A New Exact Algorithm for General Orthogonal D-Dimensional Knapsack Problems

Sándor P. Fekete, Jörg Schepers*

Center for Parallel Computing, Universität zu Köln
D–50923 Köln, GERMANY
[fekete,schepers]@zpr.uni-koeln.de

Abstract. The *d-dimensional orthogonal knapsack problem* (OKP) has a wide range of practical applications, including packing, cutting, and scheduling. We present a new approach to this problem, using a graph-theoretical characterization of feasible packings. This characterization allows us to deal with classes of packings that share a certain combinatorical structure, instead of single ones. Combining the use of this structure with other heuristics, we develop a two-level tree search algorithm for finding exact solutions for the d-dimensional OKP. Computational results are reported, including optimal solutions for all two–dimensional test problems from recent literature.

1 Introduction

The problem of cutting a rectangle into smaller rectangular pieces of given sizes is known as the *two–dimensional cutting stock problem*. It arises in many industries, where steel, glass, wood, or textile materials are cut, but it also occurs in less obvious contexts, such as machine scheduling or optimizing the layout of advertisements in newspapers. The three-dimensional problem is important for practical applications as container loading or scheduling with partitionable resources. It can be thought of as packing boxes into a container. We refer to the generalized problem in $d \geq 2$ dimensions as the *d-dimensional orthogonal knapsack problem (OKP-d)*. Being a generalizition of the bin packing problem, the OKP-d is $\mathcal{NP}$-complete in the strict sense. (Note that we consider the constrained problem, where bounds on the number of rectangles of each size are imposed.) The vast majority of work done in this field refers to a restricted problem, where only so–called *guillotine patterns* are permitted. This constraint arises from certain industrial cutting applications: guillotine patterns are those packings that can be generated by applying a sequence of edge-to-edge cuts. The recursive structure of these patterns makes this variant much easier to solve than the *general* or *non–guillotine* problem.

Relatively few authors have dealt with the exact solution of the non–guillotine problem. All of them focus on the problem in two dimensions. Biró and Boros

* Supported by the German Federal Ministry of Education, Science, Research and Technology (BMBF, Förderkennzeichen 01 IR 411 C7).

(1984) [4] gave a characterization of non–guillotine patterns using network flows but derived no algorithm. Dowsland (1987) [6] proposed an exact algorithm for the case that all boxes have equal size. Arenales and Morabito (1995) [1] extended an approach for the guillotine problem to cover a certain type of non–guillotine patterns. So far, only two exact algorithms have been proposed and tested for the general case. Beasley (1985) [2] and Hadjiconstantinou and Christofides (1995) [12] have given different 0–1 integer programming formulations of this problem. Even for small problem instances, they have to consider very large 0–1 programs, since the number of variables depends on the size of the container that is to be packed. The largest instance that was solved in either article has 9 out of 22 boxes packed into a 30×30 container. After an initial reduction phase, Beasley gets a 0-1 program with more than 8000 variables and more than 800 constraints; the program by Hadjiconstantinou and Christofides still contains more than 1400 0–1 variables und over 5000 constraints. From Lagrangean relaxations, they derived upper bounds for a branch–and–bound algorithm, which are improved using subgradient optimization. The process of traversing the search tree corresponds to the iterative generation of an optimal packing.

In this paper, we describe a different approach to characterizing feasible packings and constructing optimal solutions. We use a graph-theoretic characterization of the relative position of the boxes in a feasible packing. This allows us a much more efficient way to construct an optimal solution for a problem instance: combined with good heuristics for dismissing infeasible subsets of boxes, we develop a two-level tree search. This exact algorithm has been implemented; it outperforms previous methods by a wide margin.

The rest of this paper is organized as follows: after describing the OKP-*d* and the related *d-dimensional orthogonal packing problem (OPP-d)*, we introduce the novel concepts of packing classes (Section 2.2) and conservative scales (Section 3). Section 4 contains a solution strategy for the OPP-*d*. This method is used as a building block for the exact OKP-*d* algorithm presented in Section 5. We conclude the paper by reporting computational results for OKP-2 test problems from literature as well as new test instances.

