Scheduling and Packing Malleable Tasks with Precedence Constraints of Bounded Width

Elisabeth Günther[1], Felix G. König[1], and Nicole Megow[2]

[1] Technische Universität Berlin, Institut für Mathematik, Germany,
{eguenth,fkoenig}@math.tu-berlin.de
[2] Max-Planck-Institut für Informatik, Saarbrücken, Germany,
nmegow@mpi-inf.mpg.de

Abstract. We study two related problems in non-preemptive scheduling and packing of malleable tasks with precedence constraints to minimize the makespan. We distinguish the scheduling variant, in which we allow the free choice of processors, and the packing variant, in which a task must be assigned to a contiguous subset of processors.
For precedence constraints of bounded width, we completely resolve the complexity status for any particular problem setting concerning width bound and number of processors, and give polynomial-time algorithms with best possible performance. For both, scheduling and packing malleable tasks, we present an FPTAS for the NP-hard problem variants and exact algorithms for all remaining special cases. To obtain the positive results, we do not require the common monotonous penalty assumption on processing times, whereas our hardness results hold even when assuming this restriction.
With the close relation between contiguous scheduling and strip packing, our FPTAS is the first (and best possible) constant factor approximation for (malleable) strip packing under special precedence constraints.

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 [4, 13]. Several models have been proposed, among which scheduling malleable tasks as proposed in [15] is an important and promising model [10].

In the problem of scheduling malleable tasks, we are given a set $J = \{1, 2, \ldots, n\}$ 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 α_j 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.

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.

Quite some research has been dedicated to malleable task scheduling since its introduction in [15]. For the problem with general precedence constraints among tasks, Lepère et al. [12] provide an approximation algorithm with performance guarantee $3+\sqrt{5} \approx 5.236$. For special cases, such as series-parallel precedence constraints and precedence constraints 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 by [11]. Jansen and Zhang [10] consider the case of general precedence constraints and provide an algorithm with performance guarantee ≈ 4.730598, which they show to be asymptotically tight.

All these results crucially require 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 (simply set the task duration for forbidden allotments to some large constant). Notice that scheduling *parallel tasks*, for which the number of allotted processors is already given, is a special case in our model.

A remarkable amount of literature deals with scheduling independent parallel tasks. The only work concerning precedence constraints, that we are aware of in this setting, investigates the special case of chains [2]. 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.

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 [15]. Duin and van der Sluis [5] investigate contiguous scheduling of parallel tasks in the context of assigning check-in counters at airports; they call it *adjacent scheduling*. Another problem closely related to contiguous scheduling is *strip packing*, i.e., the problem of packing rectangles in a strip of width 1 such that the packing height is minimized. 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.

The classical strip packing problem with arbitrary precedence constraints has been investigated by Augustine et al. [1]; 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 scheduling parallel tasks as well, the approximation guarantee carries over. Moreover, with the techniques in [12] or [10], the result can be transferred to malleable task scheduling with arbitrary precedence constraints if the monotonous penalty assumption holds, see [8]. Augustine et al. [1] also provide 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.

Our results. We derive a fully polynomial-time approximation scheme (FPTAS) for scheduling malleable tasks under precedence constraints of bounded width, ω. This significantly improves on the previously best known approximation ratio of $(3+\sqrt{5})/2 \approx 2.618$ in [12] under the restriction to monotonous penalty functions. To the best of our knowledge, our FPTAS also constitutes the first constant factor approximation for scheduling parallel tasks with precedence constraints.

For the special case $\omega = m = 2$, we provide an efficient algorithm that solves the problem to optimality. We complement our positive results by showing that the problem becomes NP-hard for $\omega \geq 3$ or $m \geq 3$. Thus, our algorithms are best possible, unless P=NP.

When scheduling parallel tasks, our positive results can be extended even further, yielding an efficient exact algorithm for $\omega \geq 2$ and any m. All other cases are shown to be NP-hard. Furthermore, we resolve the complexity question for precedence constraints which form caterpillars, a special case of trees, by showing that these problems are also NP-hard for malleable tasks and even parallel tasks, for any m.

