Malleable Scheduling Beyond Identical Machines

Dimitris Fotakis
School of Electrical and Computer Engineering, National Technical University of Athens, Greece
https://www.softlab.ntua.gr/~fotakis/
fotakis@cs.ntua.gr

Jannik Matuschke
Research Center for Operations Management, KU Leuven, Belgium
https://sites.google.com/view/jannikmatuschke/
jannik.matuschke@kuleuven.be

Orestis Papadigenopoulos
Department of Computer Science, The University of Texas at Austin, USA
http://www.cs.utexas.edu/~papadig/
papadig@cs.utexas.edu

Abstract

In malleable job scheduling, jobs can be executed simultaneously on multiple machines with the processing time depending on the number of allocated machines. Jobs are required to be executed non-preemptively and in unison, in the sense that they occupy, during their execution, the same time interval over all the machines of the allocated set. In this work, we study generalizations of malleable job scheduling inspired by standard scheduling on unrelated machines. Specifically, we introduce a general model of malleable job scheduling, where each machine has a (possibly different) speed for each job, and the processing time of a job j on a set of allocated machines S depends on the total speed of S for j. For machines with unrelated speeds, we show that the optimal makespan cannot be approximated within a factor less than $\frac{e}{e-1}$, unless $P = NP$. On the positive side, we present polynomial-time algorithms with approximation ratios $\frac{2e}{e-1}$ for machines with unrelated speeds, 3 for machines with uniform speeds, and 7/3 for restricted assignments on identical machines. Our algorithms are based on deterministic LP rounding and result in sparse schedules, in the sense that each machine shares at most one job with other machines. We also prove lower bounds on the integrality gap of $1 + \varphi$ for unrelated speeds (φ is the golden ratio) and 2 for uniform speeds and restricted assignments. To indicate the generality of our approach, we show that it also yields constant factor approximation algorithms (i) for minimizing the sum of weighted completion times; and (ii) a variant where we determine the effective speed of a set of allocated machines based on the L_p norm of their speeds.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis; Theory of computation → Scheduling algorithms

Keywords and phrases malleable, jobs, moldable, machines, unrelated, uniform, parallel, speeds, approximation, scheduling

Digital Object Identifier 10.4230/LIPIcs.APPROX-RANDOM.2019.17

Category APPROX

Related Version A full version of the paper is available at https://arxiv.org/abs/1903.11016.

Acknowledgements Part of this work was carried out while the authors participated in the program "Real-Time Decision Making" at the Simons Institute for the Theory of Computing, Berkeley, CA.

1 Introduction

Since the late 60s, various models have been proposed by researchers [7, 8] in order to capture the real-world aspects and particularities of multiprocessor task scheduling systems, i.e., large collections of identical processors able to process tasks in parallel. High performance computing, parallel architectures, and cloud services are typical applications that motivate the study of multiprocessor scheduling, both theoretical and practical. An influential model is Rayward-Smith's unit execution time and unit communication time (UET-UCT) model [22], where each parallel job is partitioned into a set of tasks of unit execution time and these tasks are subject to precedence constraints modeled by a task graph. The UET-UCT model and its generalizations have been widely studied and a large number of (approximation) algorithms and complexity results have been proposed [10, 20].

However, the UET-UCT model mostly focuses on task scheduling and sequencing, and does not account for the amount of resources allocated to each job, thus failing to capture an important aspect of real-world parallel systems. Specifically, in the UET-UCT model, the level of granularity of a job (that is, the number of smaller tasks that a job is partitioned into) is decided a priori and is given as part of the input. However, it is common ground in the field of parallel processing that the unconditional allocation of resources for the execution of a job may jeopardize the overall efficiency of a multiprocessor system. A theoretical explanation is provided by Amdahl's law [1], which suggests that the speedup of a job's execution can be estimated by the formula $\frac{1}{(1-p)+\frac{p}{s}}$, where p is the fraction of the job that can be parallelized and s is the speedup due to parallelization (i.e., s can be thought as the number of processors).

Malleable Scheduling. An interesting alternative to the UET-UCT model is that of *malleable*[1] job scheduling [5, 24]. In this setting, a set $\mathcal{J}$ of jobs is scheduled on a set $\mathcal{M}$ of parallel machine(-s), while every job can be processed by more than one machines at the same time (i.e., by partitioning the job into tasks). In order to quantify the effect of parallelization, the processing time of a job $j \in \mathcal{J}$ is determined by a function $f_j : \mathbb{N} \to \mathbb{R}_+$[2] depending on the number of allocated machines. Moreover, every job must be executed *non-preemptively* and in *unison*, i.e. having the same starting and completion time on each of the allocated machines. Thus, if a job j is assigned to a set of machines S starting at time τ, all machines in S are occupied with job j during the interval $[\tau, \tau + f_j(|S|)]$. It is commonly assumed that the processing time function of a job exhibits two useful and well-motivated properties:

- For every job $j \in \mathcal{J}$, the processing time $f_j(s)$ is *non-increasing* in the number of machines.[3]
- The total *work* of the execution of a job j on s machines, that is the product $s \cdot f_j(s)$, is *non-decreasing* in the number of machines.

The latter property, known as *monotonicity* of a malleable job, is justified by Brent's law [3]: One cannot expect superlinear speedup by increasing the level of parallelism. A great deal of theoretical results have been published on scheduling malleable jobs according to the above model (and its variants) for the objective of minimizing the *makespan*, i.e., the completion time of the last finishing job, or other standard objectives (see, e.g., [6] and the references therein).

---

[1] Malleable scheduling also appears as *moldable*, while sometimes the two terms refer to slightly different models.

[2] We denote by $\mathbb{R}_+$ (resp. $\mathbb{Z}_+$) the set of non-negative reals (resp. integers).

[3] This property holds w.l.o.g., as the system always has the choice not to use some of the allocated machines.

Although malleable job scheduling represents a valiant attempt to capture real-world aspects of massively parallel processing, the latter exhibits even more complicated characteristics. Machine heterogeneity, data locality and hardware interconnection are just a few aspects of real-life systems that make the generalization of the aforementioned model necessary. In modern multiprocessor systems, machines are not all identical and the processing time of a job not only depends on the quantity, but also on the quality of the set of allocated machines. Indeed, different physical machines may have different capabilities in terms of faster CPUs or more efficient cache hierarchies. Moreover, the above heterogeneity may be job-dependent, in the sense that a specific machine may be faster when executing a certain type of jobs than another (e.g., memory- vs arithmetic-intensive applications [21]). Finally, the execution of a job on specific combinations of machines may also yield additional benefit (e.g., machines that are local in terms of memory hierarchy).

