Technische Universität Berlin

FACHBEREICH 3
MATHEMATIK

# OPTIMAL FPGA MODULE PLACEMENT WITH TEMPORAL PRECEDENCE CONSTRAINTS

by

SÁNDOR P. FEKETE    EKKEHARD KÖHLER
JÜRGEN TEICH

No. 696/2000

# Optimal FPGA Module Placement with Temporal Precedence Constraints

Sándor P. Fekete* Ekkehard Köhler† Jürgen Teich‡

## Abstract

*We consider the optimal placement of hardware modules in space and time for FPGA architectures with reconfiguration capabilities, where modules are three-dimensional boxes, with two dimensions corresponding to spatial cell requirements on the array and the third one describing execution time. Thus, optimal module placement can be modeled as a three-dimensional packing problem. A novel graph-theoretic characterization (by so-called "packing classes") of feasible packings and efficient families of lower bounds allow a drastic reduction of the search space, so that it is possible to solve the following problems for a given set of module tasks to optimality:*

*(a) Find the minimal execution time of the given problem on an FPGA of fixed size,*

*(b) Find the FPGA of minimal size to accomplish the tasks within a fixed time limit.*

*Moreover, our approach allows the treatment of precedence constraints for the sequence of tasks, which are present in virtually all practical instances. These additional constraints cause serious problems to standard combinatorial algorithms. We show how our approach of packing classes is perfectly suited for this type of problem. Additional mathematical structures are developed that lead to a powerful framework for computing optimal solutions. The usefulness is validated by computational results.*

## 1 Introduction

A Field-Programmable Gate Array (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). A particular design is realized by customizing a configuration: In traditional SRAM-based chips, this can be done at power-up by loading a configuration bit-stream serially into the chip. These chips may only be reconfigured as a whole with typical reconfiguration times ranging in the order of milliseconds.

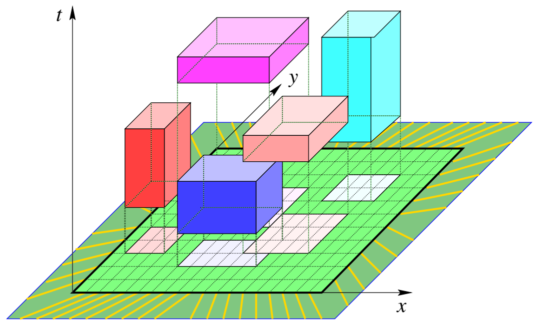


**Figure 1. An FPGA and a set of five modules (tasks), shown in ordinary two-dimensional space and in three-dimensional space-time. Modules must be placed inside the chip and must not overlap if executed simultaneously on the chip.**

Today, new generations of FPGA's have become partitionable and dynamically reconfigurable, even partially. These chips (see e.g. [1, 24]) may support several independent or interdependent tasks and designs at a time, and parts of the chip can be reconfigured quickly during run-time.

In the following, we consider architectures similar to the Xilinx 6200 FPGA [24] architecture, where column read-ins and read-outs of flipflop contents may be performed during run-time without interfering other configured parts of the chip. Under these assumptions, a task, or module, may be represented by a cuboid, with two spatial dimensions, and one representing the time of computation, see Fig. 2.

However, even if the configuration time is short, the compilation time for constructing the configuration stream for a task is still rather long. This diminishes the results that have been reported recently on on-line strategies for compiling and reconfiguring such devices. Important examples include speeding up computational problems in hardware by task compaction on hypercubes [16], or approaches to

*Department of Mathematics, TU Berlin, Berlin, Germany, email: fekete@math.tu-berlin.de.

†Department of Mathematics, TU Berlin, Berlin, Germany, email: ekoehler@math.tu-berlin.de.

‡Computer Engineering Laboratory, University of Paderborn, Paderborn, Germany, email: teich@date.upb.de.

dynamic allocation of a sequence of tasks on an FPGA of given size by using heuristics to compact tasks in execution on the chip during run-time [3, 4].

Here we consider *statically defined problems where a task set is given.* In previous work reported in [22, 23] we have described how these problems can be understood by virtue of an easy graph-theoretic characterization of feasible packings. In this paper, we show how to extend this approach in order to deal with very restrictive constraints that are present in virtually all practical instances: Typically, there are *temporal precedence constraints* imposed on the set of computing modules, since the output of one task may be needed as input for another task. Such a set of precedence constraints may be described by a dependency graph, see Figure 2. For a problem instance of this type, we are interested in finding exact solutions to the following problems. (In the following, a spatial placement is called *feasible* if the locations occupied by each pair of tasks that have overlapping execution intervals are disjoint and fit into the available space and time. In the presence of precedence constraints for the tasks, *feasibility of a schedule* implies that all of these constraints are met.)