Regarding contiguous scheduling, or discrete malleable strip packing, all of our hardness results carry over. Also, the FPTAS can be adapted naturally. For the special case of $\omega = 2$, our algorithm efficiently computes optimal solutions for classical (non-malleable) strip packing. Similarly, we efficiently solve discrete malleable strip packing for $\omega = m = 2$ to optimality. Under the assumption that the width of the strip m is polynomially bounded (see e.g. also [9]), our FPTAS is the first constant factor approximation for classical strip packing under special precedence constraints. The best previous result is a general factor $\Theta(\log n)$ approximation for arbitrary precedence constraints in [1, 8].

Quite notably, 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.

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 . On the other hand, 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 [14]. 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. 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 based 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* [3] 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 [6]. 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 $[I, \alpha, C]$ which specifies 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 $(I_j)_{j=1,\ldots,\omega}$ 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 $j = 1, \ldots, \omega$.

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

Condition 1 clearly 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 of 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. Furthermore, the makespan of such schedule is given by the largest C_i in its end state.

We have hence established, that any path P in G corresponds to a feasible schedule with makespan determined by the end state of P. 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 $\mathbb{C}_1, \ldots, \mathbb{C}_\omega$ of an appropriate chain decomposition of J. These tasks form the front tasks of F', defining ideal I' containing 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 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 was already mentioned to be bounded by n^ω, whereas the number of feasible allotments for each ideal does not exceed m^ω. The optimal makespan is at most by $Z_{UB} = np_{\max}$ with $p_{\max} := \max\{p_j(m) \mid j \in J\}$. Thus, assuming w.l.o.g. that processing times are integral (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 on the overall running time of the dynamic programming algorithm, $\mathcal{O}(n^{2\omega} m^{2\omega} Z_{UB}^{2\omega})$.

Theorem 1. *For a given scheduling instance with precedence constraints of width ω, algorithm DP finds a feasible solution with minimum makespan in time $\mathcal{O}(n^{2\omega} m^{2\omega} Z_{UB}^{2\omega})$.*

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 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} time units. We skip further details and refer to [17] for an overview of techniques for obtaining FPTASs.

Theorem 2. *There exists an FPTAS for scheduling malleable tasks with precedence constraints of bounded width with running time.*

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 solving the special case $m = \omega = 2$ to optimality. It is based on the the idea of the dynamic program in Sec. 2.1 and the following observations regarding optimal (sub)solutions for special allotments. We prove NP-hardness of all remaining cases of m and ω in Sec. 4.

Observation 1 *Given a subset of tasks $J' \subseteq J$ with an allotment $\alpha_j = 2$ for all $j \in J'$, an optimal schedule for J' can be found in polynomial time if $m = \omega = 2$.*

Observation 2 *Given a subset of tasks $J' \subseteq J$ with an allotment $\alpha_j = 1$ for all $j \in J'$, an optimal schedule for J' can be found in polynomial time if $m = \omega = 2$.*

While the former is obvious, the latter can be realized by list scheduling tasks j in topological order, and setting $\sigma_j = \max_{i \prec j}\{\sigma_i + p_i(1)\}$, or $\sigma_j = 0$ if no such i exists. The makespan obtained clearly coincides with a longest chain in $(J, \prec)$ w.r.t. processing times under $(\alpha_j)_{j \in J}$, a lower bound on the optimum. The resulting schedule is also feasible since $\omega = 2$.

Theorem 3. *The problem of scheduling malleable tasks with precedence constraints of width bounded by $\omega = 2$ on $m = 2$ processors can be solved optimally in polynomial time.*

Proof. For $m = 2$, any optimal solution can be split into maximal subsequent subschedules $S_{J'}$ for subsets $J' \subseteq J$, such that in any $S_{J'}$, either $\alpha_j = 1$ for all $j \in J'$, or $\alpha_j = 2$ for all $j \in J'$. We say these subschedules are of *type one* or *type two*, respectively. Their makespans simply add up to the makespan of the whole schedule.