2 Problem formulation and mathematical approach

Suppose we are given a finite set V of d-dimensional rectangular boxes with "sizes" $w(v) \in \mathbb{R}_0^{+d}$ and "values" $c(v) \in \mathbb{R}_0^+$ for $v \in V$. When arranging these boxes in a container C, we have to preserve the orientations of the boxes; this constraint usually arises from considerations for stability ("this side up") in packings, from asymmetric texture of material in cutting-stock problems, or from different types of coordinates in scheduling problems. The objective of the *d-dimensional orthogonal knapsack problem (OKP-d)* is to maximize the total value of a subset $V' \subseteq V$ fitting into the container C and to find a complying packing. Closely related is the *d-dimensional orthogonal packing problem (OPP-d)*, which is to decide whether a given set of boxes B fits into a unit size container, and to find a complying packing whenever possible.

2.1 Modeling the Problem

The hardness of finding optimal solutions for an OKP instance is compounded by the difficulty of giving a useful problem formulation: once we have placed a box in a container, the remaining feasible space is no longer convex. As a consequence, standard methods for integer program modeling are inedequate. Previous attempts to overcome this difficulty have used a discretization of the set of possible box positions; the number of variables (basically one 0-1 variable for each combination of a grid position and a box type) in the resulting integer program is exponential in the size of the input coordinates. Consequently, the problem instances for which Beasley [2], and Hadjiconstantinou and Christofides [12] were able to use this approach are relatively small. It seems unlikely that even moderately sized three-dimensional problems can be solved with this method.

In the following, we will describe a different way of modeling feasible packings. The basic idea is to use the combinatorial information induced by relative box positions. For this purpose, we consider projections along the different coordinate axes; overlap in these projections defines an interval graph for each coordinate. Using properties of interval graphs and additional conditions, we can tackle two fundamental problems of exact enumeration algorithms:

1. How can we prove in short time that a particular subset of boxes is infeasible for packing?
2. How do we avoid treating equivalent cases more than once?

2.2 Packing classes

Instead of dealing with single packings we handle classes of packings that share a certain combinatorial structure. This structure arises from the way different boxes in a packing can "see" each other orthogonal to one of the coordinate axes. (So-called *box visibility graphs* have been considered as means for representing graphs, see [7].) We will see that transitive orientations of certain classes of graphs correspond to possible packings if and only if specific additional conditions are satisfied. This allows us to make use of a number of powerful theorems [11, 14] to prune our branch-and-bound tree.

For a d-dimensional packing, consider the projections of the boxes onto the d coordinate axes x_i. Each of these projections induces a graph G_i: two boxes are adjacent in G_i, if and only if their x_i projections overlap. (See Figure 1 for a two-dimensional example.) By definition, the G_i are interval graphs, thus they have algorithmically useful properties. (See [11, 14].)

A set of boxes $S \subseteq V$ is called *x_i–feasible*, if the boxes in S can be lined up along the x_i–axis without exceeding the x_i-width of the container. Then we get the following straightforward statements:

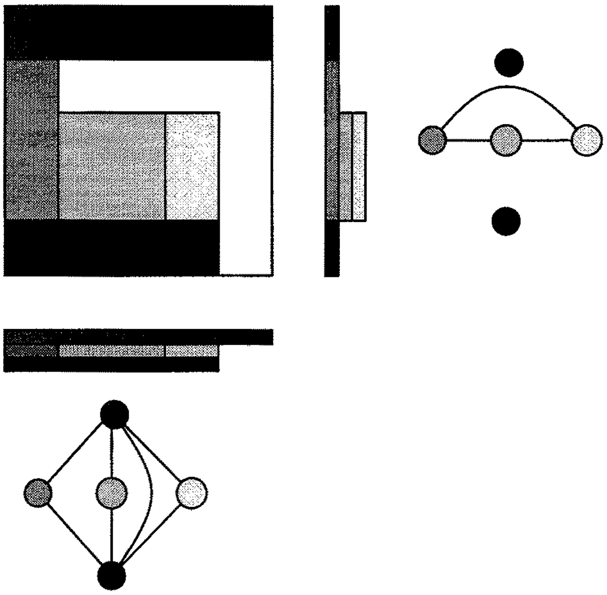

Fig. 1. A two-dimensional packing pattern E and the interval graphs G_1, G_2 induced by the projections