- Find the chip of smallest size to accommodate all tasks such that a given maximum total execution-time is satisfied (*MinA&FindS*) together with a feasible schedule. A subproblem called *MinA&FixedS* arises when the precise starting times of all tasks are already given.
- Check whether for a chip of given size and given maximum execution time, there is a feasible placement and a feasible schedule that accommodates a set of tasks (*FeasAT&FindS*).
- Find the smallest execution time of the set of tasks for a chip of fixed size (*MinT&FindS*).
- Check whether a chip of given size and a given feasible schedule allow a feasible placement (*FeasA&FixedS*).

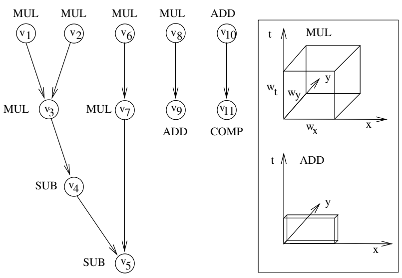


**Figure 2. Dependency graph of tasks and shape of modules (3D boxes) with the spatial dimensions $x$ and $y$ and the temporal dimension $t$ (execution time)**

Our results reported in [23] deal with solving the above *FixedS*-problems. For these cases, the problems may be modeled by three-dimensional orthogonal packing problems (OPP) without precedence constraints. Note that already the one-dimensional counterparts without precedence constraints are known to be NP-complete in the strict sense [13]. Existing ILP formulations for similar problems such as [2] fail to solve technical problems of interesting size to optimality because they use a grid decomposition and model the placement of a module at location $(x, y)$ and time $t$ by a 0-1-variable, requiring $\mathcal{O}(|V||X||Y|)$ 0-1 variables and $\mathcal{O}(|X||Y||T|)$ constraints where $|X|$, $|Y|$, $|T|$ are the dimensions of the underlying grid in the $x$-, $y$-, and $t$-direction. The largest two-dimensional packing problems that have been solved with this technique place about 20 rectangles on a $30 \times 30$ grid [2, 15]. Solving a three-dimensional problem with about 100 nodes is hopeless if these standard solution techniques are used.

A breakthrough to solve these problems to optimality was due to the introduction of so-called *packing classes* [7, 21] that drastically reduce the search space for feasible packings. The application of this idea to *FixedS*-problems in the context of FPGA module reconfiguration has been elaborated in [22, 23]. Here, we develop a framework to solve also problems with precedence constraints such as given by data dependencies.

In Section 2, we describe basic assumptions on architecture and introduce some terminology. The notion of packing classes and a solution to packing problems without precedence constraints is summarized in Section 3. In Section 4, 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 two realistic benchmarks in Section 5.

## 2 Preliminaries

### 2.1 Architecture Assumptions

**Intermodule communication.** Intermodule communication is assumed to occur at the end of operation of the sending module (task model). The issuing module may store its result register values into an external memory connected to the FPGA interface (read-out) via a bus interface. Memory is allocated to store temporarily intermediate results.[1] Afterwards, the receiving module will read the communicated data into its registers via the bus interface. With this communication style, it is justifiable to ignore routing overhead between modules that otherwise might introduce additional placement constraints.

**I/O-overhead.** The communication time needed for writing out and reading in communicated data may be ac-

[1] A static memory allocation may be deduced directly from the static placement.

counted for by considering this as an offset and being part of the execution time of a task.

**Reconfiguration overhead.** The time needed for carrying out reconfigurations may be modeled by a constant (possibly a different number for each task), depending on the target architecture. This may be considered a simplification because the reconfiguration time might depend on the result of the placement. However, many different models of taking into account reconfiguration times can be thought of, and should be adapted individually to the target architecture.

### 2.2 Mathematical Terminology

**Problem instances.** We assume that a problem instance is given by a *set of tasks* $V$. Each task has a *spatial requirement* in the $x$- and $y$-direction, denoted by $w_x(v)$ and $w_y(v)$, and a *duration*, denoted by a size $w_t(v)$ along the time axis. The reconfigurable chip $H$ consists of an array of $h_x \times h_y$ cells. In addition, there may be an overall allowable time $h_t$ for all tasks to be completed. A *schedule* is given by a start time $p_t(v)$ for each task. A schedule is *feasible*, if all tasks can be carried out without overlap of computation tasks in time or space, such that all tasks are within spatial and temporal bounds.