Our Model: Malleable Scheduling on Unrelated Machines. Quite surprisingly, no results exist on scheduling malleable jobs beyond the case of identical machines, to the best of our knowledge, despite the significant theoretical and practical interest in the model. In this work, we extend the model of malleable job scheduling to capture more delicate aspects of parallel job scheduling. In this direction, while we still require our jobs to be executed non-preemptively and in unison, the processing time of a job $j \in \mathcal{J}$ becomes a set function $f_j(S)$, where $S \subseteq \mathcal{M}$ is the set of allocated machines. We require that processing times are given by a *non-increasing* function, in the set function context, while additional assumptions on the scalability of f_j are made, in order to capture the *diminishing utility* property implied by Brent's law.

These assumptions naturally lead to a generalized malleable job setting, where processing times are given by non-increasing supermodular set functions $f_j(S)$, accessed by value queries. We show that makespan minimization in this general setting is inapproximable within $\mathcal{O}(|\mathcal{J}|^{1-\varepsilon})$ factors (unless $P = NP$, see Section 4.3). The general message of the proof is that unless we make some relatively strong assumptions on processing times (in the form e.g., of a relatively smooth gradual decrease in the processing time, as more machines are allocated), malleable job scheduling (even with monotone supermodular processing times) can encode combinatorial problems as hard as graph coloring.

Thus, inspired by (standard non-malleable) scheduling models on uniformly related and unrelated machines, we introduce the notion of *speed-implementable* processing time functions. For each machine i and each job j there is a *speed* $s_{i,j} \in \mathbb{Z}_+$ that quantifies the contribution of machine i to the execution of job j, if i is included in the set allocated to j. For most of this work, we assume that the total speed of an allocated set is given by an additive function $\sigma_j(S) = \sum_{i \in S} s_{i,j}$ (but see also Section 4.1, where we discuss more general speed functions based on L_p-norms). A function is speed-implementable if we can write $f_j(S) = f_j(\sigma_j(S))$ for some function $f_j : \mathbb{R}_+ \to \mathbb{R}_+$. Again, we assume oracle access to the processing time functions.[4]

The notion of speed-implementable processing times allows us to quantify the fundamental assumptions of *monotonicity* and *diminishing utility* in a clean and natural way. More specifically, we make the following two assumptions on speed-implementable functions:

1. *Non-increasing processing time.* For every job $j \in \mathcal{J}$, the processing time $f_j(s)$ is non-increasing in the total allocated speed $s \in \mathbb{R}_+$.
2. *Non-decreasing work.* For every job $j \in \mathcal{J}$, the work $f_j(s) \cdot s$ is non-decreasing in the total allocated speed $s \in \mathbb{R}_+$.

[4] For convenience, we use the identifier f_j for both functions. Since their arguments come from disjoint domains, it is always clear from the context which one is meant.

The first assumption ensures that allocating more speed cannot increase the processing time. The second assumption is justified by Brent's law, when the increase in speed coincides with an increase in the physical number of machines, or by similar arguments for the increase of the total speed of a single physical machine (e.g., memory access, I/O bottleneck [21] etc.). We remark that speed-implementable functions with non-increasing processing times and non-decreasing work do not need to be convex, and thus, do not belong to the class of supermodular functions.

In this work, we focus on the objective of minimizing the *makespan* $C_{max} = \max_{j \in \mathcal{J}} C_j$, where C_j the completion time of job j. We refer to this setting as the problem of *scheduling malleable jobs on unrelated machines*. To further justify this term, we present a pseudopolynomial transformation of standard scheduling on unrelated machines to malleable scheduling with speed-implementable processing times (see the full version of this reading). The reduction can be rendered polynomial by standard techniques, preserving approximation factors with a loss of $1 + \varepsilon$.

1.1 Related Work

The problem of malleable job scheduling on identical machines has been studied thoroughly for more than three decades. For the case of non-monotonic jobs, i.e., jobs that do not satisfy the monotonic work condition, Du and Leung [5] show that the problem is strongly NP-hard for more than 5 machines, while in terms of approximation, Turek, Wolf and Yu [24] provided the first 2-approximation algorithm for the same version of the problem. Jansen and Porkolab [13] devised a PTAS for instances with a constant number of machines, which was later extended by Jansen and Thöle [15] to a PTAS for the case that the number of machines is polynomial in the number of jobs.

For the case of monotonic jobs, Mounié, Rapine and Trystram [19] propose a $\frac{3}{2}$-approximation algorithm, improving on the $\sqrt{3}$-approximation provided by the same authors [18]. Recently, Jansen and Land [12] gave an FPTAS for the case that $|\mathcal{M}| \geq 8|\mathcal{J}|/\varepsilon$. Together with the approximation scheme for polynomial number of machines in [12], this implies a PTAS for scheduling monotonic malleable jobs on identical machines.

Several papers also consider the problem of scheduling malleable jobs with preemption and/or under precedence constraints [2, 14, 17]. An interesting alternative approach to the general problem is that of Srinivasa, Prasanna, and Musicus [23], who consider a continuous version of malleable tasks and develop an exact algorithm based on optimal control theory under certain assumptions on the processing time functions. While the problem of malleable scheduling on identical machines is very well understood, this is not true for malleable extensions of other standard scheduling models, such as unrelated machines or the restricted assignment model. We attempt to close this gap by introducing and investigating malleable scheduling with speed-implementable processing time functions.

A scheduling model similar to malleable tasks is that of *splittable jobs*. In this regime, jobs can be split arbitrarily and the resulting parts can be distributed arbitrarily on different machines. For each pair of job j and machine i, there is a setup time s_{ij} and a processing time p_{ij}. If a fraction $x_{ij} \in (0, 1]$ of job j is to be scheduled on machine i, the load that is incurred on the machine is $s_{ij} + p_{ij}x_{ij}$. Correa et al. [4] provide an $(1 + \varphi)$-approximation algorithm for this setting (where φ is the golden ratio), which is based on an adaptation of the classic LP rounding result by Lenstra, Shmoys, and Tardos [16] for the traditional unrelated machine scheduling problem. We remark that the generalized malleable setting considered in this paper also induces a natural generalization of the splittable setting beyond setup times, when dropping the requirement that jobs need to be executed in unison. As

in [4], we provide a rounding framework based on a variant of the assignment LP from [16]. However, the fact that processing times are only given implicitly as functions in our setting makes it necessary to very carefully choose the coefficients of the assignment LP in order to ensure a constant integrality gap. Furthermore, because jobs have to be executed in unison, we employ a more sophisticated rounding scheme in order to better utilize free capacity on different machines.