Theorem 1. *Given a packing, then the graphs $G_i = (V, E_i), i = 1, \ldots, d$ as defined have the following properties:*

$$P1: \text{ the graphs } G_i := (V, E_i) \text{ are interval graphs.}$$

$$P2: \text{ each stable set } S \text{ of } G_i \text{ is } x_i\text{-feasible.}$$

$$P3: \bigcap_{i=1}^{d} E_i = \emptyset.$$

A set $E = (E_1, \ldots, E_d)$ of edges is called a *packing class* for (V, w), if and only if it satisfies the conditions $P1$, $P2$, $P3$. For any given $G_i = (V, E_i)$, we denote by G_i^C the complement graph for G_i. If G_i arises from a packing, any edge in G_i^C corresponds to two boxes with non-overlapping x_i-projection, hence we can think of G_i^C as the comparability graph of the boxes in direction x_i. For any packing class E, we will consider transitive orientations F_i of the comparability graph ${G_i}^C$, and call $F = (F_1, \ldots, F_d)$ a *transitive orientation* of the packing class E. In the following, we will show that for any transitive orientation F of a packing class, there is a packing. For this purpose, define a mapping $p_i^F : V \to {\mathbb{R}_0^+}^d$ by

$$p_i^F(v) := \max\{p_i(u) + w_i(u) | \vec{uv} \in F_i\} \tag{1}$$

for $v \in V$, $i \in \{1, \ldots, d\}$.

Without loss of generality, it is sufficient to consider only packings that are generalized "bottom-left" justified, that is any lower bounding coordinate of a box is at zero or at the upper bounding coordinate of another box (without requiring the two boxes to touch). More formally, we say that a packing has *solid projections*, if

$$\forall i \in \{1,\ldots,d\}\ \forall v \in V: \quad p_i(v) = 0 \quad \vee \quad \exists u \in V: p_i(v) = p_i(u) + w_i(u).$$

Then we have the following theorem:

Theorem 2. *A mapping $p : V \to \mathbb{R}_0^{+d}$ is a packing with solid projections, iff there is a packing class E with a transitive orientation F, such that $p = p^F$.*

Proof. First consider a packing p with solid projections. Let F be the canonical orientation of p defined for any $i = \{1,\ldots,d\}$ by:

$$F_i = \{\vec{uv} \,|\, p_i(u) + w_i(u) \le p_i(v)\}\,.$$

Note that F is a transitive orientation of the packing class obtained from the G_i as described above. We show $p = p^F$ by induction over the number of elements in the partial order F_i. Let $v \in V$. If v is the only element in F_i, hence a minimal element of the partial order F_i, then $p_i(v) = 0$ by definition and the claim holds. Therefore consider $p_i(v) > 0$ and assume with the induction hypothesis that for all u with $\vec{uv} \in F_i$, we have $p_i(u) = p_i^F(u)$. Let t be the box where the maximum of p_i^F is attained. Since $\vec{tv} \in F_i$, we have

$$p_i^F(v) \;=\; p_i^F(t) + w_i(t) \;=\; p_i(t) + w_i(t) \;\le\; p_i(v).$$

Since p has solid projections, there must be a t' with $p_i(v) = p_i(t') + w_i(t')$. Then $\vec{t'v} \in F_i$, hence

$$p_i(v) \;=\; p_i(t') + w_i(t') \;=\; p_i^F(t') + w_i(t) \;\le\; p_i^F(v).$$

To prove the converse, consider a packing class E with a transitive orientation F. Place all boxes v at the positions given by $p^F(v)$. To show that p^F is a packing, we have to show that

1. All boxes are within the boundaries of the container.
2. No two boxes overlap.

Let $v \in V, i \in \{1,\ldots,d\}$. By construction of p^F, the circle-free digraph (V, F_i) contains a directed path $(v^{(0)},\ldots,v^{(r)})$ with $\deg_{\text{in}}(v^{(0)}) = 0$ and $v^{(r)} = v$, such that

$$p_i^F(v) = \sum_{k=0}^{r-1} w_i(v^{(k)}).$$

Since F_i is transitive, $S := \{v^{(0)},\ldots,v^{(r)}\}$ induces a clique in G_i^C, implying that S is a stable set in G_i. Hence by condition $P2$

$$p_i^F(v) + w_i(v) = \sum_{k=0}^{r} w_i(v^{(k)}) \le 1, \text{ implying 1.}$$

To see 2., consider $u, v \in V$, $u \neq v$. By $P3$, there is an $i \in \{1, \ldots, d\}$ with the (undirected) edge $\overline{uv} \notin E_i$, hence $\overline{uv} \in {E_i}^C$. Since F_i is an orientation of E_i^C, either $\vec{uv} \in F_i$ or $\vec{vu} \in F_i$. By construction of p^F, we get

$$p_i^F(v) \geq p_i^F(u) + w_i(u) \quad \vee \quad p_i^F(u) \geq p_i^F(v) + w_i(v).$$

In both cases, u and v can be separated by an x_i-orthogonal hyperplane. Hence, the boxes u and v are disjoint and p^F describes a packing. By construction of p^F, the packing has solid projections.

