Higher-Dimensional Packing with Order Constraints

Sándor P. Fekete[1], Ekkehard Köhler[2], and Jürgen Teich[3]

[1] Department of Mathematical Optimization TU Braunschweig, Braunschweig, Germany, sandor.fekete@tu-bs.de
[2] Department of Mathematics, TU Berlin, Berlin, Germany, ekoehler@math.tu-berlin.de
[3] Computer Engineering Laboratory, University of Paderborn, Paderborn, Germany, teich@date.upb.de

Abstract. We present a first exact study on higher-dimensional packing problems with order constraints. Problems of this type occur naturally in applications such as logistics or computer architecture and can be interpreted as higher-dimensional generalizations of scheduling problems. Using graph-theoretic structures to describe feasible solutions, we develop a novel exact branch-and-bound algorithm. This extends previous work by Fekete and Schepers; a key tool is a new order-theoretic characterization of feasible extensions of a partial order to a given comparability graph that is tailor-made for use in a branch-and-bound environment. The usefulness of our approach is validated by computational results.

1 Introduction

Scheduling and Packing Problems. Scheduling is arguably one of the most important topics in combinatorial optimization. Typically, we are dealing with a one-dimensional set of objects ("jobs") that need to be assigned to a finite set of containers ("machines"). Problems of this type can also be interpreted as (one-dimensional) packing problems, and they are NP-hard in the strong sense, as problems like 3-Partition are special cases.

Starting from this basic scenario, there are different generalizations that have been studied. Many scheduling problems have *precedence constraints* on the sequence of jobs. On the other hand, a great deal of practical packing problems consider *higher-dimensional* instances, where objects are axis-aligned boxes instead of intervals. More-dimensional packing problems arise in many industries, where steel, glass, wood, or textile materials are cut. The three-dimensional problem is important for practical applications such as container loading.

In this paper, we give the first study of problems that comprise both generalizations: these are higher-dimensional packing problems with order constraints—or, from a slightly different point of view, higher-dimensional scheduling problems. In higher-dimensional packing, these problems arise when dealing with precedence constraints that are present in many container-loading problems. Another practical motivation to consider multi-dimensional scheduling problems

F. Dehne, J.-R. Sack, and R. Tamassia (Eds.): WADS 2001, LNCS 2125, pp. 300–312, 2001.

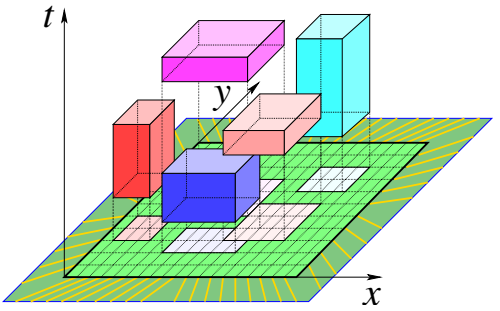

Fig. 1. *An FPGA and a set of five jobs, shown (as rectangles) in regular two-dimensional x,y-space and (as boxes) in three-dimensional space-time x, y, t. All jobs must be placed inside the chip and must not overlap if executed simultaneously on the chip.*

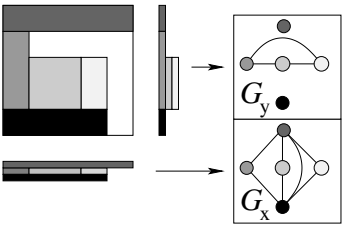

Fig. 2. *Projecting the boxes of a feasible packing onto the coordinate axes defines interval graphs (here in 2D: $\boldsymbol{G_x}$ and $\boldsymbol{G_y}$).*

arises from optimizing the reconfiguration of a particular type of computer chips called FPGA's—described below.

Field-Programmable Gate Arrays and More-Dimensional Scheduling. A particularly interesting class of instances of three-dimensional orthogonal packing arises from a new type of reconfigurable computer chips, called *field-programmable gate arrays* (FPGA's).

An FPGA typically consists of a regular rectangular grid of equal configurable cells (logic blocks) that allow the prototyping of simple logic functions together with simple registers and with special routing resources (see Figure 1). These chips (see e.g. [1,30]) may support several independent or interdependent jobs and designs at a time, and parts of the chip can be reconfigured quickly during run-time. (For more technical details on the underlying architecture, see our previous paper [28], and the more recent abstract [4].) Thus, we are faced with a general class of problems that can be seen as both scheduling and packing problems; two dimensions correspond to the space of the chip area, while the third dimension corresponds to time. In this paper, we develop a set of mathematical tools to deal with these *higher-dimensional scheduling problems*. We show that our methods are suitable for solving instances of interesting size to optimality.