1.2 Contribution and Techniques

At the conceptual level, we introduce the notion of malleable jobs with speed-implementable processing times. Hence, we generalize the standard and well-studied setting of malleable job scheduling, in a direct analogy to fundamental models in scheduling theory (e.g., scheduling on *uniformly related* and *unrelated* machines). This new and much richer model gives rise to a large family of unexplored packing problems that may be of independent interest. All omitted proofs can be found in the full version of this paper (see `https://arxiv.org/abs/1903.11016`).

From a technical viewpoint, we investigate the computational complexity and the approximability of this new setting. To the best of our understanding, standard techniques used for makespan minimization in the setting of malleable job scheduling on identical machines, such as the two-shelve approach (as used in [19, 24]) and area charging arguments, fail to yield any reasonable approximation guarantees in our more general setting. This intuition is supported by the following hardness of approximation result.

▶ **Theorem 1.** *For any $\epsilon > 0$, there is no $(\frac{e}{e-1} - \epsilon)$-approximation algorithm for the problem of scheduling malleable jobs on unrelated machines, unless $P = NP$.*

Note that the lower bound of $\frac{e}{e-1}$ is strictly larger than the currently best known approximation factor of 1.5 for malleable scheduling on identical machines.

Our positive results are based on a linear programming relaxation, denoted by [LP(C)] and described in Section 2. This LP resembles the assignment LP for the standard setting of non-malleable scheduling [16]. However, in order to obtain a constant integrality gap we distinguish between "small" jobs that can be processed on a single machine (within a given target makespan), and "large" jobs that have to be processed on multiple machines. For the large jobs, we carefully estimate their contribution to the load of their allocated machines. Specifically, we introduce the notion of *critical speed* and use the critical speed to define the load coefficients incurred by large jobs on machines in the LP relaxation by proportionally distributing the work volume according to machine speeds. For the rounding, we exploit the sparsity of our relaxation's extreme points (as in [16]) and generalize the approach of [4], in order to carefully distinguish between jobs assigned to a single machine and jobs shared by multiple machines.

▶ **Theorem 2.** *There exists a polynomial-time $\frac{2e}{e-1}$-approximation algorithm for the problem of scheduling malleable jobs on unrelated machines.*

An interesting corollary is that for malleable job scheduling on unrelated machines, there always exists an approximate solution where each machine shares at most one job with some other machines. We also get improved approximation guarantees for the special cases of *restricted assignment* and *uniform speeds*, respectively, by exploiting the special structure of the processing time functions.

▶ **Theorem 3.** *There exists a polynomial-time $\frac{7}{3}$-approximation algorithm for the problem of scheduling malleable jobs on restricted identical machines (i.e., $s_{i,j} \in \{0,1\}$ for all $i \in \mathcal{M}$ and $j \in \mathcal{J}$).*

▶ **Theorem 4.** *There exists a polynomial-time 3-approximation algorithm for the problem of scheduling malleable jobs on uniform machines (i.e., $s_{i,j} = s_i$ for all $i \in \mathcal{M}$ and $j \in \mathcal{J}$).*

All our approximation results imply corresponding upper bounds on the integrality gap of the linear programming relaxation [LP(C)]. Based on an adaptation of a construction in [4], we show a lower bound of $1 + \varphi \approx 2.618$ on the integrality gap of [LP(C)] for malleable job scheduling on unrelated machines, where φ is the golden ratio. For the cases of restricted assignment and uniformly related machines, respectively, we obtain an integrality gap of 2.

Moreover, we extend our model and approach in two directions. First, we consider a setting where the *effective speed* according to which a set S of allocated machines processing a job j is given by the L_p-norm $\sigma_j^{(p)}(S) = \left(\sum_{i \in S}(s_{i,j})^p\right)^{1/p}$ of the corresponding speed vector. In practical settings, we tend to prefer assignments to relatively small sets of physical machines, so as to avoid delays related to communication, memory access, and I/O (see e.g., [21]). By replacing the total speed (i.e., the L_1-norm) with the L_p-norm of the speed vector for some $p \geq 1$, we discount the contribution of additional machines (especially of smaller speeds) towards processing a job j. Thus, as p increases, we give stronger preference to *sparse* schedules, where the number of jobs shared between different machines (and the number of machines sharing a job) are kept small. Interestingly, our general approach is robust to this generalization and results in constant approximation factors for any $p \geq 1$. Asymptotically, the approximation factor is bounded by $\frac{p}{p - \ln p} + \sqrt[p]{\frac{p}{\ln p}}$ and our algorithm smoothly converges to the algorithm of [16] as p tends to infinity. For the extreme case where we use the L_∞-norm, our setting becomes identical to standard scheduling on unrelated machines and we recover the algorithm of [16], achieving an approximation ratio of 2. These results are discussed in Section 4.1.

In another direction, we combine our approach for makespan minimization with standard techniques employed for the objective of total weighted completion time, $\sum_{j \in \mathcal{J}} w_j C_j$, and obtain a constant factor approximation for minimizing the total weighted completion time for malleable job scheduling on unrelated machines. These results are discussed in Section 4.2.

Trying to generalize malleable job scheduling beyond the simple setting of identical machines, as much as possible, we believe that our setting with speed-implementable processing times lies on the frontier of the constant-factor approximability regime. We show a strong inapproximability lower bound of $\mathcal{O}(|\mathcal{J}|^{1-\varepsilon})$ for the (far more general) setting where the processing times are given by a non-increasing supermodular set functions. These results are discussed in Section 4.3. An interesting open question is to characterize the class of processing time functions for which malleable job scheduling admits constant factor (and/or logarithmic) approximation guarantees.

2 The general rounding framework

In this section, we provide a high-level description of our algorithm. We construct a polynomial-time *ρ-relaxed decision procedure* for malleable job scheduling problems. This procedure takes as input an instance of the problem as well as a target makespan C and either asserts correctly that there is no feasible schedule of makespan at most C, or returns a feasible schedule of makespan at most ρC. It is well-known that a ρ-relaxed decision procedure can

be transformed into a polynomial-time ρ-approximation algorithm [11] provided that one can compute proper lower and upper bounds to the optimal value of size polynomial in the size of the input.