This completes the proof. □

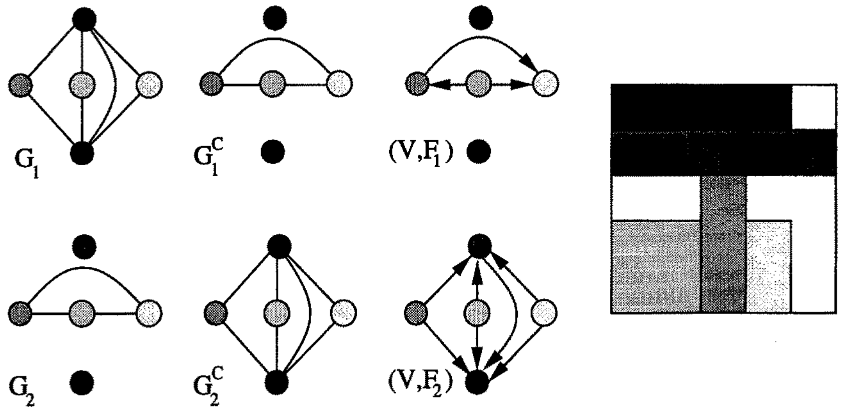

Fig. 2. Constructing a packing pattern for a two-dimensional packing class E

The complements of the component graphs E_1 and E_2 from the example in Figure 2 each have six transitive orientations. Hence there are 36 transitive orientations of E. Figure 3 shows the corresponding packings, constructed by virtue of (1).

If a set of boxes S has a total sum of x_i-widths exceeding k times the x_i-width of the container, then any feasible packing must have an x_i-orthogonal cut that intersects at least $k+1$ boxes. Using the terminology of packing classes, we get an algorithmically useful formulation of this condition:

Theorem 3. *Let E be a packing class for (V, w), $i \in \{1, \ldots, d\}$, $G_i := (V, E_i)$ and $S \subseteq V$. Then $G_i[S]$ contains a clique of size* $\left\lceil \frac{\sum_{s \in S} w_i(s)}{w_i(C)} \right\rceil$.

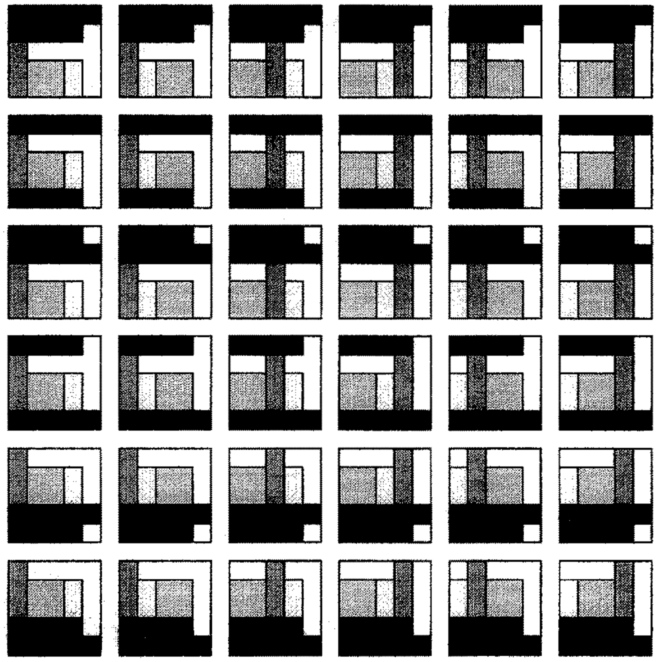

Fig. 3. Packing patterns of the packing class E

3 Conservative scales

In this section, we describe an additional way to reduce the number of cases that we have to consider when looking for an optimal subset.

