Scheduling and Packing Malleable and Parallel Tasks with Precedence Constraints of Bounded Width

Elisabeth Günther[*] Felix G. König[*] Nicole Megow[†]

Preprint 2009/20

June 1, 2009

Revised: October 26, 2010

Abstract

We study the problems of non-preemptively scheduling and packing malleable and parallel tasks with precedence constraints to minimize the makespan. In the scheduling variant, we allow the free choice of processors; in packing, each task must be assigned to a contiguous subset. Malleable tasks can be processed on different numbers of processors with varying processing times, while parallel tasks require a fixed number.

For precedence constraints of bounded width, we resolve the complexity status of every particular problem setting for any width bound and number of processors. We present an FPTAS based on Dilworth's decomposition theorem for the NP-hard problem variants, and exact efficient algorithms for all remaining special cases. For our positive results, we do not require the otherwise common monotonous penalty assumption on the processing times of malleable tasks, whereas our hardness results hold even when assuming this restriction. We complement our results by showing that these problems are all strongly NP-hard under precedence constraints which form a tree.

1 Introduction

Parallelism plays a key role in high performance computing. The apparent need for adequate models and algorithms for scheduling parallel task systems has attracted significant attention in scheduling theory over the past decade [5, 20]. Several models have been considered, among which scheduling malleable tasks as proposed in [29] is an important and promising model [17].

In the problem of scheduling malleable tasks, we are given a set $J = \{1, 2, \dots, n\}$ of tasks and m identical parallel processors. The tasks are *malleable*, which means that the processing time of a task $j \in J$ is a function $p_j(\alpha_j)$ depending on the number of processors $\alpha_j \in \mathbb{N}$ allotted to it. The tasks must be processed non-preemptively, respecting precedence constraints given by a partial order $(J, \prec)$: For any $i, j \in J$, let $i \prec j$ denote that task i must be completed before task j starts processing. Two tasks are called *incomparable* if neither $i \prec j$ nor $j \prec i$, otherwise, they are called *comparable*. The *width* ω of a partial order is the maximum number of pairwise incomparable tasks.

[*]Technische Universität Berlin, Institut für Mathematik, 10623 Berlin, Germany, Email: {eguenth,fkoenig}@math.tu-berlin.de

[†]Max-Planck-Institut für Informatik, Campus E1 4, 66123 Saarbrücken, Germany, Email: nmegow@mpi-inf.mpg.de

An *allotment* $(\alpha_j)_{j\in J}$ and an assignment of start times $\sigma_j \geq 0$ for each task $j \in J$ establish a *feasible schedule* if the precedence constraints are respected and at no point in time the number of required processors exceeds the number of available processors m. The goal is to find a feasible schedule of minimum total length, called makespan.

Virtually all previous literature on malleable scheduling with precedence constraints, requires the *monotonous penalty assumption*, which ensures that for any malleable task j, its processing time function $p_j(\alpha_j)$ is non-increasing, and its work function $\alpha_j p_j(\alpha_j)$ is non-decreasing. In this work, we abandon this restriction, such that an arbitrary set of feasible allotments can be prescribed (by simply setting the task duration for forbidden allotments to some large constant).

The special situation where one fixed allotment is given and only start times for the tasks are sought corresponds to the well-studied problem of scheduling *parallel tasks*, see [5, 20] for an overview. Parallel tasks cannot be modeled in malleable scheduling when assuming monotonous penalties, as this restriction prohibits enforcing a fixed allotment by appropriate processing time functions. In our model, however, parallel task scheduling becomes a special case as long as m is polynomially bounded in the input. This assumption is reasonable and quite common, see [16]. It is necessary at this point, because the input encoding of the malleable task scheduling problem is polynomial in m, while the input of a parallel task scheduling problem is polynomial in $\log m$.

We also consider the *contiguous* variant of the malleable scheduling problem in which we require that each task is processed on a subset of processors with consecutive indices. That such a schedule is desirable in certain applications is mentioned already in [29]. Contiguously scheduling parallel tasks corresponds to *strip packing*, i.e., the problem of packing rectangles in a strip of width one such that the packing height is minimized, see e.g. [21]. More generally, we define the *discrete malleable strip packing problem* as strip packing with malleable rectangles: For strip width m, each rectangle j may have a width $\alpha_j \in \{1,\dots,m\}$, and the height of j is a function depending on α_j. Our results for malleable scheduling also hold for this case of malleable packing.

1.1 Related Work

We commence our overview of previous work with scheduling parallel tasks before moving on to malleable scheduling. For each of the problems, we focus on results for different classes of precedence constraints, contrasting contiguous to non-contiguous approaches.

Parallel Scheduling A remarkable amount of literature deals with scheduling parallel tasks, see [5, 20] for an overview. For independent tasks, constant factor approximation algorithms are known for the non-contiguous [10] as well as the contiguous case, i.e., strip packing [26, 27]. Approximation ratios can be improved in both cases when the number of machines in polynomial in input size [16]. Moreover, Duin and van der Sluis [7] investigate the problem of finding a contiguous processor assignment for a given parallel task schedule in the context of assigning check-in counters at airports; they call it *adjacent resource scheduling*.

In the presence of precedence constraints, most results have been obtained in the context of strip packing. Augustine et al. [1] consider the case of arbitrary precedence constraints; they present a factor $\Theta(\log n)$ approximation. As lower bounds, they use the longest chain, i.e., the largest set of pairwise comparable tasks, and the total volume to be packed. Since these bounds immediately apply for non-contiguous scheduling, the approximation guarantee carries over. Moreover, the authors provides a packing instance for which the gap between the optimal value and both lower bounds is indeed $\log n$, which indicates that new ideas and bounds are necessary for improving on this performance guarantee in the general case.

Abandoning contiguousness, the special case of chain precedence constraints is considered in [3]. Therein, Błażewicz and Liu show that scheduling unit size parallel tasks with precedence constraints that form chains is NP-hard already for three processors. Assuming monotonously increasing (decreasing) processing times along chains, they give polynomial-time algorithms.

Malleable Scheduling Quite some research has been dedicated to malleable scheduling for independent tasks,see [2, 6, 14, 15, 22, 24, 25, 29]. Notably, Turek et al. [29] were the first to describe a general method to employ algorithms for independent parallel tasks to obtain similar results for the malleable case. When using a contiguous parallel scheduling algorithm, the contiguousness of solutions carries over to malleable tasks.