Given a target makespan C, let $\gamma_j(C) := \min\{q \in \mathbb{Z}_+ \mid f_j(q) \leq C\}$ be the *critical speed* of job $j \in \mathcal{J}$. Moreover, we define for every $i \in \mathcal{M}$ the sets $J_i^+(C) := \{j \mid f(s_{i,j}) \leq C\}$ and $J_i^-(C) := J \backslash J_i^+(C)$ to be the set of jobs that can or cannot be processed by i alone within time C, respectively. Note that $\gamma_j(C)$ can be computed in polynomial-time given oracle access to f_j by performing binary search. When C is clear from the context, we use the short-hand notation γ_j, J_i^+, and J_i^- instead. The following technical fact is equivalent to the non-decreasing work property and is used throughout the proofs of this paper:

▶ **Fact 5.** *Let f be a speed-implementable processing time function satisfying the properties of our problem. Then for every speed $q \in \mathbb{R}_+$ we have that:*

1. *$f(\alpha q) \leq \frac{1}{\alpha} f(q)$ for every $\alpha \in (0,1)$, and*
2. *$f(q') \leq \frac{q}{q'} f(q)$ for every $q' \leq q$.*

The following feasibility LP is the starting point of the relaxed decision procedures we construct in this work:

$$\text{[LP(C)]:} \qquad \sum_{i \in \mathcal{M}} x_{i,j} = 1 \qquad \forall j \in \mathcal{J} \tag{1}$$

$$\sum_{j \in J_i^+} f_j(s_{i,j}) x_{i,j} + \sum_{j \in J_i^-} \frac{f_j(\gamma_j)\gamma_j}{s_{i,j}} x_{i,j} \leq C \qquad \forall i \in \mathcal{M} \tag{2}$$

$$x_{i,j} \geq 0 \quad \forall j \in \mathcal{J}, i \in \mathcal{M} \tag{3}$$

In the above LP, each variable $x_{i,j}$ can be thought as the fraction of job j that is assigned to machine i. The equality constraints (1) ensure that each job is fully assigned to a subset of machines, while constraints (2) impose an upper bound to the load of every machine. As we can prove, the above formulation is feasible for any C that is greater than the optimal makespan.

▶ **Proposition 6.** *For every $C \geq \text{OPT}$, where OPT is the makespan of an optimal schedule, [LP(C)] has a feasible solution.*

Proof. Fix a schedule of makespan OPT and let $S_j \subseteq \mathcal{M}$ be the set of machines allocated to a job j in that schedule. For every $i \in \mathcal{M}$,$j \in \mathcal{J}$ set $x_{i,j} = \frac{s_{i,j}}{\sigma_j(S_j)}$ if $i \in S_j$ and $x_{i,j} = 0$, otherwise. We show that x is a feasible solution to $[LP(C)]$. Indeed, constraints (1) are satisfied since $\sum_{i \in \mathcal{M}} x_{i,j} = \sum_{i \in S_j} \frac{s_{i,j}}{\sigma_j(S_j)} = 1$ for all $j \in \mathcal{J}$. For verifying that constraints (2) are fulfilled, let $j \in \mathcal{J}$ and $i \in S_j$. If $j \in J_i^+$ then $f_j(s_{i,j}) x_{i,j} = f_j(s_{i,j}) \frac{s_{i,j}}{\sigma_j(S_j)} \leq f_j(S_j)$, using Fact 5. If $j \in J_i^-$ then $\frac{f_j(\gamma_j)\gamma_j}{s_{i,j}} x_{i,j} = \frac{f_j(\gamma_j)\gamma_j}{\sigma_j(S_j)} \leq \frac{\sigma_j(S_j) f_j(S_j)}{\sigma_j(S_j)} \leq f_j(S_j)$, again using Fact 5 and the fact that $\sigma_j(S_j) \geq \gamma_j$. Therefore for any $i \in \mathcal{M}$ we obtain: $\sum_{j \in J_i^+} f_j(s_{i,j}) x_{i,j} + \sum_{j \in J_i^-} \frac{f_j(\gamma_j)\gamma_j}{s_{i,j}} x_{i,j} \leq \sum_{j \in J | i \in S_j} f_j(S_j) \leq OPT \leq C$. ◀

Assuming that $C \geq \text{OPT}$, let x be an extreme point solution to [LP(C)]. We create the *assignment graph* $\mathcal{G}(x)$ with nodes $V := \mathcal{J} \cup \mathcal{M}$ and edges $E := \{\{i,j\} \in \mathcal{M} \times \mathcal{J} \mid x_{i,j} > 0\}$, i.e., one edge for each machine-job pair in the support of the LP solution. Notice that $\mathcal{G}(x)$ is bipartite by definition. Furthermore, since [LP(C)] is structurally identical to the LP of unrelated machine scheduling [16], the choice of x as an extreme point guarantees the following sparsity property:

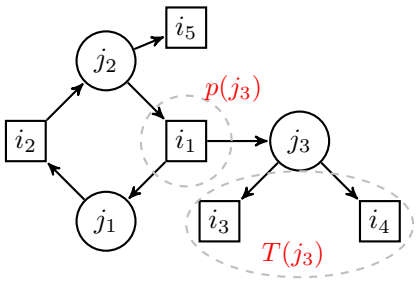

Figure 1 A properly oriented pseudotree with indegree at most 1 for each node.

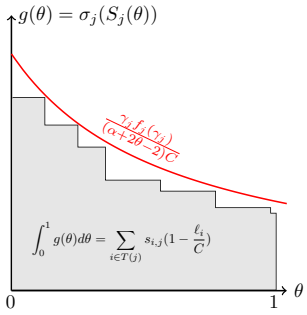

Figure 2 Volume argument for selecting a subset of the children machines in the proof of Proposition 10.

▶ **Proposition 7** ([16]). *For every extreme point solution x of [LP(C)], each connected component of $\mathcal{G}(x)$ contains at most one cycle.*

As a graph with at most one cycle is either a tree or a tree plus one edge, the connected components of $\mathcal{G}(x)$ are called *pseudotrees* and the whole graph is called a *pseudoforest*. It is not hard to see that the edges of an undirected pseudoforest can always be oriented in a way that every node has an *in-degree* of at most one. We call such a $\mathcal{G}(x)$ a *properly oriented* pseudoforest. Such an orientation can easily be obtained by first orienting the edges on the unique cycle (if it exists) consistently so as to obtain a directed cycle and, then, by orienting all remaining edges away from that cycle (see Figure 1).