Related Work. It is easy to see that any higher-dimensional packing problem (possibly with precedence constraints on the temporal order) can be relaxed to a resource-constrained scheduling problem. However, there are examples with as few as eight jobs showing that the converse is not true, even for small instances of two-dimensional packing problems without any precedence constraints: An optimal solution for the corresponding resource-constrained scheduling problem may not give rise to a feasible arrangement of rectangles for the original packing problem.

Higher-dimensional packing problems (without order constraints) have been considered by a great number of authors, but only few of them have dealt with the exact solution of general two-dimensional problems. See [6,8] for an overview. It should be stressed that unlike one-dimensional packing problems, higher-dimensional packing problems allow no straightforward formulation as integer programs: After placing one box in a container, the remaining feasible space will in general not be convex. Moreover, checking whether a given set of boxes fits into a particular container (the so-called *orthogonal packing problem*, OPP) is trivial in one-dimensional space, but NP-hard in higher dimensions.

Nevertheless, attempts have been made to use standard approaches of mathematical programming. Beasley [2] and Hadjiconstantinou and Christofides [15] have used a discretization of the available positions to an underlying grid to get a 0-1 program with a pseudopolynomial number of variables and constraints. Not surprisingly, this approach becomes impractical beyond instances of rather moderate size. More recently, Padberg [25] gave a *mixed integer programming* formulation for three-dimensional packing problems, similar to the one anticipated by Schepers [26] in his thesis. Padberg expressed the hope that using a number of techniques from branch-and-cut will be useful; however, he did not provide any practical results to support this hope.

In [6,8,9,10,28], a different approach to characterizing feasible packings and constructing optimal solutions is described. See Figure 2 for a visual description. A graph-theoretic characterization of the relative position of the boxes in a feasible packing (by so-called *packing classes*) is used, which represent d-dimensional packings by a tuple of d interval graphs (called *component graphs*). Any d-tuple of graphs G_i arising in this manner must satisfy the following conditions:

C1: G_i is an interval graph, $\forall i \in \{1, \cdots, d\}$.
C2: Any independent set S of G_i is i-admissible, $\forall i \in \{1, \cdots, d\}$, i.e., $w_i(S) = \sum_{v \in S} w_i(v) \leq h_i$, as all boxes in S must fit into the container in the ith dimension.
C3: $\cap_{i=1}^{d} E_i = \emptyset$. In other words, there must be at least one dimension in which the corresponding boxes do not overlap.

A d-tuple of component graphs satisfying these necessary conditions is called a *packing class*. The remarkable property (proven in [8,26]) is that these three conditions are also sufficient for the existence of a feasible packing.

Theorem 1
A d-tuple of graphs $G_i = (V, E_i)$ corresponds to a feasible packing, iff it is a packing class, i. e., if it satisfies the conditions **C1, C2, C3**.

This factors out a great deal of symmetries between different feasible packings, it allows to make use of a number of elegant graph-theoretic tools (like the characterizations reported in [12,13]), and it reduces the geometric problem to a purely combinatorial one without using brute-force methods like introducing an underlying coordinate grid. Combined with good heuristics for dismissing infeasible sets of boxes [7], a tree search for constructing feasible packings was

developed. This exact algorithm has been implemented; it outperforms previous methods by a clear margin. See our previous papers for details.

Graph Theory of Order Constraints. In the context of scheduling with precedence constraints, a natural problem is the following, called *transitive ordering with precedence constraints* (TOP): Consider a partial order $P = (V, \prec)$ of precedence constraints and a (temporal) comparability graph $G = (V, E)$, such that all relations in P are represented by edges in G. Is there a transitive orientation $D = (V, A)$ of G, such that P is contained in D?

Korte and Möhring [18] have given a linear-time algorithm for deciding TOP. However, their approach is only useful when the full set of edges in G is known. When running a branch-and-bound algorithm for solving a scheduling problem, these edges of G are only known partially, but they may already prohibit the existence of a feasible solution for a given partial order P. This makes it desirable to come up with structural characterizations that are already useful when only parts of G are known.