We now define a graph G whose nodes correspond to the (polynomially many) ideals of $(J, \prec)$ as in Sec. 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. There is 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 schedule by concatenating optimal schedules of type one and two for the cases $\alpha_j = 1$ and $\alpha_j = 2$, respectively. Otherwise, we define $S_{J'}$ to be an optimal subschedule of type two. Note that by Obs. 1 and 2, all of these $S_{J'}$ can be computed in polynomial time. The lengths of edges are set to the makespans of these optimal subschedules.

Clearly, paths in G from $\emptyset$ to J correspond to feasible schedules of the same length. Furthermore, any optimal schedule is represented as a path in G, since for each of its subsolutions $S_{J'}$ of type one, there are ideals I, I' with $J' = I' \setminus I$ connected by an edge, and for each $S_{J'}$ of type two, there is a path from I to I' across ideals only differing by one task and $J' = I' \setminus I$. □

Using the same idea as in the proof above, we can state an even stronger positive result for the special case of parallel tasks.

Corollary 1. *The problem of scheduling parallel tasks with precedence constraints of width $\omega = 2$ can be solved optimally in polynomial time for any number of processors m.*

The result ensues when redefining the notions of subschedules of type one and type two in the proof of Thm. 3. We adapt the correspondence between edges in the ideal graph and subschedules accordingly: Define maximal subschedules containing no concurrent tasks to be of type one, and those in which the processing interval of any tasks overlaps with that of another to be of type two. Now note that an analog of Obs. 2 remains valid.

4 Hardness results

In this section we show that the results in the previous section are the best that we can hope for, unless P=NP. We fully settle the complexity status of scheduling malleable tasks under precedence constraints of bounded width ω by showing that the problem is NP-hard even under the monotonous penalty assumption when $\omega \geq 3, m \geq 2$ (Thm. 4) or $\omega = 2, m \geq 3$ (Thm. 5), where the former result holds for parallel tasks as well. We complement these results with proving NP-hardness for precedence constraints that form a caterpillar, i.e., a special tree.

Theorem 4. *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 [7], to the scheduling problem with precedence constraints that form 3 independent chains. It is easy to see that 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: Does there exist a partition A_1, A_2 of $\{1, \ldots, n\}$ for a given set of *values* $\{v_i\}_{i=1,\ldots,n}$ with $V := \sum_{i=1}^{n} v_i$, that fulfills the condition $\sum_{i \in A_1} v_i = \sum_{i \in A_2} v_i = V/2$?

We construct an instance S of our scheduling problem by borrowing ideas from a reduction for scheduling with communication delays in [16]. Instance S consists of $3n$ tasks to be scheduled on $m = 2$ processors. The precedence relations form 3 chains $\{a_i\}_{i=1,\ldots,n}$, $\{b_i\}_{i=1,\ldots,n}$, and $\{c_i\}_{i=1,\ldots,n}$ 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 $\{a_i\}_{i=1,\ldots,n}$ and $\{b_i\}_{i=1,\ldots,n}$ has processing time V for any processor allotment. Each task i in chain $\{c_i\}_{i=1,\ldots,n}$ 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.

We prove, that there is a feasible schedule for instance 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 start times for all tasks $\sigma_{a_i} := \sum_{k\in A_1, k\le i} v_k + (i-1)V$, $\sigma_{b_i} := \sum_{k\in A_2, k\le i} v_k + (i-1)V$, and $\sigma_{c_i} := \sum_{k\in A_{\pi(i)}, k<i} v_k + (i-1)V$. Simple calculations show that no precedence relation is violated and that the number of used processors never exceeds 2. Thus, there exists a feasible schedule with makespan $nV + V/2$.

Consider now a schedule for instance S with makespan at most $nV + V/2$. Clearly, on each processor there are exactly n tasks with processing time V; these are the tasks of chains $\{a_i\}_{i=1,\dots,n}$ and $\{b_i\}_{i=1,\dots,n}$. Thus, on every processor there are $V/2$ time units left for processing the remaining tasks of chain $\{c_i\}_{i=1,\dots,n}$. And therefore, there must exist a partition for instance P. □

The reduction in the proof of Thm. 4 adapts naturally to the problem of scheduling parallel tasks.