**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 $w$.

**Precedence constraints.** There may be a temporal precedence requirement between some of the tasks, since some tasks need to be finished before others can get started. Mathematically, this means that we are given a partial order on $V$, which can be described by a directed acyclic graph $D = (V, A)$, where $A$ is the set of directed arcs. In the presence of such a partial order (denoted $A$), a feasible schedule also needs to satisfy these additional constraints.

**Packing problems.** In the following, we will treat tasks as three-dimensional boxes and feasible schedules as arrangements of boxes that satisfy all side constraints. This is implied by the term of a *feasible packing*. As described in the introduction, there are different types of objectives, 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 $h_x \times h_y \times h_t$. The *Base Minimization Problem* (BMP) is to minimize the size $h_x$ for a fixed $h_t$ such that all boxes fit into a container $h_x \times h_x \times h_t$ with quadratic base. This corresponds to minimizing chip size to carry out a set of computations within a given time – called *MinA&FindS* in the introduction. The *Strip Packing Problem* (SPP) 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 $h_x \times h_y \times h_t$. This corresponds to minimizing the time to carry out a set of computations on a given chip – called MinT&FindS in the introduction.

## 3 Solving Unconstrained Problems

### 3.1 A Framework for Optimal Solutions

Before discussing precedence constraints, we describe a number of fundamental mathematical insights and resulting computational methods for unconstrained packing problems. Mathematical details can be found in our previous papers [7, 8, 9, 10, 11, 21, 22, 23].

If we have an efficient method for solving OPP's, we can also solve BMP's and SPP's by using a binary search. However, deciding the existence of a feasible more-dimensional packing is a hard problem in higher dimensions, and proposed methods suggested by other authors [2, 15] have been of limited success.

Our framework uses a combination of different approaches to overcome these problems:

1. Try to disprove the existence of a packing by fast and good classes of lower bounds on the necessary size.
2. In case of failure, try to find a feasible packing by using fast heuristics.
3. If the existence of a packing is still unsettled, start an enumeration scheme in form of a branch-and-bound tree search.

By developing good new bounds for the first stage, we have been able to achieve a considerable reduction of the number of cases where a tree search needs to be performed. (Mathematical details for this step are described in [8, 10] and are omitted from this short paper.) However, it is clear that the efficiency of the third stage is crucial for the overall running time when considering difficult problems. Using a purely geometric enumeration scheme for this step by trying to build a partial arrangement of boxes is easily seen to be immensely time-consuming. In the following, we describe a purely combinatorial characterization of feasible packings that allows to perform this step more efficiently.

### 3.2 Packing Classes

If we consider a feasible packing in $d$-dimensional space, we can extract some partial information by considering the relative arrangement of coordinate intervals. More precisely, we can consider the projections of the boxes onto the three coordinate axes, and thus reduce the one $d$-dimensional arrangement to $d$ one-dimensional ones. (See Figure 3 for an example in $d = 2$.) In a second step, we can disregard the exact coordinates of the resulting intervals in direction $i$ and only consider the *component graph* $G_i = (V, E_i)$: Two boxes $u$ and $v$ are connected by an edge in $G_i$, iff they have overlapping $i$ coordinates. Mathematically, a graph with this characterization of edges is called an *interval graph*. These graphs have been studied intensively in graph theory (see [14, 20]), and they have a number of very useful algorithmic properties.

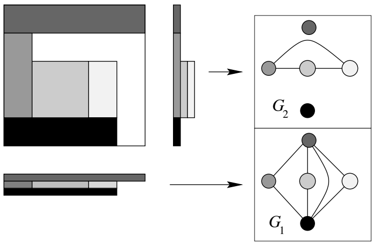


**Figure 3. The projections of the boxes onto the coordinate axes define interval graphs (here in 2D: $G_1$ and $G_2$).**

Considering sets of $d$ component graphs $G_i$ instead of complicated geometric arrangements has some clear advantages. (Algorithmic implications for our specific purposes will be discussed further down.) It is not hard to check that the following three conditions must be satisfied by all $d$-tuples of graphs $G_i$ that are constructed from a feasible packing:

**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., $w_i(S) = \sum_{v \in S} w_i(v) \leq h_i$, since all boxes in $S$ must fit into the container in the $i$th 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 [21, 9]) 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 allows it to consider only packing classes in order to decide the existence of a feasible packing, and disregard most of the geometric information.

### 3.3 Solving OPP's

Our search procedure works on packing classes, i.e., triples of component graphs with the properties C1, C2, C3. Since each packing class represents not only a single packing but a whole family of equivalent packings, we are effectively dealing with more than one possible candidate for an optimal packing at a time. (The reader may check for the example in Figure 3 that there are 36 different feasible packings that correspond to the same packing class.)