Now fix a properly oriented $\mathcal{G}(x)$ with set of oriented edges $\bar{E}$. For $j \in \mathcal{J}$, we define $p(j) \in \mathcal{M}$ to be its unique *parent-machine* (if it exists) and $T(j) = \{i \in \mathcal{M} \mid (j,i) \in \bar{E}\}$ to be the set of *children-machines* of j, respectively. Notice, that for every machine i, there exists at most one $j \in \mathcal{J}$ such that $i \in T(j)$. The decision procedures we construct in this paper are based, unless otherwise stated, on the following scheme:

ALGORITHM: Given a target makespan C:

1. If [LP(C)] is feasible, compute an extreme point solution x of [LP(C)] and construct a properly oriented $\mathcal{G}(x)$. (Otherwise, report that $C < \text{OPT}$.)
2. A *rounding scheme* assigns every job $j \in \mathcal{J}$ either only to its parent machine $p(j)$, or to the set of its children-machines $T(j)$ (see Section 3).
3. According to the rounding, every job $j \in \mathcal{J}$ that has been assigned to $T(j)$ is placed at the beginning of the schedule (these jobs are assigned to disjoint sets of machines).
4. At any point a machine i becomes idle, it processes any unscheduled job j that has been rounded to i such that $i = p(j)$.

3 Rounding schemes

In each of the following rounding schemes, we are given as an input an extreme point solution x of [LP(C)] and a properly oriented pseudoforest $\mathcal{G}(x) = (V, \bar{E})$.

3.1 A simple 4-approximation for unrelated machines

We start from the following simple rounding scheme: For each job j, assign j to its parent-machine $p(j)$ if $x_{p(j),j} \geq \frac{1}{2}$, or else, assign j to its children-machines $T(j)$. Formally, let $\mathcal{J}^{(1)} := \{j \in \mathcal{J} \mid x_{p(j),j} \geq \frac{1}{2}\}$ be the sets of jobs that are assigned to their parent-machines and $\mathcal{J}^{(2)} := \mathcal{J} \setminus \mathcal{J}^{(1)}$ the rest of the jobs. Recall that we first run the jobs

in $J^{(2)}$ and then the jobs in $J^{(1)}$ as described at the end of the previous section. For $i \in \mathcal{M}$, define $J_i^{(1)} := \{j \in \mathcal{J}^{(1)} \mid p(j) = i\}$ and $J_i^{(2)} := \{j \in \mathcal{J}^{(2)} \mid i \in T(j)\}$ as the sets of jobs in $J^{(1)}$ and $J^{(2)}$, respectively, that get assigned to i (note that $|J_i^{(2)}| \leq 1$, as each machine gets assigned at most one job as a child-machine). Furthermore, let $\ell_i := \sum_{j \in J_i^+ \cap \mathcal{J}^{(1)}} f_j(s_{i,j}) x_{i,j} + \sum_{j \in J_i^- \cap \mathcal{J}^{(1)}} f_j(\gamma_j) \frac{\gamma_j}{s_{i,j}} x_{i,j}$ be the fractional load incurred by jobs in $J^{(1)}$ on machine i in the LP solution x.

▶ **Proposition 8.** *Let $i \in \mathcal{M}$. Then $\sum_{j \in \mathcal{J}_i^{(1)}} f_j(\{i\}) \leq 2\ell_i$.*

Proof. Let $j \in J_i^{(1)}$. Since $x_{i,j} \geq \frac{1}{2}$ by definition of $\mathcal{J}^{(1)}$, we get $f_j(s_{i,j}) \leq 2 f_j(s_{i,j}) x_{i,j}$. Furthermore, if $j \in J_i^-$ then $f_j(s_{i,j}) x_{i,j} \leq f_j(\gamma_j) \frac{\gamma_j}{s_{i,j}} x_{i,j}$ by Fact 5 and the fact that $s_{i,j} < \gamma_j$. Thus, by summing up over all jobs in $J_i^{(1)}$ and then applying constraints (2), we get

$$\sum_{j \in J_i^{(1)}} f_j(\{i\}) \leq 2 \left(\sum_{j \in J_i^{(1)} \cap J_i^+} f_j(s_{i,j}) x_{i,j} + \sum_{j \in J_i^{(1)} \cap J_i^-} \frac{f_j(\gamma_j)\gamma_j}{s_{i,j}} x_{i,j} \right) \leq 2\ell_i. \qquad \blacktriangleleft$$

▶ **Proposition 9.** *Let $j \in \mathcal{J}^{(2)}$. Then $f_j(T(j)) \leq 2C$.*

Proof. If there is a machine $i \in T(j)$ with $j \in J_i^+$, then $f_j(T(j)) \leq f_j(\{i\}) \leq C$. So we can assume that $j \in J_i^-$ for all $i \in T(j)$. Hence constraints (2) imply $f_j(\gamma_j) \frac{\gamma_j}{s_{i,j}} x_{i,j} \leq C$ for all $i \in T(j)$. Summing these constraints yields $\sum_{i \in T(j)} \frac{f_j(\gamma_j)}{C} \gamma_j x_{i,j} \leq \sigma_j(T(j))$. Using the fact that $f_j(\gamma_j) \leq C$ by definition of γ_j and $\sum_{i \in T(j)} x_{i,j} > \frac{1}{2}$ because $j \in \mathcal{J}^{(2)}$, we get $\sigma_j(T(j)) \geq \frac{1}{2} \gamma_j \frac{f_j(\gamma_j)}{C}$. This implies $f_j(T(j)) \leq 2C$ by Fact 5. $\blacktriangleleft$

Clearly, the load of any machine $i \in \mathcal{M}$ in the final schedule is the sum of the load due to the execution of $\mathcal{J}^{(1)}$, plus the processing time of at most one job of $\mathcal{J}^{(2)}$. By Proposition 8 and 9, it follows that any feasible solution of [LP(C)] can be rounded in polynomial-time into a feasible schedule of makespan at most $4C$.

3.2 An improved $\frac{2e}{e-1} \approx 3.163$-approximation for unrelated machines

In the simple rounding scheme described above, it can be the case that the overall makespan improves by assigning some job $j \in \mathcal{J}^{(2)}$ only to a subset of the machines in $T(j)$. This happens because some machines in $T(j)$ may have significantly higher load from jobs of $\mathcal{J}^{(1)}$ than others, but job j will incur the same additional load to all machines it is assigned to.

We can improve the approximation guarantee of the rounding scheme by taking this effect into account and filtering out children-machines with a high load. Define $\mathcal{J}^{(1)}$ and $\mathcal{J}^{(2)}$ as before. Every job in $j \in J^{(1)}$ is assigned to its parent-machine $p(j)$, while every job $j \in \mathcal{J}^{(2)}$ is assigned to a subset of $T(j)$ as follows.

For $j \in \mathcal{J}^{(2)}$ and $\theta \in [0,1]$ define $S_j(\theta) := \{i \in T(j) \mid 1 - \frac{\ell_i}{C} \geq \theta\}$. Choose θ_j so as to minimize $2(1-\theta_j)C + f_j(\theta_j)$ (note that this minimizer can be determined by trying out at most $|T(j)|$ different values for θ_j). We then assign each job in $j \in J^{(2)}$ to the machine set $S_j(\theta_j)$.