In the precedence constrained case, approximation results rely crucially on the abovementioned monotonous penalty assumption and do not yield contiguous solutions. Interestingly, this allows for results much superior to the case of parallel tasks. For general precedence constraints, Lepère et al. [19] provide an approximation algorithm with performance guarantee $3+\sqrt{5}\approx 5.236$. For special cases, such as series-parallel precedence constraints and such of bounded width, they prove a ratio of $(3+\sqrt{5})/2\approx 2.618$ in the same paper, improving on an earlier factor $4+\varepsilon$ approximation for trees in [18]. Jansen and Zhang [17] consider the case of general precedence constraints and provide an algorithm with performance guarantee ≈ 4.730598, which they show to be asymptotically tight.

When contiguousness is required, the approximation factor of $\Theta(\log n)$ for parallel tasks with arbitrary precedence constraints in [1] can be transferred to malleable task scheduling with the techniques in [19] or [17] if the monotonous penalty assumption holds, see [13].

1.2 Our Results

We derive a fully polynomial-time approximation scheme (FPTAS) for scheduling malleable tasks under precedence constraints of bounded width ω in Section 2. As we do not require monotonous penalties, our FPTAS also covers the case of parallel tasks. We are not aware of a previous constant factor approximation for scheduling parallel tasks with precedence constraints, while the previously best known approximation ratio for malleable tasks was $(3+\sqrt{5})/2\approx 2.618$ under the restriction to monotonous penalty functions [19]. The algorithm is a dynamic programming scheme based on Dilworth's well-known decomposition theorem [4] which has been successfully used in scheduling theory [12, 19, 28, 30].

For the special case $\omega = m = 2$, we provide an efficient algorithm that solves the malleable scheduling problem to optimality in Section 3. In the special case of parallel tasks, it can be extended to yield an efficient exact algorithm for $\omega = 2$ and any m.

We complement our positive results by showing in Section 4 that scheduling malleable tasks becomes NP-hard for $\omega \geq 3$ or $m \geq 3$, and parallel tasks for $\omega \geq 3$ *and* $m \geq 2$. Thus, our algorithms are the best one can hope for in terms of approximation guarantee, unless P=NP. Furthermore, we resolve the complexity question for precedence constraints which form trees by showing that malleable and parallel scheduling is NP-hard even on caterpillars, a special case of trees, for any m. This complexity question was left open in [18, 19].

In all cases, our algorithms can be modified to yield contiguous solutions, as we argue in Section 5. Hence, they also solve the corresponding classical and malleable strip packing problems. In the FPTAS for classical strip packing, we require that the width of the strip m is polynomially bounded (see e.g. also [16]). We are not aware of any previous constant factor approximation for classical strip packing under special precedence constraints. The closest previous result is the factor $\Theta(\log n)$ approximation for arbitrary precedence constraints in [1, 13].

Unlike most previous algorithms for malleable scheduling, none of our algorithms requires the monotonous penalty assumption on processing times. On the other hand, our hardness results hold even when assuming this restriction. Our results are summarized in Table 1.

precedence constraints		parallel tasks	malleable tasks	
			$m = 2$	$m > 2$
ω	≤ 2	$\in P$ (Thms. 5 and 6)		FPTAS (Thm. 3)
ω	> 2	FPTAS (Thms. 3 and 4)		
caterpillar		strongly NP-hard (Thm. 10)		

Table 1: Algorithms and complexity results for scheduling parallel and malleable tasks under precedence constraints of bounded width ω, and such forming a caterpillar, i.e., a special tree. The cases for which there is an FPTAS are also shown to be NP-hard, for the polynomial cases efficient exact algorithms are given. Identical results hold for the case when schedules are required to be contiguous, i.e., strip packing, where only for the FPTAS in the case of parallel tasks and $\omega > 2$, we need the common additional assumption that m is polynomial in the input size.

2 A Fully Polynomial-Time Approximation Scheme (FPTAS)

Given a scheduling instance with precedence constraints of width bounded by a constant ω, the number of tasks processed concurrently in a feasible schedule can never exceed ω . Moreover, any maximal set A of incomparable tasks partitions the set of all tasks into two subsets containing tasks that must be processed before, respectively after, some task in A. We exploit this structure to obtain an exact dynamic programming algorithm with pseudo-polynomial running time. Then, we show how to turn this algorithm into an FPTAS.

2.1 Dynamic Programming Algorithm (DP)

The structure of our dynamic program is based on a correlation between feasible subschedules and *ideals* of orders as described in [23]. An ideal I of $(J, \prec)$ is a subset of J such that every task of I implies all its predecessors to be elements of I as well. In order to respect precedence constraints, every initial part of a feasible schedule must consist of a subset of tasks fulfilling the ideal property. We will define our dynamic program in terms of finding paths in a directed graph on ideals; reaching an ideal I' from $I \subset I'$ will correspond to feasibly extending a subschedule by the tasks in $I' \setminus I$.

Utilizing *Dilworth's Decomposition Theorem* [4], which states that for any partial order $(J, \prec)$ of width ω, there exists a partition of $(J, \prec)$ into ω chains $\mathbb{C}_1, \ldots, \mathbb{C}_\omega$, we can represent order ideals as follows. For a given chain decomposition $\mathbb{C}_1, \ldots, \mathbb{C}_\omega$, every ideal I of $(J, \prec)$ can be described by an ω-tuple $(I_i)_{i=1,\ldots,\omega}$, where component I_i indicates that the first I_i tasks of chain $\mathbb{C}_i$ are contained in I. Thus, the number of distinct ideals is bounded by n^ω. Such a chain decomposition can be found in polynomial time, see e.g. Fulkerson [9]. To simplify notation, we identify I_i with the I_i-th task of chain $\mathbb{C}_i$ and denote its processing time as $p_i(\cdot)$.

A *state* in our dynamic program is a triple $[I, \alpha, C]$ specifying an ideal I represented by its *front tasks* $I_1, \ldots, I_\omega$, as well as an allotment vector $\alpha = (\alpha_i)_{i=1,\ldots,\omega}$ and a vector of completion times $C = (C_i)_{i=1,\ldots,\omega}$ for the front tasks of I. This information also defines start times $\sigma_i := C_i - p_i(\alpha_i)$ for all front tasks $(I_i)_{i=1,\ldots,\omega}$. We call a state *valid*, if the number of processors used by its front tasks does not exceed the number of available processors m at any completion time C_i. If $I_i = 0$, no task of chain $\mathbb{C}_i$ is contained in I, and we set α_i and C_i to 0. We call the particular state $\oslash := [\emptyset, 0, 0]$ *start state* and every state $[I, \alpha, C]$ with $I = J$ *end state*.