The search tree is traversed by Depth First Search, see [11, 21] for details. Branching is done by fixing an edge $\{b, c\} \in E_i$ or $\{b, c\} \notin E_i$. After each branching step, it is checked if one of the three conditions C1, C2, C3 is violated, or whether a violation can only be avoided by fixing further edges. This is easy for two of the conditions: enforcing C3 is obvious; property C2 is hereditary, so adding edges to $E_i$ later will keep it satisfied. (Note that computing maximum weighted cliques on comparability graphs can be done efficiently, see [14].) In order to ensure that property C1 is not violated, we use a number of graph-theoretic characterizations of interval graphs and comparability graphs. These characterizations are based on two forbidden substructures (again, see [14] for details). In particular, this means that the following configurations have to be avoided:

1. induced chordless cycles of length 4 in $E_i$;
2. so-called 2-chordless odd cycles in the set $\overline{E_i}$ of edges excluded from $E_i$ (see [11, 14] for details);
3. infeasible stable sets in $E_i$.

Each time we detect such a fixed subgraph, we can abandon the search on this node. Furthermore, if we detect a fixed subgraph, except for one unfixed edge, we can fix this edge, such that the forbidden subgraph is avoided.

Our experience shows that these conditions are already useful when only small subsets of edges have been fixed, since by excluding small sub-configurations, like induced chordless cycles of length 4, each branching step triggers a cascade of more fixed edges.

## 4 Solving Optimization Problems with Precedence Constraints

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 tasks. It should be stressed that for standard approaches, adding constraints makes the three-dimensional packing problems much harder. This is significantly different from our approach, where the nature of the data structures simplifies these problems from three-dimensional 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 two graphs[2]. As we have worked out in detail in [22, 23], this allows it to solve the resulting FixedS-Problems quite efficiently.

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.

### 4.1 Packing Classes and Interval Orders

Any edge in a graph $G_i$ corresponds to an overlap between the corresponding intervals. 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* [20]. 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 4 for a (two-dimensional) example of a packing class, the corresponding comparability graph, a transitive orientation, and the packing corresponding to the transitive orientation.

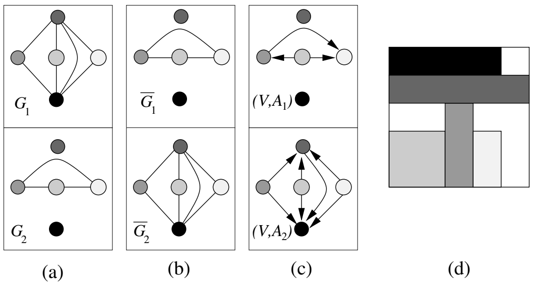


**Figure 4. (a) A two-dimensional packing class. (b) The corresponding comparability graphs. (c) A transitive orientation. (d) A feasible packing corresponding to the orientation.**

Now consider a situation where we need to satisfy a partial order $A$ of precedence constraints in the time dimension. It follows that each arc $a = (u, w) \in A$ 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 $A$ to be contained in $\overline{E_t}$. After running the original algorithm, we may get additional comparability edges. As the example in Figure 5 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 $(V, A)$ 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 $A$.

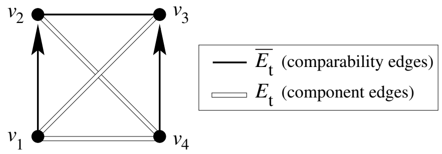


**Figure 5. A comparability graph $\overline{G_t} = (V, \overline{E_t})$ with a partial order $A$ contained in $\overline{E_t}$, such that there is no transitive orientation of $\overline{G_t}$ that extends $A$.**

### 4.2 Finding Feasible Transitive Orientations

Consider a comparability graph $\overline{G}$ that is the complement of an interval graph $G$. Deciding whether $\overline{G}$ has a transitive orientation that extends a given partial order $A$ is a problem that has been studied in the context of scheduling. Korte and Möhring [18] give a linear-time algorithm for determining a solution, 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 our more-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.

### 4.3 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

[2] To emphasize the motivation of temporal precedence constraints, we write $E_t$ to suggest that the time coordinate is constrained, and $E_x$ and $E_y$ to imply that the space coordinates are unrestricted. Clearly, our approach works the same way when dealing with spatial restrictions.

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) 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 6:

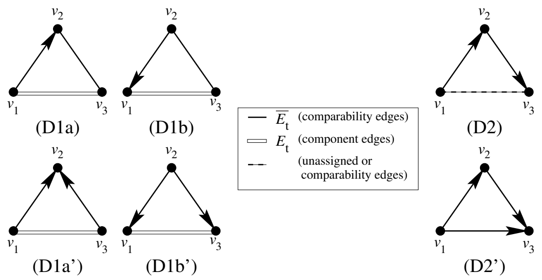


**Figure 6. Implications for edges and their orientations: Above are path implications (D1, left) and transitivity implications (D2, right); below the forced orientations of edges.**

The first configuration consists of 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 figure. Since this configuration corresponds to an induced path in $\overline{G_i}$, we call this arrangement a *path implication*.