Results of this paper. In this paper, we give the first exact study of higher-dimensional packing with order constraints, which can also be interpreted as *higher-dimensional scheduling problems*. We develop a general framework for problems of this type by giving a pair of necessary and sufficient conditions for the existence of a solution for the problem TOP on graphs G in terms of forbidden substructures. Using the concept of packing classes described above, our conditions can be used quite effectively in the context of a branch-and-bound framework, since it can recognize infeasible subtrees at "high" branches of the search tree. In particular, we describe how to find an exact solution to the problem of minimizing the height of a container of given base area. If this third dimension represents time, this amounts to minimizing the makespan of a higher-dimensional scheduling problem. We validate the usefulness of these concepts and results by providing computational results. Other problem versions (like higher-dimensional knapsack or bin packing problems with order constraints) can be treated similarly.

The rest of this paper is organized as follows. In Section 2, we describe basic assumptions and some terminology. In Section 3, we introduce precedence constraints, describe the mathematical foundations for incorporating them into the search, and explain how to implement the resulting algorithms. Finally, we present computational results for a number of different benchmarks in Section 4.

2 Preliminaries

Problem instances. We assume that a problem instance is given by a *set of jobs* V. All our results can be applied to instances in arbitrary fixed dimension. For the purposes of this abstract, the reader may wish to focus on the scenario where each job has a *spatial requirement* in the x- and y-direction, denoted by $w_x(v)$ and $w_y(v)$, and possibly a *duration*, denoted by a size $w_t(v)$ along the

time axis. The available space H consists of an area of size $h_x \times h_y$. In addition, there may be an overall allowable time h_t for all jobs to be completed.

Graphs. Some of our descriptions make use of a number of different graph classes. An (undirected) graph $G = (V, E)$ is given by a set of vertices V, and a set of edges E; each edge describes the adjacency of a pair of vertices, and we write $\{u, w\}$ for an edge between vertices u and w. For a graph G, we obtain the *complement graph* $\overline{G}$ by exchanging the set E of edges with the set $\overline{E}$ of non-edges. In a directed graph $D = (V, A)$, edges are oriented, and we write (u, w) to denote an edge directed from u to w. A graph $G = (V, E)$ is a *comparability graph* if the edges E can be oriented to a set of directed arcs A, such that we get a (transitively closed) partial order, i.e., a cycle-free digraph for which the existence of edges $(u, v) \in A$ and $(v, w) \in A$ for any $u, v, w \in V$ implies the existence of $(u, w) \in A$.

Precedence constraints. Mathematically, a set of precedence constraints is given by a partial order $P = (V, \prec)$ on V. The relations in $\prec$ form a directed acyclic graph $D_P = (V, A_P)$, where A_P is the set of directed arcs. In the presence of such a partial order, a feasible schedule is assumed to satisfy the capacity constraints of the container, as well as these additional constraints.

Packing problems. In the following, we treat jobs as axis-aligned multi-dimensional boxes with given orientation, and feasible schedules as arrangements of boxes that satisfy all side constraints. This is implied by the term of a *feasible packing*. There may be different types of objective functions, corresponding to different types of packing problems. The *Orthogonal Packing Problem* (OPP) is to decide whether a given set of boxes can be placed within a given "container" of size $h_x \times h_y \times h_t$. For the Constrained OPP (COPP), we also have to satisfy a partial order $P = (V, \prec)$ of precedence constraints in the t-dimension. (To emphasize the motivation of temporal precedence constraints, we write t to suggest that the time coordinate is constrained, and x and y to imply that the space coordinates are unrestricted. Clearly, our approach works the same way when dealing with spatial restrictions.)

There are various optimization problems that have the OPP or COPP as their underlying decision problem. Since our main motivation arises from dynamic chip reconfigurations, where we want to minimize the overall running time, we focus on the *Constrained Strip Packing Problem* (CSPP), which is to minimize the size h_t for a given base size $h_x \times h_y$, such that all boxes fit into the container $h_x \times h_y \times h_t$. Clearly, we can use a similar approach for other objective functions.