Corollary 2. *The problem of scheduling parallel tasks, each using exactly one processor, under precedence constraints of width bounded by a constant $\omega \ge 3$ on any fixed number of processors $m \ge 2$ is NP-hard.*

Theorem 5. *The problem of scheduling malleable tasks with precedence constraints of width $\omega = 2$, is NP-hard on any fixed number of processors $m \ge 3$, even under the monotonous penalty assumption.*

Proof. We give a reduction from an arbitrary instance of the NP-complete KNAPSACK decision problem, see [7], to the problem of scheduling 2 independent chains of 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 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., $\sum_{i\in A} w_i \le W$ and a complement valued at most V, i.e., $\sum_{i\notin A} v_i \le V$? 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 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 the same processing time $p_b := 2p_h$ on 2 and 3 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. These processing times clearly obey the monotonous penalty assumption.

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

Given a feasible solution set A for K we can construct a feasible scheduling solution for the corresponding instance S as follows: For every item $i \in A$ we allot 2 processors to task j_i and 1 processor to task $\bar{j}_i$; for every item $i \notin A$ vice versa. The remaining tasks get 1 processor. We schedule every task directly after the completion of its predecessor, i.e., $\sigma_{j_i} := (i-1)p_h + \sum_{k<i} p_{j_k}(\alpha_{j_k})$, $\sigma_{h_i} := (i-1)p_h + \sum_{k\le i} p_{j_k}(\alpha_{j_k})$; the 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.,

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

This holds analogously for $\bar{j}_{i-1}$ and j_i. Thus, there are never more than 3 processors in use and the schedule is feasible. 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 $(n-1)p_h + np_b + V$.

For a given schedule 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 $(n-1)p_h + np_b + \sum_{i\in A} w_i \le (n-1)p_h + np_b + V$ which implies $\sum_{i\in A} w_i \le V$. On the other hand, $(n-1)p_h + np_b + \sum_{i\notin A} v_i \le (n-1)p_h + np_b + V$ which satisfies $\sum_{i\notin A} v_i \le V$. □

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. However, in Cor. 1 we have already 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. 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 in [11, 12].

Theorem 6. *The problem of scheduling malleable tasks with precedence constraints that form a caterpillar is NP-hard for every fixed $m \ge 2$, 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 NP-complete 3-PARTITION decision problem, see [7]. 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 $i = 1,\dots,3z$, and the question is whether there exists a partition of $\{1,\dots,3z\}$ into z disjoint sets $A_1,\dots,A_z$ such that $\sum_{i\in A_j} a_i = B$ for all $j = 1,\dots,z$.

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 $i = 1,\dots,3z$ and two tasks g_i, h_i for

all $i = 1, \ldots, z$ with the following processing times with monotonous penalties:

$$p_{j_i}(1) = a_i, \qquad p_{j_i}(2) = a_i; \qquad\qquad p_{k_i}(1) = 2B, \qquad p_{k_i}(2) = B;$$
$$p_{g_i}(1) = B, \qquad p_{g_i}(2) = B; \qquad\qquad p_{h_i}(1) = 2B, \qquad p_{h_i}(2) = B.$$

We introduce the following precedence constraints, which obviously form a tree:

$$k_1 \prec \cdots \prec k_{3z} \prec g_1 \prec h_1 \prec \cdots \prec g_z \prec h_z \tag{1}$$
$$k_1 \prec j_1, \ldots, k_{3z} \prec j_{3z} \tag{2}$$

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 h_i on two processors, and one task g_i on one processor. In parallel to each g_i we schedule three tasks a_i belonging to the same subset A_x in the partition. This results in a schedule with makespan

$$3z \cdot p_{k_i}(2) + z \cdot p_{h_i}(2) + \sum_{i=1,\ldots,z} \max\{B, \sum_{k \in A_i} a_k\} = 3zB + zB + zB = 5zB.$$

If there exists a solution to S with makespan at most $5zb$, we know that all tasks k_i and h_i must run on two processors, already accounting for time $4zB$ during which all processors are busy. The remaining tasks j_i and g_i must thus be scheduled within time zB, and this can only be achieved when all of these tasks are allotted only one processor and there are no gaps in their schedule.