By Proposition 8, we know that the total load of each machine $i \in \mathcal{M}$ due to the execution of jobs from $\mathcal{J}^{(1)}$ is at most $2\ell_i$. Recall that there is at most one $j \in J^{(2)}$ with $i \in T(j)$. If $i \notin S_j(\theta_j)$, then load of machine i bounded by $2\ell_i \leq 2C$. If $i \in S_j(\theta_j)$, then the load of machine i is bounded by

$$\max_{i' \in S_j(\theta_j)} \left\{ 2\ell_{i'} + f_j(S_j(\theta_j)) \right\} \leq 2(1-\theta_j)C + f_j(S_j(\theta_j)), \tag{4}$$

where the inequality comes from the fact that $1 - \frac{\ell_{i'}}{C} \geq \theta_j$ for all $i' \in S_{\theta_j}$. The following proposition gives an upper bound on the RHS of (4) as a result of our filtering technique and proves Theorem 2.

▶ **Proposition 10.** *For each $j \in \mathcal{J}^{(2)}$, there is a $\theta \in [0,1]$ with $2(1-\theta)C + f_j(S_j(\theta)) \leq \frac{2e}{e-1}C$.*

Proof. Define $\alpha := \frac{2e}{e-1}$. We show that there is a $\theta \in [0,1]$ with $\sigma_j(S_j(\theta)) \geq \frac{\gamma_j f_j(\gamma_j)}{(\alpha+2\theta-2)C}$. Then $f_j(S_j(\theta)) \leq (\alpha + 2\theta - 2)C$ by Fact 5, implying the lemma.

Define the function $g : [0,1] \to \mathbb{R}_+$ by $g(\theta) := \sigma_j(S_j(\theta))$. It is easy to see g is non-increasing integrable and that

$$\int_0^1 g(\theta)d\theta = \sum_{i \in T(j)} s_{i,j}(1 - \frac{\ell_i}{C}).$$

See Figure 2 for an illustration.

Now assume by contradiction that $g(\theta) < \frac{\gamma_j f_j(\gamma_j)}{(\alpha+2\theta-2)C}$ for all $\theta \in [0,1]$. Note that $\ell_i + \frac{\gamma_j f_j(\gamma_j)}{s_{i,j}} x_{i,j} \leq C$ for every $i \in T(j)$ by constraints (2) and the fact that $\frac{\gamma_j f_j(\gamma_j)}{s_{i,j}} \leq f_j(s_{i,j})$ for all i such that $j \in J_i^+$. Hence $\frac{f_j(\gamma_j)\gamma_j}{C} x_{i,j} \leq s_{i,j}(1 - \frac{\ell_i}{C})$ for all $i \in T(j)$. Summing over all $i \in T(j)$ and using the fact that $\sum_{i \in T(j)} x_{i,j} \geq \frac{1}{2}$ because $j \in J^{(2)}$ we get

$$\frac{f_j(\gamma_j)\gamma_j}{2C} \leq \sum_{i \in T(j)} s_{i,j}(1 - \frac{\ell_i}{C}) = \int_0^1 g(\theta)d\theta < \frac{f_j(\gamma_j)\gamma_j}{C} \int_0^1 \frac{1}{\alpha + 2\theta - 2} d\theta,$$

where the last inequality uses the assumption that $g(\theta) < \frac{\gamma_j f_j(\gamma_j)}{(\alpha+2\theta_j-2)C}$ for all $\theta \in [0,1]$. By simplifying the above inequality, we get the contradiction

$$1 < \int_{\alpha-2}^{\alpha} \frac{1}{\lambda} d\lambda = \ln(\frac{\alpha}{\alpha-2}) = 1. \qquad \blacktriangleleft$$

By the above analysis, our main result for the case of unrelated machines follows.

▶ **Theorem 2.** *There exists a polynomial-time $\frac{2e}{e-1}$-approximation algorithm for the problem of scheduling malleable jobs on unrelated machines.*

▶ Remark 11. We can slightly improve the above analysis by optimizing the threshold of assigning a job to the parent. This optimization gives a slightly better approximation guarantee of $\alpha = \inf_{\beta \in (0,1)} \left\{ \frac{e^{\frac{1}{\beta}-1}}{\beta(e^{\frac{1}{\beta}-1}-1)} \right\} \approx 3.14619$.

3.3 A (7/3)-approximation for restricted identical machines

We are able to provide an algorithm of improved approximation guarantee for the special case of restricted identical machines: Each job $j \in \mathcal{J}$ is associated with a set of machines $\mathcal{M}_j \subseteq \mathcal{M}$, such that $s_{i,j} = 1$ for $i \in \mathcal{M}_j$ and $s_{i,j} = 0$, otherwise.

Given a feasible solution to [LP(C)] and a properly oriented $\mathcal{G}(x)$, we define the sets $\mathcal{J}^{(1)} := \{j \in \mathcal{J} \mid x_{p(j),j} = 1\}$ and $\mathcal{J}^{(2)} := \mathcal{J} \setminus \mathcal{J}^{(1)}$. The rounding scheme for this special case can be described as follows: **(a)** Every job $j \in \mathcal{J}^{(1)}$ is assigned to $p(j)$ (which is the only machine in $\mathcal{G}(x)$ that is assigned to j). **(b.i)** Every job of $j \in \mathcal{J}^{(2)}$ such that $|T(j)| = 1$ or $|T(j)| \geq 3$ is assigned to the set $T(j)$ of its children-machines. **(b.ii)** For every job of $j \in \mathcal{J}^{(2)}$ such that $|T(j)| = 2$, the algorithm schedules the job to the subset $S \subseteq T(j)$ that results in the minimum makespan over $T(j)$. Notice that for $|T(j)| = 2$ there are exactly three such subsets. As usual, the jobs of $\mathcal{J}^{(2)}$ are placed at the beginning of the schedule, followed by the jobs of $\mathcal{J}^{(1)}$.

▶ **Theorem 3.** *There exists a polynomial-time $\frac{7}{3}$-approximation algorithm for the problem of scheduling malleable jobs on restricted identical machines (i.e., $s_{i,j} \in \{0,1\}$ for all $i \in \mathcal{M}$ and $j \in \mathcal{J}$).*

3.4 A 3-approximation for uniform machines

We prove an algorithm of improved approximation guarantee for the special case of uniform machines, i.e., every machine has a unique speed s_i such that $s_{i,j} = s_i$ for all $j \in \mathcal{J}$. Given a target makespan C, we say that a machine i is *j-fast* for a job $j \in \mathcal{J}$ if $j \in J_i^+$, while we say that i is *j-slow* if $j \in J_i^-$. As opposed to the previous cases, the rounding for the uniform case starts by transforming the feasible solution of [LP(C)] into another extreme point solution that satisfies a useful structural property, as described in the following proposition.