3 Packing Problems with Precedence Constraints

As mentioned in the introduction, a key advantage of considering packing classes is that it allows to deal with packing problems independent of precise geometric placement, and that it allows arbitrary feasible interchanges of placement. However, for most practical instances, we have to satisfy additional constraints for the temporal placement, i.e., for the start times of jobs. For our approach, the nature of the data structures may simplify these problems from three-dimensional

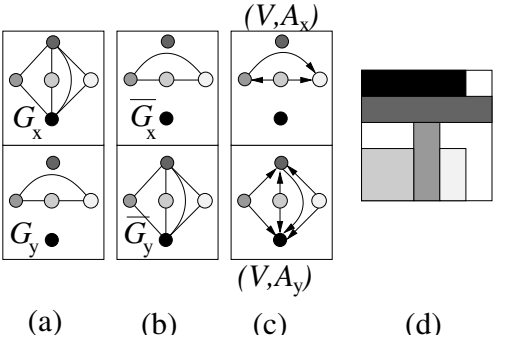

Fig. 3. *(a) A two-dimensional packing class, with G_x representing x-, G_y representing y-projections. (b) The corresponding comparability graphs. (c) A transitive orientation. (d) A feasible packing corresponding to the orientation.*

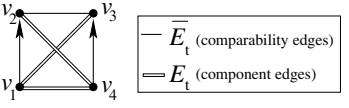

Fig. 4. *An example of a comparability graph $\overline{G_t} = (V, \overline{E_t})$ with a partial order P contained in $\overline{E_t}$, such that no transitive orientation of $\overline{G_t}$ extends P.*

to purely two-dimensional ones: If the whole schedule is given, all edges E_t in one of the graphs are determined, so we only need to construct the edge sets E_x and E_y of the other graphs. As worked out in detail in [27,28], this allows it to solve the resulting problems quite efficiently if the arrangement in time is already given.

A more realistic, but also more involved situation arises if only a set of precedence constraints is given, but not the full schedule. We describe in the following how further mathematical tools in addition to packing classes allow useful algorithms.

3.1 Packing Classes and Interval Orders

An important concept used frequently in the following is the concept of a *component graph*. Any edge $\{v_1, v_2\}$ in a component graph G_i corresponds to an overlap between the projections of boxes 1 and 2 onto the x_i-axis. This means that the complement graph $\overline{G_i}$ given by the complement $\overline{E_i}$ of the edge set E_i consists of all pairs of coordinate intervals that are "comparable": Either the first interval is "to the left" of the second, or vice versa. Any (undirected) graph of this type is a so-called *comparability graph* (see [14] for further details). By orienting edges to point from "left" to "right" intervals, we get a partial order of the set V of vertices, a so-called *interval order* [22]. Obviously, this order relation is transitive, i.e., $e \prec f$ and $f \prec g$ imply $e \prec g$, which is the reason why we also speak of a *transitive orientation* of the undirected comparability graph G_i. See Figure 3 for a (two-dimensional) example of a packing class, the corresponding comparability graph, a transitive orientation, and the packing corresponding to the transitive orientation.

Now consider a situation where we need to satisfy a partial order $P = (V, A_P)$ of precedence constraints in the time dimension. It follows that each arc $a = (u, w) \in A_P$ in this partial order forces the corresponding undirected edge $e = \{u, w\}$ to be excluded from E_t. Thus, we can simply initialize our algorithm for constructing packing classes by fixing all undirected edges corresponding to A_P to be contained in $\overline{E_t}$. After running the original algorithm, we may get additional comparability edges. As the example in Figure 4 shows, this causes an additional problem: Even if we know that the graph $\overline{G_t}$ has a transitive orientation, and all arcs $a = (u, w)$ of the precedence order (V, A_P) are contained in $\overline{E_t}$ as $e = \{u, w\}$, it is not clear that there is a transitive orientation that contains all arcs of A_P.

3.2 Finding Feasible Transitive Orientations

Consider a comparability graph $\overline{G}$ that is the complement of an interval graph G. The problem TOP of deciding whether $\overline{G}$ has a transitive orientation that extends a given partial order P has been studied in the context of scheduling. where $\overline{G}$ is the comparability graph of an interval order. For this scenario, Korte and Möhring [18] give a linear-time algorithm for determining a solution for TOP, or deciding that none exists. Their approach is based on a very special data structure called *modified PQ-trees*.

In principle, it is possible to solve higher-dimensional packing problems with precedence constraints by adding this algorithm as a black box to test the leaves of our search tree for packing classes: In case of failure, backtrack in the tree. However, the resulting method cannot be expected to be reasonably efficient: During the course of our tree search, we are not dealing with one fixed comparability graph, but only build it while exploring the search tree. This means that we have to expect spending a considerable amount of time testing similar leaves in the search tree, i.e., comparability graphs that share most of their graph structure. It may be that already a very small part of this structure that is fixed very "high" in the search tree constitutes an obstruction that prevents a feasible orientation of all graphs constructed below it. So a "deep" search may take a long time to get rid of this obstruction. This makes it desirable to use more structural properties of comparability graphs and their orientations to make use of obstructions already "high" in the search tree.