Due to precedence constraints (1), tasks g_i need to be scheduled alternately with tasks h_i. Since the latter run on two processors each, the only way to avoid gaps is to divide tasks a_i into B triplets of length z each, and to process each triplet in parallel to one task g_i. □

This reduction can be adapted to suit the case of parallel tasks.

Corollary 3. *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 (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 $\{1, \ldots, m\}$. Clearly, any strip packing instance with rational data can be stated this way.

Note that even the 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. This was shown in [5] in the context of assigning check-in counters at airports, and independently in [8]. We will argue, however, that all of our algorithms and reductions can be adapted to yield contiguous processors by construction. Thus, our results also hold for the corresponding strip packing problems. We omit the detailed proofs due to space constraints.

First, the dynamic program from Sec. 2.1 can easily be adapted to yield contiguous schedules: We merely need to keep track of the distinct processors used by every task in each state. Hence, the deduced FPTAS remains valid with a running time increased by a factor $m^{2\omega}$.

Corollary 4. *There exists an FPTAS for finding an optimal contiguous schedule of malleable tasks under precedence constraints of bounded width.*

Note that our algorithm yields a polynomial running time for classical strip packing instances with integral rectangle widths, only when the strip width m is polynomial in the input size. This assumption is quite common, see [9].

Next, observe that for $m \leq 2$, any schedule is contiguous. Consequently, Thm. 3 can be formulated for the contiguous case. Also, Cor. 1 remains valid, since for $\omega = 2$, at most two parallel tasks can be processed concurrently.

Corollary 5. *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 Thms. 4 and 6 only use $m = 2$ processors. Also, the scheduling instances arising from the reduction for Thm. 5 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 6. *Even when assuming monotonous penalties, contiguous scheduling of malleable and parallel tasks is NP-hard under precedence constraint which form a caterpillar on $m \geq 2$ processors, and under precedence constraints of width bounded by a constant, ω, with $\omega \geq 3$. For malleable tasks, it remains NP-hard for $\omega = 2$ when $m \geq 3$.*

Acknowledgments. 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. In *Proceedings of SPAA*, pages 180–189, 2006.
2. J. Błażewicz and Z. Liu. Scheduling multiprocessor tasks with chain constraints. *European Journal of Operational Research*, 94(2):231–241, 1996.
3. R. P. Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, 51(1):161–166, 1950.

4. M. Drozdowski. Scheduling multiprocessor tasks: an overview. *European Journal of Operational Research*, 94(2):215–230, 1996.
5. C. W. Duin and E. V. Sluis. On the complexity of adjacent resource scheduling. *Journal of Scheduling*, 9(1):49–62, 2006.
6. D. R. Fulkerson. Note on Dilworth's decomposition theorem for partially ordered sets. *Proceedings of the American Mathematical Society*, 7(4):701–702, 1956.
7. 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.
8. E. Günther. Bin Scheduling: Partitionieren verformbarer Jobs mit Nebenbedingungen (in German). Master's thesis, Technische Universität Berlin, 2008.
9. K. Jansen and R. Thöle. Approximation algorithms for scheduling parallel jobs: Breaking the approximation ratio of 2. In *Proceedings of ICALP*, pages 234–245, 2008.
10. 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.
11. 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.
12. 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.
13. J.-T. Leung, editor. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. Chapman and Hall/CRC, 2004.
14. R. H. Möhring. Computationally tractable classes of ordered sets. In I. Rival, editor, *Algorithms and Order*, pages 105–194. Kluwer Academic Publishers, 1989.
15. J. Turek, J. Wolf, and P. Yu. Approximate algorithms for scheduling parallelizable tasks. In *Proceedings of SPAA*, pages 323–332, 1992.
16. 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.
17. 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.