When defining the packing classes, the size of the boxes only matters when defining the i-feasible sets in P2. Therefore, an OPP-d instance $P = (V, w)$ is characterized by the families

$$\mathcal{Z}(V, w_i) := \{S \subseteq V | w_i(S) \leq w_i(C)\}, \quad i \in \{1, \ldots, d\}.$$

In particular, P2 remains valid when enlarging these families. Let $P = (V, w)$ and $Q = (V, w')$ be instances of OPP-d. If

$$\mathcal{Z}(V, w_i) \subseteq \mathcal{Z}(V, w_i'),\ i \in \{1, \ldots, d\} \tag{2}$$

holds, then any packing class for P is also a packing class for Q, thus any packing for P is a packing for Q. We call such a w' a *conservative scale* for P.

The following theorem provides an easy construction method for conservative scales.

Theorem 4. *Consider an OPP-d instance given by (V, w) and $k \in \mathbb{R}_0^{+d}$. Define w' by*

$$w'_i(v) := \frac{w_i(C)\left(\left\lceil \frac{(k_i+1)w_i(v)}{w_i(C)} \right\rceil - 1\right)}{k_i}.$$

for all $v \in V$, $i \in \{1, \ldots, d\}$ Then (V, w') satisfies (2).

In [8], we use a modfied version of this rounding technique for the fast generation of a class of lower bounds for the one-dimensional bin packing problem. See also [15].

Conservative scales can be used in several ways:

- A simple rejection heuristic for the OPP-d tries to find a conservative scale w' for (V, w), so that the total (d-dimensional) volume of V regarding w' exceeds the volume of the unit container. In this case it is shown that no packing for (V, w') and therefore for the original problem (V, w) can exist.
- A relaxation of the OKP-d ist the corresponding one–dimensional knapsack problem where the box volumes serve as costs. This is used in Beasley's reduction test *Area Program*. The box voluminas regarding conservative scales can either be used as additional costs for this program, turning it into a multidimensional knapsack problem, or as costs for additional one–dimensional relaxations.
- Theorem 3 can be strengthened:

Theorem 5. *Given the assumptions of Theorem 3 and a conservative scaling of (V, w). Then $G_i[S]$ contains a clique of size* $\left\lceil \frac{\sum_{s \in S} w'_i(s)}{w_i(C)} \right\rceil$.

4 Solution of OPP-d

In our search for an optimal subset of boxes, we may have to decide whether a specific subset has a feasible packing. Our solution strategy for this Orthogonal Packing Problem consists of three steps. Their ordering is due to increasing computational effort.

1. Try to disprove the existence of a packing with the heuristic rejection method described above. Stop in case of success.
2. Try to find a packing using an efficient packing heuristic. Stop in case of success.
3. Try to build up a packing class by the following tree search algorithm. In case of failure we can state that no packing for the given OPP–instance exists.

We give only the ideas of the tree search method. A fully detailed description can be found in [16].

The search tree is traversed by DFS. The algorithm branches by fixing $(b, c) \in E_i$ or $(b, c) \notin E_i$. After each assignment, it is checked if new edges forced by one of the defining properties of a packing class or by Theorem 5 must be added to some E_i or if this condition yields a contradiction. The use of P3 is clear. Properties P1 and P2 are hereditary, we have to avoid three types of forbidden induced subgraphs:

1. C_4's,
2. a generalization of odd antiholes; see [16] for more details,
3. infeasible stable sets.

Each time we detect such a fixed subgraph, we can abandon the search on this node. Each time we detect such a fixed subgraph, except for a set of "equivalent edges", we can fix one of them.

These tests as well as the test if the fixed edges already form a packing class are based on comparability graph recognition and on the calculation of maximal weighted cliques in comparability graphs, so they can be performed efficiently [11].

5 A tree search algorithm for OKP-d

The subsets of V that are candidates for an optimal solution are enumerated in a search tree, where each node corresponds to the OKP-d restricted by upper and lower bounds on the number of boxes used from each *box type*.

Box types are classes of the partition $V = \sum_{j=1}^{m} V_j$ that arise as the union of boxes with equal size and equal value. For $j \in \{1, \ldots, m\}$ let $n_j = |V_j|$ and order $V_j = \{b_{j,1}, \ldots, b_{j,n_j}\}$. $\overline{n}_j$ and $\underline{n}_j$ denote the upper and lower bounds on box type V_j.