The second configuration consists of two directed comparability edges $(v_1, v_2), (v_2, v_3)$. 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 corresponds to a triangle 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. Considering sequences of path implications leads to the following 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 5. 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 partial order $A$. We call this type of contradiction a *path conflict* on a path implication class.

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

Similar to possible orientation conflicts for path implication classes, we may get a violation of transitivity implications, as a sequence of implications may force a directed cycle. (An example can be found in our mathematical report [6].) This type of violation we call a *transitivity conflict*.

Thus, we have the following necessary conditions for the existence of a transitive orientation that extends a given partial order $A$:

**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)** *Consider a partial order $A$ with arc set contained in the edge set of a given comparability graph $G$. $A$ can be extended to a transitive orientation of $G$, iff all arising path implications and transitivity can be carried out in any order without creating a path conflict or a transitivity conflict.*

A proof and further mathematical details[3] are described in our forthcoming mathematical paper [5].

### 4.4 Solving OPP's 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 tests suggested by path implications and triangle implications. 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.

## 5 Computational Experiments

The first example is a numerical method for solving a differential equation (DE) with 11 nodes. The node operations are either multiplications or ALU-type operations. In

[3]The interested reader may take note that we are extending previous work by Gallai [12], who extensively studied implication classes of comparability graphs. See Kelly [17], Möhring [20] for informative surveys on this topic, and Krämer [19] for an application in scheduling theory.

a second example, a video-codec using the H.261 norm is optimized. These examples are meant to demonstrate the general applicability of our method for practical problems; given other problem instances, or additional constraints, we can easily adapt our algorithm.

### 5.1 DE Benchmark

Let the module library contain two hardware modules (box types): an array-multiplier and a module of type ALU that realizes all other node operations (comparison, addition, subtraction). For a word-length of n=16 bits, we assume a module geometry of 16 x 1 cells for the ALU module, and of 16 x 16 cells for the multiplier. Furthermore, the execution time of an ALU node takes one clock cycle, while a multiplication requires 2 clock cycles on our target chip.

The dependency graph is shown in Fig. 2. First, we compute the transitive closure of all data dependencies to allow our algorithm to find contradictions to feasible packings already in the input.

Next, we solve several instances of the BMP problem for different values of $h_t$ reported in Table 1. Each $h_t$ listed yields a test case for which the container size is minimized (*MinA*), assuming $h_x = h_y$. Also shown is the CPU-time needed for finding a solution.

**Table 1. Computational results for optimizing reconfigurations for the DE benchmark**

| $test$ | container sizes | | | CPU-time (s) |
|---|---|---|---|---|
| | $h_t$ | $h_x$ | $h_y$ | |
| 1 | 6 | 32 | 32 | 55.76 s |
| 2 | 13 | 17 | 17 | 0.04 s |
| 3 | 14 | 16 | 16 | 0.03 s |

The reported optimization times were measured as the CPU-times on a SUN-Ultra 30 architecture.

For the DE benchmark, it turns out that a chip of 32 x 32 freely programmable cells is necessary to obtain a latency between 6 and 12 clock cycles. As the longest path in the graph has length 6, there does not exist any faster schedule. For 12 and 13 cycles, a chip of size 17 x 17 is necessary, for $h_t \geq 14$, a chip of size 16 x 16 cells is sufficient which is the smallest chip possible to implement the problem as one multiplication by itself uses the full chip.

Similarly, the SPP is solved. The tradeoff between area size and necessary time is visualized in Fig. 7, where the Pareto-optimal points are shown. The figure also shows the Pareto points for the case where no partial order needs to be satisfied (shown dashed).

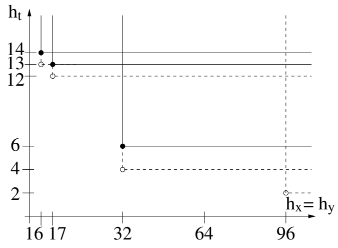


**Figure 7. Pareto-optimal points for minimizing chip area and processing time for the DE benchmark. (a) Including partial order constraints (solid lines). (b) Without consideration of partial order constraints (dashed lines).**

### 5.2 Video-Codec Benchmark

Figure 8 shows a block diagram of the operation of a hybrid image sequence coder/decoder. 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.