▶ **Proposition 12.** *There is an extreme point solution x of [LP(C)] that satisfies the following property: For each $j \in \mathcal{J}$ there is at most one j-slow machine $i \in \mathcal{M}$ such that $x_{i,j} > 0$ and $x_{i,j'} > 0$ for some job $j' \neq j$. Furthermore, this machine, if it exists, is the slowest machine that j is assigned to.*

Let x be an extreme point solution of [LP(C)] that satisfies the property of Proposition 12 and let $\mathcal{G}(x)$ a properly oriented pseudoforest. By the above proposition, each job j has at most three types of assignments in $\mathcal{G}(x)$: (i) j-fast machines F_j, (ii) *exclusive* j-slow machines D_j, i.e. j-slow machines that are completely assigned to j, and (iii) at most one shared j-slow machine i_j (which is the slowest machine that j is assigned to).

We now describe the rounding scheme for the special case of uniform machines. For any job $j \in \mathcal{J}$ in any order: **(a)** If $x_{p(j),j} \geq \frac{1}{2}$ then j is assigned to its parent-machine $p(j)$, otherwise **(b)** j is assigned to a subset $S \subseteq T(j)$. In the second case, the subset S is chosen according to the following rule: **(b.i)** If $F_j \neq \emptyset$, then assign j to any $i \in F_j$, else **(b.ii)** if $\sigma_j(D_j) \geq \frac{\gamma_j f_j(\gamma_j)}{3C}$, then assign j only to the machines of D_j (but not to the shared i_j). In any other case, **(b.iii)** j is assigned to $D_j \cup \{i_j\}$.

▶ **Theorem 4.** *There exists a polynomial-time 3-approximation algorithm for the problem of scheduling malleable jobs on uniform machines (i.e., $s_{i,j} = s_i$ for all $i \in \mathcal{M}$ and $j \in \mathcal{J}$).*

4 Model extensions and discussion

4.1 Sparse allocations via p-norm regularization

In the model of speed-implementable processing time functions that we study in the previous sections, each function $f_j(S)$ depends on the total additive speed, yet is oblivious to the actual number of allocated machines. However, the overhead incurred by the synchronization of physical machines naturally depends on their number and we therefore need to take into account both the total speed and the cardinality of the machine set allocated to a job. In this section, we model the impact of the number of machines through the notion of *effective speed*. In this setting, every job j is associated with a speed *regularizer* $p_j \geq 1$, while the total speed of a set $S \subseteq \mathcal{M}$ is given by: $\sigma_j^{(p_j)}(S) = \left(\sum_{i \in S} s_{i,j}^{p_j}\right)^{\frac{1}{p_j}}$. For simplicity, we assume that every job has the same speed regularizer, $p = p_j, \forall j \in \mathcal{J}$.

Clearly, the choice of p controls the effect of the cardinality of a set to the resulting speed of an allocation, given that as p increases a sparse set has higher effective speed than a non-sparse set of the same total speed. Notice that for $p = 1$ we return to the standard case of additive speeds, while for $p \to \infty$, parallelization is no longer helpful

as $\lim_{p\to\infty} \sigma_j^{(p)}(S) = \max_{i\in S}\{s_{i,j}\}$. As before, the processing time functions satisfy the standard properties of malleable scheduling, i.e., $f_j(s)$ is non-increasing while $f_j(s)\cdot s$ is decreasing. For simplicity of presentation we assume that all jobs have the same regularizer p, but we comment on the case of job-dependent regularizers at the end of this section.

Quite surprisingly, we can easily modify the algorithms of the previous section in order to capture the above generalization. Given a target makespan C, we start from a new feasibility program [LP$^{(p)}(C)$], which is given by constraints (1),(3) of [LP(C)], combined with:

$$\sum_{j\in J_i^+} f_j(s_{i,j})x_{i,j} + \sum_{j\in J_i^-} f_j(\gamma_j)\left(\frac{\gamma_j}{s_{i,j}}\right)^p x_{i,j} \leq C \qquad \forall i \in \mathcal{M} \tag{5}$$

Note that J_i^+, J_i^-, and $\gamma_j(C)$ are defined exactly as before, and that the only difference between [LP(C)] and [LP$^{(p)}(C)$] is that we replace the coefficient $\frac{\gamma_j}{s_{i,j}}$ with $\left(\frac{\gamma_j}{s_{i,j}}\right)^p$ in constraints (2) of the former. It can be shown that for every $C \geq OPT$, where OPT is the makespan of an optimal schedule, [LP$^{(p)}$(C)] has a feasible solution.

The algorithm for this case is similar to the one of the standard case (see Section 3.1), having [$LP^{(p)}(C)$] as a starting point. Moreover, the rounding scheme is a parameterized version of the simple rounding of Section 3.1, with the difference that the threshold parameter $\beta \in [0,1]$ (i.e., the parameter that controls the decision of assigning a job j to either $p(j)$ or $T(j)$) is not necessarily $\frac{1}{2}$. In short, given a pseudoforest $\mathcal{G}(x)$, the rounding scheme assigns any job j to $p(j)$ if $x_{p(j),j} \geq \beta$, or to $T(j)$, otherwise.

By similar arguments as in Propositions 8,9, it can be proved that the makespan of the produced schedule is at most $\left(\frac{1}{\beta} + \frac{1}{(1-\beta)^{1/p}}\right)C$. Therefore, the algorithm can initially compute a threshold $\beta \in [0,1]$ that minimizes the above theoretical bound. Clearly, for $p = 1$ the minimizer of the expression is $\beta = 1/2$, yielding the 4-approximation of the standard case, while for $p \to +\infty$ one can verify that $\beta \to 1$ and:

$$\lim_{p\to+\infty} \inf_{\beta\in[0,1]} \left(\frac{1}{\beta} + \frac{1}{(1-\beta)^{1/p}}\right) = 2.$$

As expected, for the limit case where $p \to +\infty$, our algorithm converges to the well-known algorithm by Lenstra et al. [16] given that our problem becomes non-malleable. By using the standard approximation $\beta = 1 - \frac{\ln p}{p}$ for $p \geq 2$, we can prove the following theorem.

▶ **Theorem 13.** *Any feasible solution of [$LP^{(p)}(C)$] for $p \geq 2$ can be rounded in polynomial-time into a feasible schedule of makespan at most* $\left(\frac{p}{p-\ln p} + \sqrt[p]{\frac{p}{\ln p}}\right)C$.