Every feasible subschedule has a representation as a state defined by the allotment values and the completion times of its front tasks. We establish a state graph G by linking two valid states $F = [I, \alpha, C]$, $F' = [I', \alpha', C']$ by an arc (F, F'), if F' is an extension of F by one task j with feasible α_j and C_j. More formally, the conditions for inserting the arc are:

1. The ideals differ only in one component i, and $I'_i = I_i + 1$. All other components of I', α' and C' remain equal.
2. The start time σ'_i of I'_i in F' respects precedence constraints, i.e., $\sigma'_i \geq C_j$ for all front tasks $(I_j)_{j=1,\dots,\omega}$ with $I_j \prec I'_i$
3. The new task starts no earlier than the other front tasks, i.e., $\sigma'_i \geq \sigma'_j$ for all $j = 1, \dots, \omega$.

Note, that the validity of a state as well as conditions 1–3 can be checked in constant time $\mathscr{O}(\omega^2)$.

Condition 1 ensures that across a path P in G from $\oslash$ to an end state, every task in J is assigned exactly one α_j and σ_j. By conditions 2 and 3 and the ideal property in the states, these start times respect all precedence constraints. Finally, the number of available processors is never exceeded due to condition 3: When a new task j is added to F with start time σ_j and allotment α_j, all tasks in I active at or after σ_j are front tasks, thus they are all taken into account when determining α_j.

We have hence established that any path P in G corresponds to a feasible schedule, and the makespan of such schedule is given by the largest C_i in the path's end state. We will now prove that the converse holds as well.

Lemma 1. *Any feasible schedule S with makespan $C_{\max}$ corresponds to a path in G from $\oslash$ to an end state with latest completion time $C_{\max}$.*

Proof. Our argument is inductive, and we start with an arbitrary feasible schedule S containing all tasks in J. The graph G obviously contains an end state $F' = [I', \alpha', C']$, in which α' and C' respectively correspond to the allotment and completion times of the last tasks in the chains $\mathbb{C}_1, \dots, \mathbb{C}_\omega$ of an appropriate chain decomposition of J. These tasks form the front tasks of F', defining ideal I' which contains all tasks in J. Let j denote a front task of I' with the latest start time. Now $I := I' \setminus \{j\}$ is again an ideal. Thus, there is a valid state $F = [I, \alpha, C]$ in G with α and C corresponding to the allotment values and completion times of the front tasks of I in S. By construction and the feasibility of S the states F and F' fulfill conditions 1–3. Hence, G contains the edge (F, F'). By induction, this yields the desired result. □

With Lemma 1 we can find an optimal schedule as follows: We search for an end state with minimum makespan reachable from the start state, and create the schedule by backtracking.

The number of distinct ideals I is bounded by n^ω as already mentioned, whereas the number of feasible allotments for each ideal does not exceed m^ω. The optimal makespan is at most $Z_{UB} = np_{\max}$ with $p_{\max} := \max\{p_j(m) \mid j \in J\}$. Thus, assuming w.l.o.g. that processing times are integral (by a standard scaling argument), the task completion time can attain up to Z_{UB} different values. Hence, the number of valid states is bounded by $n^\omega m^\omega Z_{UB}^\omega$, which gives a bound of $\mathscr{O}(n^{2\omega} m^{2\omega} Z_{UB}^{2\omega})$ on the overall running time of the dynamic programming algorithm.

Theorem 2. *For any instance of malleable scheduling with precedence constraints of width ω, algorithm DP finds a feasible solution with minimum makespan in time $\mathscr{O}(n^{2\omega} m^{2\omega} Z_{UB}^{2\omega})$.*

2.2 FPTAS

In a fully polynomial time algorithm, we cannot afford to consider all values in $[0, Z_{UB}]$ for possible completion times of front tasks. Using a standard rounding technique, this number can be reduced to be polynomially bounded in the input size and $1/\varepsilon$ at the cost of increasing the makespan by at most a factor $(1+\varepsilon)$ in the following way.

For a given parameter $\varepsilon > 0$, we partition the interval $[0, Z_{UB}]$ into subintervals of size $\varepsilon Z_{LB}/n$ and restrict the possible completion times to the set E of endpoints of the subintervals. This reduces the number of values for possible completion times to $nZ_{UB}/(\varepsilon Z_{LB}) \leq n^2/\varepsilon$, for some lower bound on the optimal value $Z_{LB} \geq p_{\max}$. Now we run algorithm DP$'$, which is a slightly modified variant of algorithm DP in which we round the completion times in a state up to the nearest value in E. This restriction increases an optimal solution value of DP by at most $\varepsilon Z_{LB}/n$ per task. Thus, DP$'$ finds a schedule with a makespan that exceeds the optimal makespan found by DP by at most εZ_{LB}. We refer to [31] for an overview of techniques for obtaining FPTASs.

Theorem 3. *There exists an FPTAS for scheduling malleable tasks under precedence constraints of bounded width ω. For any given $\varepsilon > 0$, a solution with makespan at most $(1+\varepsilon)$ times the optimum can be computed in time $\mathcal{O}((n^3/\varepsilon)^{2\omega} m^{2\omega})$.*

In case a particular allotment is given, the number of states which need to be considered obviously reduces significantly. Here, the running time of DP drops to $\mathcal{O}(n^{2\omega} Z_{UB}^{2\omega})$, and we obtain an FPTAS for scheduling parallel tasks, even if m is not polynomially bounded.

Theorem 4. *There exists an FPTAS for scheduling parallel tasks under precedence constraints of bounded width ω. For any given $\varepsilon > 0$, a solution with makespan at most $(1+\varepsilon)$ times the optimum can be computed in time $\mathcal{O}((n^3/\varepsilon)^{2\omega})$.*

3 Optimally Solvable Special Cases

The problem of scheduling malleable tasks is clearly optimally solvable in polynomial time if either the number of processors is $m = 1$, or the width of the partial order is $\omega = 1$. Thus, we assume $m, \omega \geq 2$ from now on. We provide an efficient algorithm for optimally scheduling malleable tasks in the special case $m = \omega = 2$, and parallel tasks for $\omega = 2$ and any m. We prove NP-hardness of all remaining cases in Section 4.

Our algorithm is based on the idea of the dynamic program in Section 2.1 and the following observations regarding optimal (sub)solutions for special allotments: Consider a subset of tasks $J' \subseteq J$ together with an allotment α'_j for all $j \in J'$. We call (J', α') *nice* if α' and the earliest possible start times

$$\sigma'_j = \begin{cases} 0 & \text{if } j \text{ has no predecessor in } J' \\ \max_{i \prec j, i \in J'} \{\sigma_i + p_i(\alpha'_i)\} & \text{otherwise} \end{cases}$$

result in a feasible (sub)schedule. Calculating σ' and testing for niceness can be done in polynomial time. The obtained makespan clearly coincides with a longest chain in $(J', \prec)$ w.r.t. processing times under $(\alpha'_j)_{j \in J'}$. This is a lower bound on the optimum of the corresponding scheduling instance (J', α') where parallel tasks with the fixed allotment α' must be scheduled. Hence, we have the following.