The initial tree node starts with the original OKP-d. (For $j \in \{1, \ldots, m\}$: $\overline{n}_j = n_j$, $\underline{n}_j = 0$.)
Consider a node N given by $\overline{n} = (\overline{n}_j)_{j=1,\ldots,m}$ and $\underline{n} = (\underline{n}_j)_{j=1,\ldots,m}$.
If $\overline{n} = \underline{n}$ holds, N is a leaf. The corresponding restricted OKP-d is equivalent to the OPP-d $(\underline{V}, w)$ with

$$\underline{V} := \sum_{\substack{j=1 \\ \underline{n}_j > 0}}^{m} \{b_{j,1}, \ldots, b_{j,\underline{n}_j}\}.$$

We apply the solution strategy from Section 4.
Otherwise we choose j with $\overline{n}_j > \underline{n}_j$, and branch to the tree nodes given by $(\underline{n}_1, \ldots, \underline{n}_{j-1}, \nu, \underline{n}_{j+1}, \ldots, \underline{n}_m)$ and $(\overline{n}_1, \ldots, \overline{n}_{j-1}, \nu, \overline{n}_{j+1}, \ldots, \overline{n}_m)$
for all $\nu \in \{\underline{n}_j, \ldots, \overline{n}_j\}$.

The tree is traversed by best–first search. In order to keep it small, we implemented the following pruning and reduction techniques:

- The reduction tests *Overlapping Pieces*, *Free Area*, *Free Value* and *Area Program* from [2] are applied to reduce the gap between $\overline{n}$ and $\underline{n}$. *Area Program* is used with eight different conservative scales, constructed according to Theorem 4. In addition, *Area Program* supplies an upper bound for the OKP-d.
- Define $\overline{V}$ on the analogue of $\underline{V}$. If $\overline{V}$ can be packed into C using a simple heuristic, we have an optimal packing for this subtree.
- The lower bound $\underline{n}$ demands that $\underline{V}$ is covered by each optimal solution of the restricted OKP-d. So if the OPP-d $(\underline{V}, w)$ can be shown to have no packing, the search on this subtree can be abandoned. Again we apply the method from Section 4.

If V contains "large" boxes, the following theorem provides an additional reduction method.

Theorem 6. *Let $\iota \in \{1, \ldots, d\}$ be fixed. Consider a disjoint partition $V = V_1 \cup V_2 \cup V_3$ with $w_i(v_1) + w_i(v_2) > w_i(C)$ for $v_1 \in V_1$, $v_2 \in V_1 \cup V_2$ and $\iota \neq i$. Let C' be a container with size $w_i(C)$ in all directions except of ι, where it has size $\sum_{v_1 \in V_1} w_\iota(v_1)$.*

If V fits into C and $V_1 \cup V_3$ fits into C', then there exists a packing for (V, w) where the boxes in $V_1 \cup V_3$ and the boxes in V_2 are separated by the hyperplane $x_i = \sum_{v_1 \in V_1} w_i(v_1)$.

If the assumption holds, we can restrict our attention to the solution of the OKP-d given by (V_2, w) with the container C'' of size $w(C)$ in all directions except for x_ι, where it has size $w_\iota(C) - \sum_{v_1 \in V_1} w_\iota(v_1)$, then merge the optimal packing with the packing of $(V_1 \cup V_3, C')$.

6 Computational Results

The algorithm was implemented in C++ and run on a SPARCserver 1000. The code was compiled using the gcc compiler. Besides the running times we list the number of nodes in the OKP search tree, the number of calls of the OPP search tree, and the total number of OPP nodes.

We solved the test problems beasley 1–12 from the OR–Library [2, 3]. The problem instances chrhad 3,8,11,12 come from [12]. These are all published test instances for the general OKP. All optimal solutions were found in less than 0.3 s. For solutions to chrhad8 and chrhad11 we could give the first proof of optimality. For comparison, we show the running times reported in [2, 12].

The instances wang 20 and chrwhi 62 come from [17] and [5]. They were originally meant as test instances for the guillotine-constrained problem.

The new instances okp 1-5 are random instances generated in the same way as beasley 1-12 (see [2]) after applying initial reduction.