3.3 Implied Orientations

As in the basic packing class approach, we consider the component graphs G_i and their complements, the comparability graphs $\overline{G_i}$. This means that we continue to have three basic states for any edge: (1) edges that have been fixed to be in E_i, i.e., *component edges*; (2) edges that have been fixed to be in $\overline{E_i}$, i.e., *comparability edges*; (3) *unassigned edges*.

In order to deal with precedence constraints, we also consider orientations of the comparability edges. This means that during the course of our tree search, we can have three different possible states for each comparability edge: (2a)

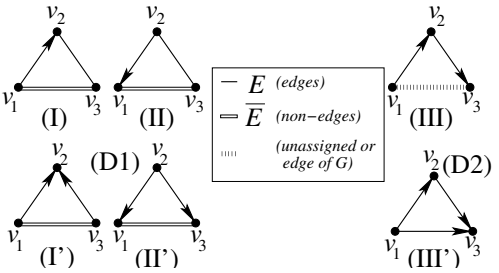

Fig. 5. *Two types of implications for edges and their orientations: Above are the arrangements that trigger path implications (D1, left) and transitivity implications (D2, right); below are the forced orientations of edges.*

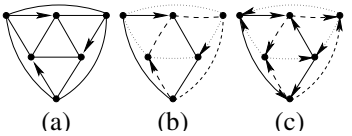

Fig. 6. *(a) A graph $\overline{G_t}$ with a partial order formed by three directed edges; (b) there are three path implication classes that each have one directed arc; (c) carrying out path implications creates directed cycles, i.e., transitivity conflicts.*

one possible orientation; (2b) the opposite possible orientation; (2c) no assigned orientation.

A stepping stone for this approach arises from considering the following two configurations – see Figure 5:

The first consists of the two comparability edges $\{v_1, v_2\}, \{v_2, v_3\} \in \overline{E_t}$, such that the third edge $\{v_1, v_3\}$ has been fixed to be an edge from the component graph E_t. Now any orientation of one of the comparability edges forces the orientation of the other comparability edge, as shown in the left part of the figure. We call this arrangement a *path implication*, since this configuration corresponds to an induced path in $\overline{G_i}$,

The second configuration consists of two directed comparability edges, say, the edges (v_1, v_2) and (v_2, v_3). In this case we know that the edge $\{v_1, v_3\}$ must also be a comparability edge, with an orientation of (v_1, v_3). Since this configuration arises directly from transitivity in $\overline{G_i}$, we call this arrangement a *transitivity implication.*

Clearly, any implication arising from one of the above configurations can induce further implications.

In particular, when considering only sequences of path implications, we get a partition of comparability edges into *path implication classes.* Two comparability edges are in the same implication class, iff there is a sequence of path implications, such that orienting one edge forces the orientation of the other edge. For an example, consider the arrangement in Figure 4. Here, all three comparability edges $\{v_1, v_2\}$, $\{v_2, v_3\}$, and $\{v_3, v_4\}$ are in the same path implication class. Now the orientation of (v_1, v_2) implies the orientation (v_3, v_2), which in turn implies the orientation (v_3, v_4), contradicting the orientation of $\{v_3, v_4\}$ in the given par-

tial order P. It is not hard to see that the implication classes form a partition of the comparability edges, since we are dealing with an equivalence relation.

We call a violation of a path implication a *path conflict*.

As the example in Figure 6 shows, only excluding path conflicts when recursively carrying out path implications does not suffice to guarantee the existence of a feasible orientation: Working through the queue of path implications, we end up with a directed cycle, which violates a transitivity implication.

We call a violation of a transitivity implication a *transitivity conflict*.

Summarizing, we have the following necessary conditions for the existence of a transitive orientation that extends a given partial order P:

D1: Any path implication can be carried out without a conflict.
D2: Any transitivity implication can be carried out without a conflict.

These necessary conditions are also sufficient:

Theorem 2 (Fekete, Köhler, Teich)
Let $P = (V,A_P)$ be a partial order with arc set A_P that is contained in the edge set E of a given comparability graph $G = (V, E)$. A_P can be extended to a transitive orientation of G, iff all arising path implications and transitivity can be carried out without creating a path conflict or a transitivity conflict.