Observation 1. *If a set of jobs with given allotment $(J' \subseteq J, \alpha')$ is nice, an optimal schedule for (J', α') can be found in polynomial time.*

Since $m \geq 2$, this applies in particular to the case when all tasks are alloted exactly one processor and $\omega = 2$, as precedence constraints alone prevent more than two tasks from being processed concurrently.

Observation 2. *If $\omega = 2$ and $m \geq 2$, any (J', α') with $\alpha'_j = 1$ for all $j \in J'$ is nice.*

Theorem 5. *The problem of scheduling malleable tasks under precedence constraints of width bounded by $\omega = 2$ on $m = 2$ processors can be solved optimally in polynomial time.*

Proof. For any optimal schedule, we refer to the optimal schedule resulting from shifting all start times to the earliest possible point in time (without changing the structure of the schedule) as *normalized*. Any normalized optimal solution can be divided into subsequent subschedules $S_{J'}$ for subsets $J' \subseteq J$ by splitting it whenever all processors become unused simultaneously, i.e., a processor is idle or has just finished processing a task. See Figure 1 for an illustration.

The makespans of these subschedules add up to the makespan of the whole schedule. Moreover, for each subschedule $S_{J'}$, either $\alpha_j = 1$ for all $j \in J'$, or $|J'| = 1$, since all $m = 2$ processors are unused at the start and completion time of any j with $\alpha_j = 2$.

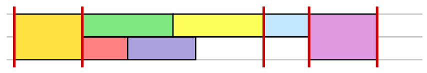
Figure 1: Partitioning a normalized schedule into nice subschedules.

We now define a state graph G whose nodes correspond to the (polynomially many) ideals of $(J, \prec)$ as in Section 2.1, such that any solution of the above structure is represented as a path in G—a shortest path in G will correspond to an optimal schedule. We insert an edge from I to I' in G, if and only if $I \subset I'$. Such an edge corresponds to a subschedule $S_{J'}$ for the tasks in $J' := (I' \setminus I) \subseteq J$ as follows. If J' forms a chain in $(J, \prec)$, in particular, if $|J'| = 1$, we set $\alpha_j = \arg\min\{p_j(\alpha) \mid \alpha \in \{1,2\}\}$ for all $j \in J'$ and define the corresponding optimal subschedule by concatenating the tasks according to their precedence relations. Otherwise, we set $\alpha_j = 1$ for all $j \in J'$ and compute an optimal subschedule for $S_{J'}$ for J', α'_j according to Observation 2. The lengths of the edges are set to the makespans of their subschedules.

Clearly, paths in G from $\emptyset$ to J define feasible schedules of the same length. Furthermore, any normalized optimal schedule is represented as a path in G: For each of its subschedules $S_{J'}$, split as explained above, there are ideals I, I' with $J' = I' \setminus I$ connected by an edge. Moreover, for each of these J', either $\alpha_j = 1$ for all $j \in J'$, or $|J'| = 1$. In both cases, edge lengths correspond to optimal subschedules by the construction of G, hence the path corresponding to an optimal schedule has length equal to its makespan. $\square$

Using a similar idea as in the proof above, we can state a stronger result for the special case of parallel tasks.

Theorem 6. *The problem of scheduling parallel tasks under precedence constraints of width $\omega = 2$ can be solved optimally in polynomial time for any number of processors m.*

Proof. In contrast to the malleable case, we do not have to decide on an allotment for each task, but merely assign a start time to it. Thus, the set of edges of the state graph changes slightly. Now, an edge from an ideal I to ideal I' is inserted in G, if and only if $I \subset I'$, and $J' := I' \setminus I$ with the input allotment is nice. The length of an edge is set to the makespan of the corresponding optimal subschedule, which again can be computed in polynomial time by Observation 1.

Again, each path in G clearly defines a feasible schedule, and it remains to show that every normalized optimal schedule is represented by a path in G of the same length. Consider a subschedule $S_{J'}$ in a partition into subschedules as defined above. Suppose there is a task $j \in J'$ whose start time in $S_{J'}$ is after $\sigma'(j)$. Then, task j must be blocked by some task i using more than $m - \alpha_j$ processors, and i is not a predecessor of j. Hence, i and j are incomparable, and due to $\omega = 2$, there can be no other task incomparable to both i and j. Consequently, no task is running in parallel to i, so all processors are unused when i finishes. This entails that i and j must be in different subschedules, so J' with the input allotment is nice and G contains an edge of correct length corresponding to $S_{J'}$. $\square$

4 Hardness Results

In this section we show that our algorithms in the previous sections are, in a sense, best possible. We show that the problem is NP-hard, even under the monotonous penalty assumption, when $\omega \geq 3, m \geq 2$ (Theorem 7) or $\omega = 2, m \geq 3$ (Theorem 9), where the first result holds for parallel tasks as well. We complement these results by proving NP-hardness for precedence constraints that form a caterpillar, i.e., a special tree. Recall that trees are a special case of series-parallel orders. This complexity status was left open in previous work presenting approximation algorithms for trees and generally series-parallel precedence constraints for malleable tasks [18, 19].

Theorem 7. *The problem of scheduling malleable tasks with precedence constraints of width bounded by a constant $\omega \geq 3$ on any fixed number of processors $m \geq 2$ is NP-hard, even under the monotonous penalty assumption.*

Proof. We give a reduction from the NP-complete PARTITION problem, see [11], to malleable scheduling problem with precedence constraints forming three independent chains. This specific problem variant can be reduced to any other problem with $\omega \geq 3$ and $m \geq 2$ by adapting the number of processors needed by tasks to achieve the processing times used in the reduction.

Consider an instance P of PARTITION: Given a set of *values* $\{v_i\}_{i=1,\dots,n}$ with $V := \sum_{i=1}^{n} v_i$, does there exist a partition A_1, A_2 of $\{1,\dots,n\}$ such that $\sum_{i\in A_1} v_i = \sum_{i\in A_2} v_i = V/2$?

We construct an instance S of malleable scheduling, borrowing ideas from a reduction for scheduling with communication delays in [30]. Instance S consists of $3n$ tasks to be scheduled on $m = 2$ processors. The precedence relations form three chains $\{a_i\}_{i=1,\dots,n}$, $\{b_i\}_{i=1,\dots,n}$, and $\{c_i\}_{i=1,\dots,n}$ of n tasks each. Suppose that the tasks of each chain are ordered with respect to their indices, i.e., $a_i \prec a_j$ for all $i < j$. Each node in chains $\{a_i\}_{i=1,\dots,n}$ and $\{b_i\}_{i=1,\dots,n}$ has processing time V for any processor allotment. Each task i in chain $\{c_i\}_{i=1,\dots,n}$ corresponds to element i of instance P and has processing time v_i independently of the processor allotment. Note that all processing times obey the monotonous penalty assumption. The construction is illustrated in Figure 2.