Table 1. Computational results. The columns "B85" and "HC95" show the running times as reported in [2,12].

problem	container size	box types	# boxes	OKP nodes	OPP calls	OPP nodes	time/s	B85	HC95	opt. sol.
beasley1	(10, 10)	5	10	8	0	0	0.03	0.9		164
beasley2	(10, 10)	7	17	5	0	0	0.04	4.0		230
beasley3	(10, 10)	10	21	21	3	35	0.09	10.5		247
beasley4	(15, 10)	5	7	1	0	0	0.01	0.1	0.04	268
beasley5	(15, 10)	7	14	1	0	0	0.01	0.4		358
beasley6	(15, 10)	10	15	34	4	5	0.13	55.2	45.2	289
beasley7	(20, 20)	5	8	0	0	0	0.01	0.5	0.04	430
beasley8	(20, 20)	7	13	18	3	149	0.09	218.6		834
beasley9	(20, 20)	10	18	3	0	0	0.03	18.3	5.2	924
beasley10	(30, 30)	5	13	1	0	0	0.02	0.9		1452
beasley11	(30, 30)	7	15	45	10	10	0.13	79.1		1688
beasley12	(30, 30)	10	22	46	6	63	0.26	229.0	>800	1865
chrhad3	(30, 30)	7	7	1	0	0	0.01		532	1178
chrhad8	(40, 40)	10	10	3	0	0	0.02		>800	2517
chrhad11	(30, 30)	15	15	45	1	47	0.20		>800	1270
chrhad12	(40, 40)	15	15	2	0	0	0.02		65.2	2949
wang20	(70, 40)	20	42	630	69	9235	12.51			2726
chrwhi62	(40, 70)	20	62	398	87	17431	7.47			1860
okp1	(100,100)	15	50	986	79	4937	11.60			27718
okp2	(100,100)	30	30	9355	2677	3413	116.24			22502
okp3	(100,100)	30	30	5799	140	148	73.03			24019
okp4	(100,100)	33	61	1610	0	0	50.09			32893
okp5	(100,100)	29	97	1557	159	12038	40.14			27923

Table 2. The new problem instances okp1–okp5.

Problem okp1: container = (100,100), 15 box types (50 boxes)	
size =	[(4,90),(22,21),(22,80),(1,88),(6,40),(100,9),(46,14),(10,96), (70,27),(57,18),(10,84),(100,1),(2,41),(36,63),(51,24)]
value =	[838,521,4735,181,706,2538,1349,1685,5336,1775,1131,129,179, 6668,3551]
n =	[5,2,3,5,5,5,3,1,3,1,1,5,5,2,4]
Problem okp2: container = (100,100), 30 box types (30 boxes)	
size =	[(8,81),(5,76),(42,19),(6,80),(41,48),(6,86),(58,20),(99,3),(9,52), (100,14),(7,53),(24,54),(23,77),(42,32),(17,30),(11,90),(26,65), (11,84),(100,11),(29,81),(10,64),(25,48),(17,93),(77,31),(3,71), (89,9),(1,6),(12,99),(33,72),(21,26)]
value=	[953,389,1668,676,3580,1416,3166,537,1176,3434,676,1408,2362, 4031,1152,2255,3570,1913,1552,4559,713,1279,3989,4850,299, 1577,12,2116,2932,1214]
$n_j =$	$1, \quad j \in \{1, \ldots, 30\}$
Problem okp3: container = (100,100), 30 box types (30 boxes)	
size =	[(3,98),(34,36),(100,6),(49,26),(14,56),(100,3),(10,90),(23,95), (10,97),(50,47),(41,45),(13,12),(19,68),(50,46),(23,70),(28,82), (12,65),(9,86),(21,96),(19,64),(21,75),(45,26),(19,77),(5,84), (16,21),(23,69),(5,89),(22,63),(41,6),(76,30)]
value=	[756,2712,1633,2332,2187,470,1569,4947,2757,4274,4347,396,3866, 5447,2904,6032,1799,929,5186,2120,1629,2059,2583,953,1000, 2900,1102,2234,458,5458]
$n_j =$	$1, \quad j \in \{1, \ldots, 30\}$
Problem okp4: container = (100,100), 33 box types (61 boxes)	
size =	[(48,48),(6,85),(100,14),(17,85),(69,20),(12,72),(5,48),(1,97), (66,36),(15,53),(29,80),(19,77),(97,7),(7,57),(63,37),(71,14),(3,76), (34,54),(5,91),(14,87),(62,28),(6,7),(20,71),(92,7),(10,77),(99,4), (14,44),(100,2),(56,40),(86,14),(22,93),(13,99),(7,76)]
value =	[5145,874,2924,3182,2862,1224,531,249,6601,1005,6228,3362,907, 473,6137,1556,313,4123,581,1999,5004,2040,3143,795,1460,841, 1107,280,5898,2096,4411,3456,1406]
n =	[1,2,1,1,1,1,3,3,2,1,3,1,1,2,2,1,3,1,2,1,3,3,1,1,2,3,2,3,2,1,1,3,3]
Problem okp5: container = (100,100), 29 box types (97 boxes)	
size =	[(8,81),(5,76),(42,19),(6,80),(41,48),(6,86),(58,20),(99,3),(9,52), (100,14),(7,53),(24,54),(23,77),(42,32),(17,30),(11,90),(26,65), (11,84),(100,11),(29,81),(10,64),(25,48),(17,93),(77,31),(3,71), (89,9),(1,6),(12,99),(21,26)]
value =	[953,389,1668,676,3580,1416,3166,537,1176,3434,676,1408,2362, 4031,1152,2255,3570,1913,1552,4559,713,1279,3989,4850,299, 1577,12,2116,1214]
n =	[3,4,4,4,1,5,5,5,5,4,5,1,1,5,5,4,2,3,1,1,2,1,4,1,5,4,5,2,5]