A proof and further mathematical details are described in our report [5]. The interested reader may take note that we are extending previous work by Gallai [11], who extensively studied implication classes of comparability graphs. In particular, we use the concept of *modular decomposition*. For more background on implication classes and comparability graphs, see Kelly [17], Möhring [22] for informative surveys on this topic, and Krämer [20] for an application in scheduling theory.

3.4 Solving Problems with Precedence Constraints

We start by fixing for all arcs $(u, v) \in A$ the edge $\{u, v\}$ as an edge in the comparability graph $\overline{G_t}$, and we also fix its orientation to be (u, v). In addition to the tests for enforcing the conditions for unoriented packing classes (C1, C2, C3), we employ the implications suggested by conditions D1 and D2. For this purpose, we check directed edges in $\overline{G_t}$ for being part of a triangle that gives rise to either implication. Any newly oriented edge in $\overline{G_t}$ gets added to a queue of unprocessed edges. Like for packing classes, we can again get cascades of fixed edge orientations. If we get an orientation conflict or a cycle conflict, we can abandon the search on this tree node. The correctness of the overall algorithm follows from Theorem 2; in particular, the theorem guarantees that we can carry out implications in an arbitrary order.

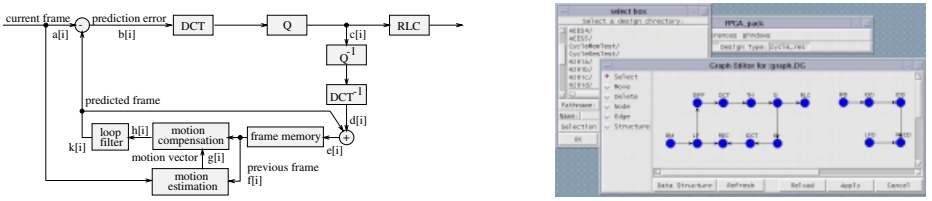

Fig. 7. *Block diagram of a video-codec (H.261) (left); precedence constraints for the video-codec (right).*

4 Computational Experiments

In the following we present our results for different types of instances: The video-codec benchmark described in Section 4.1 arises from an actual application to FPGA's. In Section 4.2 we give a number of results arising from different geometric packing problems.

Our code was implemented in C++ and tests were carried out on a SUN Ultra 2.

4.1 Video-Codec Benchmark

Figure 7 shows a block diagram of the operation of a hybrid image sequence coder/decoder that arises from the FPGA application. The purpose of the coder is to compress video images using the H.261 standard. In this device, transformative and predictive coding techniques are unified. The compression factor can be increased by a predictive method for motion estimates: blocks inside a frame are predicted from blocks of previous images. See our report [4] for more technical details. The result is shown in Table 1.

4.2 Geometric Instances

Here we describe computational results for two types of two-dimensional objects. See Table 2 for an overview. The first class of instances was constructed from a

Table 1. Optimizing reconfigurations for the Video-Codec

test	container sizes			CPU-time (s)
	h_t	h_x	h_y	
1	59	64	64	24.87 s

Table 2. Optimal packing with order constraints

instance	optimal h_t	h_x	upper bound	lower bound
okp17-0	169	100	7.29 s	179 s
okp17-1	172	100	6.73 s	1102 s
okp17-2	182	100	5.39 s	330 s
okp17-3	184	100	236 s	553 s
okp17-4	245	100	0.17 s	0.01 s
square21-no	112	112	84.28 s	0.01 s
square21-mat	117	112	15.12 s	277 s
square21-tri	125	112	107 s	571 s
square21-2mat	[118,120]	[118,120]	346 s	476 s

Table 3. The `okp17` problem instances.

`okp17`:	base width of container = 100, number of boxes = 17
sizes =	[(8,81),(5,76),(42,19),(6,80), (9,52),(41,48),(6,86),(58,20), (99,3),(100,14),(7,53),(24,54), (23,77),(42,32),(17,30),(11,90), (26,65)]
okp17-0:	no order constraints
okp17-1:	11→8, 11→16
okp17-2:	11→8, 11→16, 8→16
okp17-3:	11→8, 11→16, 8→16, 8→17, 11→7, 16→7
okp17-4:	11→8, 11→16, 8→16, 8→17, 11→7, 16→7, 17→16

Table 4. The `square21` problem instances.