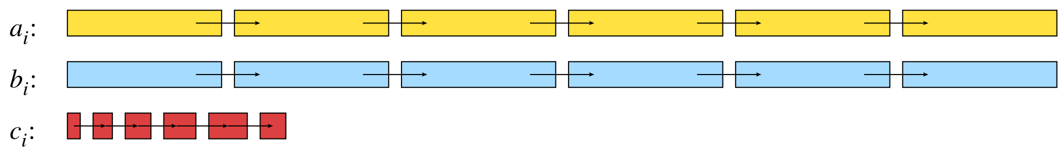

Figure 2: The three chains of jobs in an instance of malleable scheduling constructed from an instance of PARTITION; all tasks' processing times are oblivious of the allotment.

We prove that there is a feasible schedule for S with makespan at most $nV + V/2$ if and only if P is a yes-instance. Let A_1, A_2 be a partition satisfying $\sum_{i\in A_1} v_i = \sum_{i\in A_2} v_i = V/2$, and let $\pi(i)$ denote the index of the subset containing i. Consider the schedule, in which all tasks a_i are processed on the first processor, all tasks b_i on the second processor, and every task c_i on processor $\pi(i)$ placed between task a_{i-1} and a_i respectively b_{i-1} and b_i. More formally, we define the following start times for all tasks:

$$\sigma_{a_i} := \sum_{k \in A_1, k \leq i} v_k + (i-1)V,$$
$$\sigma_{b_i} := \sum_{k \in A_2, k \leq i} v_k + (i-1)V,$$
$$\sigma_{c_i} := \sum_{k \in A_{\pi(i)}, k < i} v_k + (i-1)V.$$

Simple calculations show that no precedence relation is violated and that the number of used processors never exceeds two. Thus, there exists a feasible schedule with makespan $nV + V/2$ as illustrated in Figure 3.

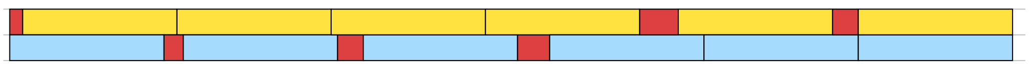

Figure 3: A feasible schedule corresponding to a certificate for a yes-instance of PARTITION. The work from the chain of short tasks is distributed equally between the two processors. Long tasks are just long enough to ensure short tasks can respect precedence constraints without gaps in the schedule.

Now assume there is a schedule for S with makespan at most $nV + V/2$. Clearly, each processor must handle exactly n tasks with processing time V; these have to be the tasks of chains $\{a_i\}_{i=1,\dots,n}$ and $\{b_i\}_{i=1,\dots,n}$. Thus, on every processor there are exactly $V/2$ time units left in the makespan for processing the remaining tasks of chain $\{c_i\}_{i=1,\dots,n}$. Consequently, there must exist a partition for instance P. $\square$

The reduction in the proof of Theorem 7 adapts naturally to the problem of scheduling parallel tasks. Here, we simply fix the allotment for all tasks to one processor.

Corollary 8. *The problem of scheduling parallel tasks under precedence constraints of width bounded by a constant $\omega \geq 3$ on any fixed number of processors $m \geq 2$ is NP-hard.*

Theorem 9. *The problem of scheduling malleable tasks with precedence constraints of width $\omega = 2$ is NP-hard on any fixed number of processors $m \geq 3$, even under the monotonous penalty assumption.*

Proof. We give a reduction from an arbitrary instance of the NP-complete KNAPSACK decision problem, see [11], to the problem of scheduling two independent chains of malleable tasks on $m = 3$ processors. It is easy to see that this case can be reduced to any problem setting with $m > 3$ by adapting the number of processors needed by tasks to achieve the processing times used in the reduction.

Let K denote an instance of KNAPSACK in a slightly modified formulation: Given a set of *values* $\{v_i\}_{i=1,\dots,n}$, a set of *weights* $\{w_i\}_{i=1,\dots,n}$ and numbers V and W, does there exist a set $A \subseteq \{1,\dots,n\}$ with total weight at most W, i.e., $\sum_{i \in A} w_i \leq W$ and a complement valued at most V, i.e., $\sum_{i \notin A} v_i \leq V$? By a standard scaling argument we may assume w.l.o.g. that $V = W$.

We construct a corresponding instance S of our scheduling problem as follows: For each item $i = 1,\dots,n$, we introduce tasks j_i and $\bar{j}_i$. All j_i, and all $\bar{j}_i$ respectively, form a chain in the order of their indices. In each chain, between any two consecutive tasks, there is an additional task h_i, respectively $\bar{h}_i$, which has processing time $p_h := n \max_i\{v_i, w_i\}$ for any allotment of processors. All tasks j_i, $\bar{j}_i$ have

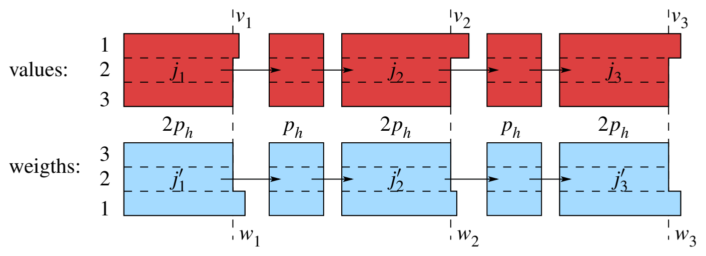

Figure 4: The two chains of malleable tasks constructed from a KNAPSACK instance with processing times on one, two, and three processors. Every other element in each chain is oblivious of the allotment, while for the remaining tasks, allotting only one processor increases their processing time according to their knapsack value in one chain, and according to their knapsack weight in the other.

the same processing time $p_b := 2p_h$ on two and three processors; when processed by a single processor, the processing time of task j_i increases by v_i, and the processing time of task $\bar{j}_i$ increases by w_i, respectively; see Figure 4 for an illustration. These processing times clearly obey the monotonous penalty assumption.

We prove that the KNAPSACK instance K has a solution (is a yes-instance) if and only if there is a schedule for S with makespan at most $(n-1)p_h + np_b + V$.

