for the container loading problem. *Operations Research Proceedings 1991*, pages 445–452, 1992.

[VB93] K.V. Viswanathan and A. Bagchi. Best-first search methods for constrained two-dimensional cutting stock problem. *Operations Research*, 41:768–776, 1993.

[Wan83] P.Y. Wang. Two algorithms for constrained two-dimensional cutting stock problems. *Opererations Research*, 31:573–586, 1983.

[WL86] D. F. Wong and C. L. Liu. A new algorithm for floorplan design. In *DAC '86: Proceedings of the 23rd ACM/IEEE conference on Design automation*, pages 101–107, Piscataway, NJ, USA, 1986. IEEE Press.