Note that an analogous approach can handle the case where jobs have different regularizers, with the approximation ratio for this scenario determined by the smallest regularizer that appears in the instance (note that the approximation factor is always at most 4).

4.2 Minimizing the $\sum_{j\in\mathcal{J}} w_j C_j$ objective

The LP-based nature of our algorithms allows the design of efficient $\mathcal{O}(1)$-approximation algorithms for the objective of minimizing the sum of weighted completion times, i.e., $\sum_{j\in\mathcal{J}} w_j C_j$, employing the standard technique of *interval-indexed formulations* [9]. In this setting, every job $j \in \mathcal{J}$ is associated with a *weight* $w_j \in \mathbb{Z}_{\geq 0}$ and the objective is to compute a feasible schedule of minimum $\sum_{j\in\mathcal{J}} w_j C_j$, where C_j the completion time of job j. In the malleable setting, the approximation guarantee of our algorithm for the $\sum_{j\in\mathcal{J}} w_j C_j$ objective depends on the approximation guarantee of the underlying makespan problem.

▶ **Theorem 14.** *There exists a $\mathcal{O}(\rho)$-approximation algorithm for the problem of malleable scheduling minimizing the $\sum_{j\in\mathcal{J}} w_j C_j$ objective, where ρ the approximation ratio of the best rounding scheme of $[LP(C)]$.*

4.3 Supermodular processing time functions

In this paper we concentrated our study on speed-implementable processing time functions. However, the general definition of malleable scheduling given in Section 1 leaves room for many other possible variants of the problem with processing times given by monotone non-increasing set functions. One natural attempt of capturing the assumption of non-decreasing workload is to assume that for each job $j \in \mathcal{J}$ the corresponding processing time function f_j is supermodular, i.e.,

$$f_j(T \cup \{i\}) - f_j(T) \geq f_j(S \cup \{i\}) - f_j(S)$$

for all $S \subseteq T \subseteq \mathcal{M}$ and $i \in \mathcal{M} \setminus T$. The interpretation of this assumption is that the decrease in processing time when adding machine i diminishes the more machines are already used for job j (note that the terms on both sides of the inequality are non-positive because f_j is non-increasing). For this setting, which we refer to as *generalized malleable scheduling with supermodular processing time functions*, we derive a strong hardness of approximation result.

▶ **Theorem 15.** *There is no $|\mathcal{J}|^{1-\varepsilon}$-approximation for generalized malleable scheduling with supermodular processing time functions, unless $P = NP$.*

References

1 G. M. Amdahl. Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities. *IEEE Solid-State Circuits Society Newsletter*, 12(3):19–20, Summer 2007.

2 J. Blazewicz, M. Y. Kovalyov, M. Machowiak, D. Trystram, and J. Weglarz. Preemptable Malleable Task Scheduling Problem. *IEEE Trans. Comput.*, 55(4):486–490, April 2006.

3 R. P. Brent. The Parallel Evaluation of General Arithmetic Expressions. *J. ACM*, 21(2):201–206, April 1974.

4 J. Correa, A. Marchetti-Spaccamela, J. Matuschke, L. Stougie, O. Svensson, V. Verdugo, and J. Verschae. Strong LP formulations for scheduling splittable jobs on unrelated machines. *Mathematical Programming*, 154(1-2):305–328, 2015.

5 J. Du and J. Leung. Complexity of Scheduling Parallel Task Systems. *SIAM Journal on Discrete Mathematics*, 2(4):473–487, 1989.

6 P. Dutot, G. Mounié, and D. Trystram. Scheduling Parallel Tasks: Approximation Algorithms. In Joseph T. Leung, editor, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, chapter 26, pages 26–1–26–24. CRC Press, 2004.

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

8 R. Graham. Bounds on Multiprocessing Timing Anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, 1969.

9 L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein. Scheduling to Minimize Average Completion Time: Off-Line and On-Line Approximation Algorithms. *Math. Oper. Res.*, 22(3):513–544, August 1997.

10 C. Hanen and A. Munier. An approximation algorithm for scheduling dependent tasks on m processors with small communication delays. *Discrete Applied Mathematics*, 108(3):239–257, 2001.

11 D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: Theoretical and practical results. In *26th Annual Symposium on Foundations of Computer Science*, FOCS '85, pages 79–89, October 1985.

12 K. Jansen and F. Land. Scheduling Monotone Moldable Jobs in Linear Time. In *2018 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2018, Vancouver, BC, Canada, May 21-25, 2018*, pages 172–181, 2018.

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

14 K. Jansen and H. Zhang. An Approximation Algorithm for Scheduling Malleable Tasks Under General Precedence Constraints. *ACM Trans. Algorithms*, 2(3):416–434, July 2006.

15 Klaus Jansen and Ralf Thöle. Approximation algorithms for scheduling parallel jobs. *SIAM Journal on Computing*, 39(8):3571–3615, 2010.

16 J. K. Lenstra, D. B. Shmoys, and É. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46(1):259–271, January 1990.

17 Konstantin Makarychev and Debmalya Panigrahi. Precedence-Constrained Scheduling of Malleable Jobs with Preemption. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, pages 823–834, 2014.

18 G. Mounié, C. Rapine, and D. Trystram. Efficient Approximation Algorithms for Scheduling Malleable Tasks. In *Proceedings of the Eleventh Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA '99, Saint-Malo, France, June 27-30, 1999*, pages 23–32, 1999.

19 G. Mounié, C. Rapine, and D. Trystram. A 3/2-Approximation Algorithm for Scheduling Independent Monotonic Malleable Tasks. *SIAM J. Comput.*, 37(2):401–412, 2007.

20 C. H. Papadimitriou and M. Yannakakis. Towards an Architecture-independent Analysis of Parallel Algorithms. *SIAM J. Comput.*, 19(2):322–328, April 1990.

21 D. A. Patterson and J. L. Hennessy. *Computer Organization and Design, Fifth Edition: The Hardware/Software Interface*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 5th edition, 2013.

22 V. J. Rayward-Smith. UET Scheduling with Unit Interprocessor Communication Delays. *Discrete Appl. Math.*, 18(1):55–71, November 1987.

23 G. N. Srinivasa Prasanna and B. R. Musicus. Generalised Multiprocessor Scheduling Using Optimal Control. In *Proceedings of the Third Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '91, pages 216–228, New York, NY, USA, 1991. ACM.

24 J. Turek, J. L. Wolf, and P. S. Yu. Approximate Algorithms Scheduling Parallelizable Tasks. In *Proceedings of the Fourth Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '92, pages 323–332, New York, NY, USA, 1992. ACM.