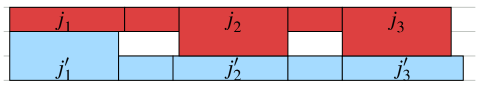

Figure 5: A feasible schedule corresponding to a certificate for a yes-instance of KNAPSACK. Every other task in each chain serves as a separator, guaranteeing the desired structure of the schedule. For the remaining tasks, out of each pair of parallel elements from the two chains, one has to be processed on one processor only. This entails an increase in processing time for each pair, corresponding to either the weight, or the value of an item. An increase by weight corresponds to packing, by value to leaving; the schedule depicted corresponds to packing items two and three.

Given a feasible set A for K we can construct a feasible schedule for S as follows: For every item $i \in A$ we allot two processors to task j_i and one processor to task $\bar{j}_i$; for every item $i \notin A$, we proceed vice versa. The remaining tasks run on one processor. We schedule every task directly after the completion of its predecessor, i.e.,

$$\sigma_{j_i} := (i-1)p_h + \sum_{k<i} p_{j_k}(\alpha_{j_k}),$$
$$\sigma_{h_i} := (i-1)p_h + \sum_{k\leq i} p_{j_k}(\alpha_{j_k}).$$

Start times $\sigma_{\bar{j}_i}$ and $\sigma_{\bar{h}_i}$ are defined the same way. In this schedule, the processing of task j_{i-1} is completed before the processing of task $\bar{j}_i$ starts, i.e.,

$$\begin{aligned}
&\sigma_{j_{i-1}} + p_{j_{i-1}}(\alpha_{j_{i-1}}) \\
=&(i-2)p_h + \sum_{k\leq i-1} p_{j_k}(\alpha_{j_k}) \\
=&(i-2)p_h + \sum_{k\leq i-1} p_b + \sum_{k\leq i-1, k\notin A} v_k \\
\leq&(i-1)p_h + \sum_{k<i} p_b \leq \sigma_{\bar{j}_i}.
\end{aligned}$$

This holds analogously for $\bar{j}_{i-1}$ and j_i. Thus, there are never more than three processors in use and the schedule is feasible. Moreover, its makespan is

$$(n-1)p_h + np_b + \max\{\sum_{i\notin A} v_i, \sum_{i\in A} w_i\},$$

which is at most $(n-1)p_h + np_b + V$ as depicted in Figure 5.

Conversely, for a given schedule for S with makespan at most $(n-1)p_h + np_b + V$ we can construct a feasible set A for the KNAPSACK instance K. Suppose there are two tasks j_i and $\bar{j}_i$ with disjoint processing intervals; w.l.o.g. let $\bar{j}_i$ be processed before j_i. Then all predecessors of $\bar{j}_i$ must be processed before $\bar{j}_i$ and all successors of j_i must be processed after task j_i. Thus, the makespan of the schedule is at least

$$np_b + (n-1)p_h + p_b > (n-1)p_h + np_b + V.$$

Thus, for any two tasks j_i or $\bar{j}_i$ at least one of them must be processed on a single processor.

Let A contain all items i that correspond to tasks $\bar{j}_i$ using a single processor. It is easy to verify that this set is a feasible solution to K. Given the makespan of the schedule, we have that

$$(n-1)p_h + np_b + \sum_{i\in A} w_i \;\leq\; (n-1)p_h + np_b + V,$$

which implies $\sum_{i\in A} w_i \leq V$. On the other hand,

$$(n-1)p_h + np_b + \sum_{i\notin A} v_i \;\leq\; (n-1)p_h + np_b + V,$$

which results in $\sum_{i\notin A} v_i \leq V$. □

Note that the use of malleable tasks in the reduction above is imperative—thus, the proof does not carry over to the case of parallel tasks. This fits in with Corollary 6 where we have shown this case to be tractable for $\omega = 2$ and any m.

We conclude our complexity investigations by showing that the scheduling problem is also NP-hard under precedence constraints that form a caterpillar, i.e., a special tree, composed of a path and leaves only. This result still holds for parallel tasks or when assuming monotonous penalties.

Theorem 10. *The problem of scheduling malleable tasks with precedence constraints that form a caterpillar is strongly NP-hard for every fixed $m \geq 2$, even under the monotonous penalty assumption.*

Proof. It suffices to consider the case $m = 2$, since again for any greater number of processors, we can adapt the number of processors needed by tasks to achieve the processing times used below. We give a reduction from an arbitrary instance P of the strongly NP-complete 3-PARTITION decision problem, see [11]. Such instance consists of natural numbers z, B and a_i with $\sum_{i=1}^{3z} a_i = zB$ and $B/4 < a_i < B/3$ for all $i = 1, \ldots, 3z$, and the question is whether there exists a partition of $\{1, \ldots, 3z\}$ into z disjoint sets $A_1, \ldots, A_z$ such that $\sum_{i \in A_j} a_i = B$ for all $j = 1, \ldots, z$.

We construct the following instance S of malleable scheduling on two processors: The set of tasks J contains two tasks j_i, k_i for all $i = 1, \ldots, 3z$ and two tasks g_i, h_i for all $i = 1, \ldots, z$ with the following processing times with monotonous penalties:

$$\begin{aligned}
p_{j_i}(1) &= a_i, & p_{j_i}(2) &= a_i;\\
p_{k_i}(1) &= 2B, & p_{k_i}(2) &= B;\\
p_{g_i}(1) &= B, & p_{g_i}(2) &= B;\\
p_{h_i}(1) &= 2B, & p_{h_i}(2) &= B.
\end{aligned}$$

We introduce the following precedence constraints, which obviously form a caterpillar, see also Figure 6:

$$k_1 \prec \cdots \prec k_{3z} \prec g_1 \prec h_1 \prec \cdots \prec g_z \prec h_z \tag{1}$$

$$k_1 \prec j_1, \ldots, k_{3z} \prec j_{3z} \tag{2}$$

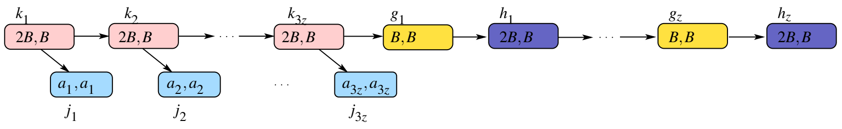

Figure 6: The instance of malleable scheduling constructed from an instance of 3-PARTITION; the precedence constraints clearly form a caterpillar. All tasks j_i and g_i are oblivious of their allotment, while the processing time of all k_i and h_i is halved when using two processors.