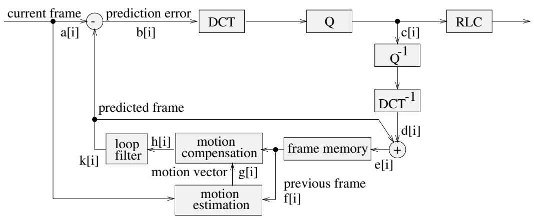


**Figure 8. Block diagram of a video-codec (H.261)**

The blocks of the operational description in Figure 8 possess the granularity of more complex functions. However, this description contains no information corresponding to timing, architecture, and mapping of blocks onto an architecture.

Figure 9 shows a problem graph $G$ of the video-codec in Figure 8. The problem graph contains a subgraph for the coder and one subgraph for the decoder.

For realizing the device, we have a library of three different modules. One is a simple processor core with a (normalized) area requirement of 625 units (25 x 25 cells, normalized to other modules in order to obtain a coarser grid) called PUM. Secondly, there are two dedicated special-purpose modules: a block matching module (BMM) that is used for motion estimation and requires 64 x 64 = 4096

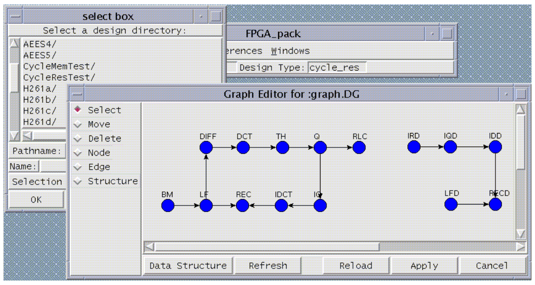


**Figure 9. Problem graph $G$ of the video-codec in Figure 8**

cells; and a module DCTM for computing DCT/IDCT-computations, requiring 16 x 16 = 256 cells.

Again, the BMP and the SSPP were solved for different latency constraints. Here, there is only one Pareto-point found, shown in Table 2.

**Table 2. Computational results for 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 |

Note that there is no solution for container sizes smaller than 64 x 64. For this value, $h_t = 59$ is the smallest latency possible due to the data dependencies.

## Acknowledgment

We are very grateful to Jörg Schepers for letting us use and extend the code for more-dimensional packing that he started as part of his thesis [21].

## References

[1] Atmel. *AT6000 FPGA configuration guide*. Atmel Inc.

[2] J. E. Beasley. An exact two-dimensional non-guillotine cutting tree search procedure. *Op. Research*, 33:49–64, 1985.

[3] O. Diessel and H. ElGhindy. Partial FPGA rearrangement by local repacking. Report 97-08, Dept. of Comp. Sci and Software Eng., Univ. of Newcastle, Australia, 1997.

[4] O. Diessel and H. ElGhindy. Run-time compaction of FPGA designs. In *Proc. of FPL'97–the 7th Int. Workshop on field-programmable logic and applications*, pages 131–140, Berlin, 1997.

[5] S. P. Fekete, E. Köhler, and J. Teich. Extending partial suborders and implication classes. Report 697-2000, TU Berlin.

[6] S. P. Fekete, E. Köhler, and J. Teich. More-dimensional packing with order constraints. Report 698-2000, TU Berlin.

[7] S. P. Fekete and J. Schepers. A new exact algorithm for general orthogonal d-dimensional knapsack problems. In *Algorithms – ESA '97*, volume 1284, pages 144–156, Springer LNCS, 1997.

[8] 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, pages 257–270, Springer LNCS, 1998.

[9] S. P. Fekete and J. Schepers. On more-dimensional packing I: Modeling. Report 97-288, Center for Applied Computer Science, Universität zu Köln, Available at `http://www.zpr.uni-koeln.de/ABS/~papers`, 1997.

[10] S. P. Fekete and J. Schepers. On more-dimensional packing II: Bounds. Report 97-289, Universität zu Köln, 1997.

[11] S. P. Fekete and J. Schepers. On more-dimensional packing III: Exact algorithms. Report 97-290, Universität zu Köln, 1997.

[12] T. Gallai. Transitiv orientierbare Graphen. *Acta Math. Acd. Sci. Hungar.*, 18:25–66, 1967.

[13] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, New York, 1979.

[14] M. Golumbic. *Algorithmic graph theory and perfect graphs*. Academic Press, New York, 1980.

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

[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.*, pages 211–217, 1990.

[17] D. Kelly. Comparability graphs. In I. Rival, editor, *Graphs and Order*, pages 3–40. D. Reidel Publishing, Dordrecht, 1985.