`square21`:	base width of container = 112, number of boxes = 21
sizes =	[(50,50),(42,42),(37,37),(35,35), (33,33),(29,29),(27,27),(25,25), (24,24),(19,19),(18,18),(17,17), (16,16),(15,15),(11,11),(9,9), (8,8), (7,7),(6,6),(4,4),(2,2)]
square21-0:	no order constraints
square21-mat:	2→4, 6→7, 8→9, 11→15, 16→17, 18→19, 24→25, 27→29, 33→35, 37→42, 2→50, 50→4
square21-tri:	2→15, 15→17, 2→27, 4→16, 16→29, 4→29, 6→17, 17→33, 6→33, 7→18, 18→35, 7→35, 8→19, 19→37, 8→37, 9→24, 24→42, 9→42, 11→25, 25→50, 11→50
square21-2mat:	x-constraints: 2→19, 6→25, 8→29, 11→35, 16→42, 18→4, 24→7, 27→9, 33→15, 37→17, 50→4, 18→50 y-constraints: 2→4, 6→7, 8→9, 11→15, 16→17, 18→19, 24→25, 27→29, 33→35, 37→42, 2→50, 50→4

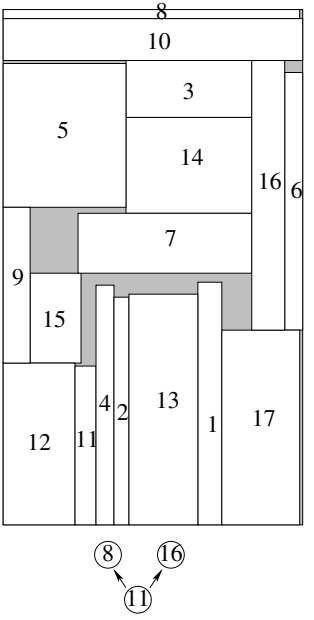

Fig. 8. *An optimal solution for the instance okp17-1.*

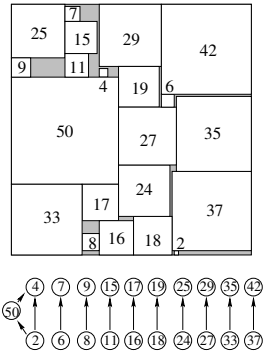

Fig. 9. *An optimal solution for the instance square21-mat.*

particularly difficult random instance of 2-dimensional knapsack (see [6]). Results are given for order constraints of increasing size. In order to give a better idea of the computational difficulty, we give separate running times for finding an optimal feasible solution, and for proving that this solution is best possible.

See Table 3 for the exact sizes of the 17 rectangles involved for the geometric layout of optimal packings. For easier reference, the boxes are labeled 1–17 in the order in which they are listed in the table.

The second class of instances arises from the well-known tiling of a 112x112 square by 21 squares of different sizes. Again, we have added order constraints of various sizes. (See Table 4 for details.) For the instance square21-2mat (with order constraints in two dimensions), we could not close the gap between upper and lower bound. For this instance, we report the running times for achieving the best known bounds.

Two examples of resulting packings are shown in Figures 8 and 9.

Acknowledgments. We are extremely grateful to Jörg Schepers for letting us continue the work with the packing code that he started as part of his thesis, and for several helpful hints, despite of his departure to industry. We also thank Marc Uetz for a useful discussion on resource-constrained scheduling.

References

1. Atmel. *AT6000 FPGA configuration guide*. Atmel Inc.
2. J. E. Beasley. An exact two-dimensional non-guillotine cutting 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. S. P. Fekete, E. Köhler, and J. Teich. Optimal FPGA Module Placement with Temporal Precedence Constraints. In: *Proc. DATE 2001, Design, Automation and Test in Europe*, pp. 658–665.
5. S. P. Fekete, E. Köhler, and J. Teich. Extending partial suborders and implication classes. Technical Report 697-2000, TU Berlin.
6. S. P. Fekete and J. Schepers. A new exact algorithm for general orthogonal d-dimensional knapsack problems. In *Algorithms – ESA '97*, volume 1284, pp. 144–156, Springer Lecture Notes in Computer Science, 1997.
7. S. P. Fekete and J. Schepers. New classes of lower bounds for bin packing problems. In *Proc. Integer Programming and Combinatorial Optimization (IPCO'98)*, volume 1412, pp. 257–270, Springer Lecture Notes in Computer Science, 1998.
8. S. P. Fekete and J. Schepers. On more-dimensional packing I: Modeling. Technical Report 97-288, Center for Applied Computer Science, Universität zu Köln, available at `http://www.zpr.uni-koeln.de/ABS/~papers`, 1997.
9. S. P. Fekete and J. Schepers. On more-dimensional packing II: Bounds. Technical Report 97-289, Universität zu Köln, 1997.
10. S. P. Fekete and J. Schepers. On more-dimensional packing III: Exact algorithms. Technical Report 97-290, Universität zu Köln, 1997.
11. T. Gallai. Transitiv orientierbare Graphen. *Acta Math. Acd. Sci. Hungar.*, **18** (1967), pp. 25–66.