References

1. M. Arenales and R. Morabito. An AND/OR–graph approach to the solution of two–dimensional non–guillotine cutting problems. *European Journal of Operations Research*, **84**, 1995, pp. 599–617.
2. J. E. Beasley. An exact two–dimensional non–guillotine cutting stock tree search procedure. *Operations Research*, **33**, 1985, pp. 49–64.
3. J. E. Beasley. OR-Library: distributing test problems by electronic mail. *Journal of the Operations Research Society*, **41**, 1990, pp. 1069–1072.
4. M. Biró and E. Boros. Network flows and non–guillotine cutting patterns. *European Journal of Operations Research*, **16**, 1984, pp. 215–221.
5. N. Christofides and C. Whitlock. An algorithm for two–dimensional cutting problems. *Operations Research*, **25**, 1977, pp. 31–44.
6. K. A. Dowsland. An exact algorithm for the pallet loading problem. *European Journal of Operations Research*, **31**, 1987, pp. 78-84.
7. S. P. Fekete and H. Meijer. Rectangle and box visibility graphs in 3D. To appear in *International Journal of Computational Geometry and its Applications.* Available at `ftp://ftp.zpr.uni-koeln.de/pub/paper/zpr96-224.ps.gz`.
8. S. P. Fekete and J. Schepers. A new classs of lower bounds for bin packing problems. ZPR Report 97-265. Available at `ftp://ftp.zpr.uni-koeln.de/pub/paper/zpr97-265.ps.gz`.
9. S. P. Fekete, J. Schepers, and M. Wottawa. PACKLIB: a library of packing problems. Under construction.
10. G. Galambos and G. J. Woeginger. On-line bin packing – a restricted survey. *Zeitschrift für Operations Research*, **42**, 1995, pp. 25–45.
11. M. C. Golumbic. *Algorithmic graph theory and perfect graphs.* Academic Press, New York, 1980.
12. E. Hadjiconstantinou and N. Christofides. An exact algorithm for general, orthogonal, two–dimensional knapsack problems. *European Journal of Operations Research*, **83**, 1995, pp. 39–56.
13. M. R. Jerrum. A data structure for systems of orthogonal, non-overlapping rectangles. Internal Report CSR-239-87, Department of Computer Science, Edinburgh University, August 1987.
14. N. Korte and R. H. Möhring. An incremental linear–time algorithm for recognizing interval graphs. *Siam Journal of Computing*, **18**, 1989, pp. 68-81.
15. S. Martello and P. Toth. *Knapsack Problems – Algorithms and Computer Implementations,* Wiley, Chichester, 1990.
16. J. Schepers. *Exakte Algorithmen für orthogonale Packungsprobleme.* Doctoral thesis, Universität zu Köln, in preparation for 1997.
17. P. Y. Wang. Two algorithms for constrained two–dimensional cutting stock problems. *Operations Research*, **31**, 1983, pp. 573-586.