[18] N. Korte and R. Möhring. Transitive orientation of graphs with side constraints. In H. Noltemeier, editor, *Proceedings of WG'85*, pages 143–160. Trauner Verlag, 1985.

[19] A. Krämer. *Scheduling Multiprocessor Tasks on Dedicated Processors*. Doctoral thesis, Fachbereich Mathematik und Informatik, Universität Osnabrück, 1995.

[20] 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.

[21] J. Schepers. Exakte Algorithmen für orthogonale Packungsprobleme. Doctoral thesis, Universität Köln, 1997, available as Report 97-302.

[22] 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)*, pages 1097–1103, Las Vegas, U.S.A., June 1999.

[23] J. Teich, S. Fekete, and J. Schepers. Optimization of dynamic hardware reconfigurations. *J. of Supercomputing, to appear*, 2000.

[24] Xilinx. XC6200 field programmable gate arrays. Technical report, Xilinx, Inc., October 1996.

Reports from the group

## "Combinatorial Optimization and Graph Algorithms"

of the Department of Mathematics, TU Berlin

**702/2000** *Frederik Stork:* Branch-and-Bound Algorithms for Stochastic Resource-Constrained Project Scheduling

**698/2000** *Sándor P. Fekete, Ekkehard Köhler, and Jürgen Teich:* More-dimensional packing with order constraints

**697/2000** *Sándor P. Fekete, Ekkehard Köhler, and Jürgen Teich:* Extending partial suborders and implication classes

**696/2000** *Sándor P. Fekete, Ekkehard Köhler, and Jürgen Teich:* Optimal FPGA module placement with temporal precedence constraints

**695/2000** *Sándor P. Fekete, Henk Meijer, André Rohe, and Walter Tietze:* Solving a "hard" problem to approximate an "easy" one: heuristics for maximum matchings and maximum Traveling Salesman Problems

**694/2000** *Esther M. Arkin, Sándor P. Fekete, Ferran Hurtado, Joseph S. B. Mitchell, Marc Noy, Vera Sacristánm and Saurabh Sethia:* On the reflexivity of point sets

**693/2000** *Frederik Stork and Marc Uetz:* On the representation of resource constraints in project scheduling

**691/2000** *Martin Skutella and Marc Uetz:* Scheduling precedence constrained jobs with stochastic processing times on parallel machines

**689/2000** *Rolf H. Möhring, Martin Skutella, and Frederik Stork:* Scheduling with AND/OR precedence constraints

**685/2000** *Martin Skutella:* Approximating the single source unsplittable min-cost flow problem

**684/2000** *Han Hoogeveen, Martin Skutella, and Gerhard J. Woeginger:* Preemptive scheduling with rejection

**683/2000** *Martin Skutella:* Convex quadratic and semidefinite programming relaxations in Scheduling

**682/2000** *Rolf H. Möhring and Marc Uetz:* Scheduling scarce resources in chemical engineering

**681/2000** *Rolf H. Möhring:* Scheduling under uncertainty: optimizing against a randomizing adversary

**680/2000** *Rolf H. Möhring, Andreas S. Schulz, Frederik Stork, and Marc Uetz:* Solving project scheduling problems by minimum cut computations (Journal version for the previous Reports 620 and 661)

**674/2000** *Esther M. Arkin, Michael A. Bender, Erik D. Demaine, Sándor P. Fekete, Joseph S. B. Mitchell, and Saurabh Sethia:* Optimal covering tours with turn costs

**669/2000** *Michael Naatz:* A note on a question of C. D. Savage

**667/2000** *Sándor P. Fekete and Henk Meijer:* On geometric maximum weight cliques

**666/2000** *Sándor P. Fekete, Joseph S. B. Mitchell, and Karin Weinbrecht:* On the continuous Weber and $k$-median problems

**664/2000** *Rolf H. Möhring, Andreas S. Schulz, Frederik Stork, and Marc Uetz:* A note on scheduling problems with irregular starting time costs

**661/2000** *Frederik Stork and Marc Uetz:* Resource-constrained project scheduling: from a Lagrangian relaxation to competitive solutions

**658/1999** *Olaf Jahn, Rolf H. Möhring, and Andreas S. Schulz:* Optimal routing of traffic flows with length restrictions in networks with congestion

**655/1999** *Michel X. Goemans and Martin Skutella:* Cooperative facility location games

**654/1999** *Michel X. Goemans, Maurice Queyranne, Andreas S. Schulz, Martin Skutella, and Yaoguang Wang:* Single machine scheduling with release dates