12. A. Ghouilà-Houri. Caractérization des graphes non orientés dont on peut orienter les arrêtes de manière à obtenir le graphe d'une relation d'ordre. *C.R. Acad. Sci. Paris*, **254** (1962), pp. 1370–1371.
13. P.C. Gilmore and A.J. Hoffmann. A characterization of comparability graphs and of interval graphs. *Canadian Journal of Mathematics*, **16** (1964), pp. 539–548.
14. M. Golumbic. *Algorithmic graph theory and perfect graphs.* Academic Press, 1980.
15. E. Hadjiconstantinou and N. Christofides. An exact algorithm for general, orthogonal, two-dimensional knapsack problems. *European J. of Operations Research*, **83** (1995), 39–56.
16. C.-H. Huang and J.-Y. Juang. A partial compaction scheme for processor allocation in hypercube multiprocessors. In *Proc. of 1990 Int. Conf. on Parallel Proc.*, pp. 211–217, 1990.
17. D. Kelly. Comparability graphs. In I. Rival, editor, *Graphs and Order*, pp. 3–40. D. Reidel Publishing Company, Dordrecht, 1985.
18. N. Korte and R. Möhring. Transitive orientation of graphs with side constraints. In H. Noltemeier, editor, *Proceedings of WG'85*, pp. 143–160. Trauner Verlag, 1985.
19. N. Korte and R. H. Möhring. An incremental linear–time algorithm for recognizing interval graphs. *SIAM Journal of Computing*, **18** (1989), pp. 68-81.
20. A. Krämer. *Scheduling Multiprocessor Tasks on Dedicated Processors.* Doctoral thesis, Fachbereich Mathematik und Informatik, Universität Osnabrück, 1995.
21. E. L. Lawler, J. K. Lenstra, A. H. G. Rinooy Kan, and D. B. Shmoys. *Sequencing and Scheduling: Algorithms and Complexity.* in: S. C. Graves, A. H. G. Rinnooy Kan, and P. H. Zipkin. Logistics of Production and Inventory, vol. 4, Handbooks in Operations Research and Management, pp. 445–522. North–Holland, Amsterdam, 1993,
22. R. H. Möhring. Algorithmic aspects of comparability graphs and interval graphs. In I. Rival, editor, *Graphs and Order*, pages 41–101. D. Reidel Publishing Company, Dordrecht, 1985.
23. R. H. Möhring. Algorithmic aspects of the substitution decomposition in optimization over relations, set systems, and Boolean functions. *Annals of Oper. Res.*, **4** (1985), pp. 195–225.
24. R. H. Möhring, A. S. Schulz, F. Stork, and M. Uetz. Solving Project Scheduling Problems by Minimum Cut Computations. Technical Report 680-2000, TU Berlin.
25. M. Padberg. Packing small boxes into a big box. *Math. Meth. of Op. Res.*, **52** (2000), pp. 1–21.
26. J. Schepers. Exakte Algorithmen für orthogonale Packungsprobleme. Doctoral thesis, Universität Köln, 1997, available as Technical Report 97-302.
27. J. Teich, S. Fekete, and J. Schepers. Compile-time optimization of dynamic hardware reconfigurations. In *Proc. Int. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA'99)*, pp. 1097–1103, Las Vegas, U.S.A., June 1999.
28. J. Teich, S. Fekete, and J. Schepers. Optimization of dynamic hardware reconfigurations. *J. of Supercomputing*, **19** (2001), pp. 57–75.
29. J. Weglarz. *Project Scheduling. Recent Models, Algorithms and Applications.* Kluwers Academic Publishers, Norwell, MA, USA, 1999.
30. Xilinx. XC6200 field programmable gate arrays. Tech. report, Xilinx, Inc., October 1996.