We now argue that P is a yes-instance if and only if S has a solution with makespan at most $5zB$. Suppose there exists a partition of the a_i as required. To obtain a solution to S, we can first process all tasks k_i on two processors each. Then, we alternately schedule one task g_i on one processor, and one task h_i on two processors. In parallel to each g_i we schedule three tasks j_i whose processing times belong to the same subset A_x in the partition. This results in a schedule with makespan

$$\begin{aligned}
&\sum_{i=1}^{3z} p_{k_i}(2) + \sum_{i=1}^{z} p_{h_i}(2) + \sum_{i=1}^{z} \max\{g_i, \sum_{k \in A_i} a_k\}\\
=&3zB + zB + zB = 5zB,
\end{aligned}$$

as illustrated in Figure 7.

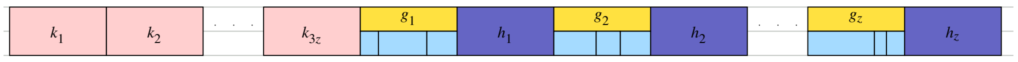

Figure 7: A feasible schedule corresponding to a certificate for a yes-instance of 3-PARTITION. The chain formed jointly by tasks k_i, g_i, and h_i ensures that the schedule can have no gaps if and only if tasks j_i can be partitioned into triplets with equal processing time.

Conversely, assume there exists a solution to S with makespan at most $5zB$. Tasks k_i, g_i, and h_i jointly form a chain whose processing time must not exceed the makespan. Consequently, all tasks k_i

and h_i must run on two processors, and all tasks of this chain must be scheduled one after the other with no gaps. So during the makespan of $5zB$, both processors are busy for time $4zB$, and only one processor remains available for time zB, namely in parallel to processing tasks g_i. Now the remaining tasks j_i and g_i must fit into the schedule without any gaps. This is only possible if the set of tasks j_i can be partitioned into B triplets with processing time z each, which by construction is equivalent to P being a yes-instance. □

This reduction can be adapted to suit the case of parallel tasks: We simply fix the allotment of all tasks k_i and h_i to two processors, and that of all tasks g_i and j_i to one.

Corollary 11. *The problem of scheduling parallel tasks with precedence constraints that form a tree is NP-hard for every fixed $m \geq 2$.*

5 Scheduling on Contiguous Processors and Strip Packing

When we require each task to run on contiguous processors, an allotment $(\alpha_j)_{j\in J}$ can be interpreted as associating a rectangle of width α_j and height $p_j(\alpha_j)$ with each task $j \in J$. Optimally scheduling the tasks in J under this allotment amounts to packing these rectangles into a strip of width m of minimum length (or height). Hence, malleable scheduling on contiguous processors corresponds to strip packing under discrete malleability. In the case of parallel tasks, contiguous scheduling is equivalent to strip packing with strip width m and rectangle widths in $\{1,\dots,m\}$, and any strip packing instance with rational data can be stated this way.

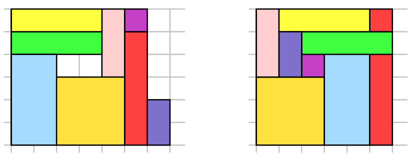

Figure 8: A contiguous schedule with optimal makespan seven on the left-hand side vs. a non-contiguous schedule with optimal makespan six on the right-hand side.

Figure 8 depicts a contiguous schedule on the left, and a schedule with non-contiguous assignment of tasks to processors on the right, see also [8]. It is not too hard to check that no feasible contiguous processor assignment for the latter schedule is possible. In fact, scheduling the given parallel tasks contiguously in our example requires a schedule with makespan at least seven, whereas the minimum makespan of a non-contiguous schedule is six. This shows that the condition of contiguousness is a proper restriction, see also [29]. Moreover, the general problem of deciding whether a given feasible schedule is contiguous, i.e., whether it possesses a feasible contiguous mapping of tasks to processors, is NP-hard in the strong sense. This follows from results in [7] and was independently shown in [13].

We now argue that all algorithms and reductions in this paper can be adapted to yield contiguous processor assignments by construction. Thus, our results also hold for the corresponding strip packing problems.

First, in order for the dynamic program from Section 2.1 to yield contiguous schedules, we merely need to keep track of the distinct processors used by every task in each state. This can be done by extending the definition of a state by $(\pi_i)_{i=1,\dots,\omega}$ to $[I,\alpha,C,\pi]$, where π_i defines the first processor that should be used by the front task I_i. Thus, the contiguous set of processors $[\pi_i, \pi_i + \alpha_i - 1]$ will be assigned to I_i.

Moreover, we have to adapt the definition of a valid state regarding processor usage: Instead of checking that the number of processors used by front tasks does not exceed m, we now require that each front task I_i is the only front task using *its assigned* processors at its completion time C_i. This condition is stronger than the original, but can still be checked in $\mathcal{O}(\omega^2)$. Finally, in the construction of the state graph G, we have to extend Condition 1 for inserting an arc between states F, F': We now require additionally, that also the vector π remains the same in all but the component corresponding to the chain from which a task is added to the schedule when moving from F to F'.

With this construction, an analog of the argumentation in Section 2.1 remains valid for the case of contiguous schedules. However, the number of nodes in the graph increases by a factor of m^{ω}. Hence, the FPTAS deduced remains valid only with a running time increased by a factor of $m^{2\omega}$. In the case of parallel tasks, the number of nodes in the graph reduces again by a factor of m^{ω}, since each front task no longer has ω valid allotment values α_i, but only one. However, the distinct processor assignments π_i still need to be regarded, and m remains a factor in running time. Consequently, for classical strip packing, our FPTAS yields polynomial running time only when the strip width m is polynomial in the input size. In any case, this assumption is quite common, see [16].

Corollary 12. *There exists an FPTAS for finding an optimal contiguous schedule of malleable tasks under precedence constraints of bounded width. For any given $\varepsilon > 0$, a solution with makespan at most $(1+\varepsilon)$ times the optimum can be computed in time $\mathcal{O}((n^3/\varepsilon)^{2\omega}m^{4\omega})$. Furthermore, there is an FPTAS for classical strip packing with integral rectangle widths and polynomially bounded m under precedence constraints of bounded width. Here, the running time of a $(1+\varepsilon)$-approximation for any $\varepsilon > 0$ reduces to $\mathcal{O}((n^3/\varepsilon)^{2\omega}m^{2\omega})$.*

Next, observe that for $m \leq 2$, any schedule is contiguous. Consequently, Theorem 5 can be formulated for the contiguous case. Also, Theorem 6 remains valid, since for $\omega = 2$, at most two parallel tasks can be processed concurrently.

Corollary 13. *Optimal contiguous schedules on m processors under precedence constraints of width bounded by a constant ω can be found in polynomial time for malleable tasks with $\omega = m = 2$, and for parallel tasks with $\omega = 2$ and arbitrary m.*

Furthermore, the scheduling instances constructed in the reductions for Theorems 7 and 10 only use $m = 2$ processors. Also, the scheduling instances arising from the reduction for Theorem 9 clearly permit a contiguous schedule with the required makespan if and only if they permit any such schedule. Consequently, all of our hardness results carry over to the contiguous case.