**653/1999** *Andreas S. Schulz and Martin Skutella:* Scheduling unrelated machines by randomized rounding

**646/1999** *Rolf H. Möhring, Martin Skutella, and Frederik Stork:* Forcing relations for AND/OR precedence constraints

**640/1999** *Foto Afrati, Evripidis Bampis, Chandra Chekuri, David Karger, Claire Kenyon, Sanjeev Khanna, Ioannis Milis, Maurice Queyranne, Martin Skutella, Cliff Stein, and Maxim Sviridenko:* Approximation schemes for minimizing average weighted Completion time with release dates

**639/1999** *Andreas S. Schulz and Martin Skutella:* The power of $\alpha$-points in preemptive single machine scheduling

**634/1999** *Karsten Weihe, Ulrik Brandes, Annegret Liebers, Matthias Müller–Hannemann, Dorothea Wagner and Thomas Willhalm:* Empirical design of geometric algorithms

**633/1999** *Matthias Müller–Hannemann and Karsten Weihe:* On the discrete core of quadrilateral mesh refinement

**632/1999** *Matthias Müller–Hannemann:* Shelling hexahedral complexes for mesh generation in CAD

**631/1999** *Matthias Müller–Hannemann and Alexander Schwartz:* Implementing weighted $b$-matching algorithms: insights from a computational study

**629/1999** *Martin Skutella:* Convex quadratic programming relaxations for network scheduling problems

**628/1999** *Martin Skutella and Gerhard J. Woeginger:* A PTAS for minimizing the total weighted completion time on identical parallel machines

**624/1999** *Rolf H. Möhring:* Verteilte Verbindungssuche im öffentlichen Personenverkehr: Graphentheoretische Modelle und Algorithmen

**627/1998** *Jens Gustedt:* Specifying characteristics of digital filters with FilterPro

**620/1998** *Rolf H. Möhring, Andreas S. Schulz, Frederik Stork, and Marc Uetz:* Resource constrained project scheduling: computing lower bounds by solving minimum cut problems

**619/1998** *Rolf H. Möhring, Martin Oellrich, and Andreas S. Schulz:* Efficient algorithms for the minimum-cost embedding of reliable virtual private networks into telecommunication networks

**618/1998** *Friedrich Eisenbrand and Andreas S. Schulz:* Bounds on the Chvátal rank of polytopes in the 0/1-Cube

**617/1998** *Andreas S. Schulz and Robert Weismantel:* An oracle-polynomial time augmentation algorithm for integer proramming

**616/1998** *Alexander Bockmayr, Friedrich Eisenbrand, Mark Hartmann, and Andreas S. Schulz:* On the Chvátal rank of polytopes in the 0/1 cube

**615/1998** *Ekkehard Köhler and Matthias Kriesell:* Edge-dominating trails in AT-free graphs

**613/1998** *Frederik Stork:* A branch and bound algorithm for minimizing expected makespan in stochastic project networks with resource constraints

**612/1998** *Rolf H. Möhring and Frederik Stork:* Linear preselective policies for stochastic project scheduling

**611/1998** *Rolf H. Möhring and Markus W. Schäffter:* Scheduling series-parallel orders subject to 0/1-communication delays

**609/1998** *Arfst Ludwig, Rolf H. Möhring, and Frederik Stork:* A computational study on bounding the makespan distribution in stochastic project networks

**605/1998** *Friedrich Eisenbrand:* A note on the membership problem for the elementary closure of a polyhedron

**596/1998** *Andreas Fest, Rolf H. Möhring, Frederik Stork, and Marc Uetz:* Resource constrained project scheduling with time windows: A branching scheme based on dynamic release dates

**595/1998** *Rolf H. Möhring Andreas S. Schulz, and Marc Uetz:* Approximation in stochastic scheduling: The power of LP-based priority policies

**591/1998** *Matthias Müller–Hannemann and Alexander Schwartz:* Implementing weighted $b$-matching algorithms: Towards a flexible software design

**590/1998** *Stefan Felsner and Jens Gustedt and Michel Morvan:* Interval reductions and extensions of orders: bijections to chains in lattices

**584/1998** *Alix Munier, Maurice Queyranne, and Andreas S. Schulz:* Approximation bounds for a general class of precedence constrained parallel machine scheduling problems

**577/1998** *Martin Skutella:* Semidefinite relaxations for parallel machine scheduling

Reports may be requested from: Hannelore Vogt-Möller
Fachbereich Mathematik, MA 6–1
TU Berlin
Straße des 17. Juni 136
D-10623 Berlin – Germany

e-mail: moeller@math.TU-Berlin.DE

Reports are also available in various formats from

`http://www.math.tu-berlin.de/coga/publications/techreports/`

and via anonymous ftp as

`ftp://ftp.math.tu-berlin.de/pub/Preprints/combi/Report-`*number*`-`*year*`.ps`