Corollary 14. *Even when assuming monotonous penalties, contiguous scheduling of malleable and parallel tasks is NP-hard under precedence constraints which form a caterpillar on $m \geq 2$ processors, and under precedence constraints of width bounded by a constant $\omega \geq 3$. For malleable tasks, it remains NP-hard for $\omega = 2$ when $m \geq 3$.*

We close our investigations with an interesting open question regarding the difference between scheduling and packing. Figure 8 demonstrates that requiring contiguous processors may increase the optimal makespan of a set of parallel tasks by as much as $7/6 \approx 1.167$. It seems intriguing to determine the worst case ratio between the makespans of optimal contiguous and non-contiguous schedules for a given instance. For independent tasks, it can be bounded by two due to approximation algorithms for the classical strip packing problem with this factor, analyzed using lower bounds that also hold for non-contiguous scheduling, see [26,27]. Yet we conjecture the actual bound to be much lower.

Acknowledgements We thank Rolf H. Möhring and Martin Skutella for fruitful discussions.

References

[1] J. Augustine, S. Banerjee, and S. Irani. Strip packing with precedence constraints and strip packing with release times. *Theoretical Computer Science*, 410(38–40):3792–3803, 2009.

[2] K. P. Belkhale and P. Banerjee. An approximate algorithm for the partitionable independent task scheduling problem. In *Proceedings of ICPP, Vol. 1*, pages 72–75. Pennsylvania State University Press, 1990.

[3] J. Błażewicz and Z. Liu. Scheduling multiprocessor tasks with chain constraints. *European Journal of Operational Research*, 94(2):231–241, 1996.

[4] R. P. Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, 51(1):161–166, 1950.

[5] M. Drozdowski. Scheduling multiprocessor tasks: an overview. *European Journal of Operational Research*, 94(2):215–230, 1996.

[6] J. Du and J. Y.-T. Leung. Complexity of scheduling parallel task systems. *SIAM Journal on Discrete Mathematics*, 2(4):473–487, 1989.

[7] C. W. Duin and E. V. Sluis. On the complexity of adjacent resource scheduling. *Journal of Scheduling*, 9(1):49–62, 2006.

[8] S. P. Fekete, E. Köhler, and J. Teich. Higher-dimensional packing with order constraints. *SIAM Journal on Discrete Mathematics*, 20(4):1056–1078, 2006.

[9] D. R. Fulkerson. Note on Dilworth's decomposition theorem for partially ordered sets. *Proceedings of the American Mathematical Society*, 7(4):701–702, 1956.

[10] M. R. Garey and R. L. Graham. Bounds for Multiprocessor Scheduling with Resource Constraints. *SIAM Journal on Computing*, 4(2):187–200, 1975.

[11] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.

[12] A. Grigoriev and G. J. Woeginger. Project scheduling with irregular costs: complexity, approximability, and algorithms. *Acta Informatica*, 41(2–3):83–97, 2004.

[13] E. Günther. Bin Scheduling: Partitionieren verformbarer Jobs mit Nebenbedingungen (in German). Master's thesis, Technische Universität Berlin, 2008.

[14] K. Jansen. Scheduling malleable parallel tasks: an asymptotic fully polynomial time approximation scheme. *Algorithmica*, 39(1):59–81, 2004.

[15] K. Jansen and L. Porkolab. Linear-Time Approximation Schemes for Scheduling Malleable Parallel Tasks. *Algorithmica*, 32(3):507–520, 2002.

[16] K. Jansen and R. Thöle. Approximation algorithms for scheduling parallel jobs: Breaking the approximation ratio of 2. In *Proceedings of ICALP*, pages 234–245. Springer Berlin / Heidelberg, 2008.

[17] K. Jansen and H. Zhang. An approximation algorithm for scheduling malleable tasks under general precedence constraints. *ACM Transactions on Algorithms*, 2(3):416–434, 2006.

[18] R. Lepère, G. Mounié, and D. Trystram. An approximation algorithm for scheduling trees of malleable tasks. *European Journal of Operational Research*, 142(2):242–249, 2002.

[19] R. Lepère, D. Trystram, and G. J. Woeginger. Approximation algorithms for scheduling malleable tasks under precedence constraints. *International Journal of Foundations of Computer Science*, 13(4):613–627, 2002.

[20] J.-T. Leung, editor. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. Chapman and Hall/CRC, 2004.

[21] A. Lodi, S. Martello, and M. Monaci. Two-dimensional packing problems: A survey. *European Journal of Operational Research*, 141(2):241–252, 2002.

[22] W. Ludwig and P. Tiwari. Scheduling malleable and nonmalleable parallel tasks. In *Proceedings of SODA*, pages 167–176. Society for Industrial and Applied Mathematics Philadelphia, PA, USA, 1994.

[23] R. H. Möhring. Computationally tractable classes of ordered sets. In I. Rival, editor, *Algorithms and Order*, pages 105–194. Kluwer Academic Publishers, 1989.

[24] G. Mounie, C. Rapine, and D. Trystram. Efficient approximation algorithms for scheduling malleable tasks. In *Proceedings of SPAA*, pages 23–32. ACM New York, NY, USA, 1999.

[25] G. Mounie, C. Rapine, and D. Trystram. A 3/2-Dual Approximation Algorithm for Scheduling Independent Monotonic Malleable Tasks. *SIAM Journal on Computing*, 37(2):401–412, 2008.

[26] I. Schiermeyer. Reverse-fit: A 2-optimal algorithm for packing rectangles. In *Proceedings of ESA*, pages 290–299, London, UK, 1994. Springer.

[27] A. Steinberg. A Strip-Packing Algorithm with Absolute Performance Bound 2. *SIAM Journal on Computing*, 26:401, 1997.

[28] G. Steiner. On the complexity of dynamic programming for sequencing problems with precedence constraints. *Annals of Operations Research*, 26:103–123, 1990.

[29] J. Turek, J. Wolf, and P. Yu. Approximate algorithms for scheduling parallelizable tasks. In *Proceedings of SPAA*, pages 323–332, New York, USA, 1992. ACM.

[30] J. Verriet. The complexity of scheduling graphs of bounded width subject to non-zero communication delays. Technical Report UU-CS-1997-01, Utrecht University, 1997.

[31] G. J. Woeginger. When does a dynamic programming formulation guarantee the existence of a fully polynomial time approximation scheme (FPTAS)? *INFORMS Journal on Computing*, 12(1):57–74, 2000.