SIAM J. COMPUT.
Vol. 41, No. 3, pp. 565–586

UNIVERSAL SEQUENCING ON AN UNRELIABLE MACHINE*

LEAH EPSTEIN†, ASAF LEVIN‡, ALBERTO MARCHETTI-SPACCAMELA§, NICOLE MEGOW¶, JULIÁN MESTRE‖, MARTIN SKUTELLA**, AND LEEN STOUGIE††

Abstract. We consider scheduling on an unreliable machine that may experience unexpected changes in processing speed or even full breakdowns. Our objective is to minimize $\sum w_j f(C_j)$ for any nondecreasing, nonnegative, differentiable cost function $f(C_j)$. We aim for a universal solution that performs well without adaptation for all cost functions for any possible machine behavior. We design a deterministic algorithm that finds a universal scheduling sequence with a solution value within 4 times the value of an optimal clairvoyant algorithm that knows the machine behavior in advance. A randomized version of this algorithm attains in expectation a ratio of e. We also show that both performance guarantees are best possible for any unbounded cost function. Our algorithms can be adapted to run in polynomial time with slightly increased cost. When jobs have individual release dates, the situation changes drastically. Even if all weights are equal, there are instances for which any universal solution is a factor of $\Omega(\log n/\log\log n)$ worse than an optimal sequence for any unbounded cost function. Motivated by this hardness, we study the special case when the processing time of each job is proportional to its weight. We present a nontrivial algorithm with a small constant performance guarantee.

Key words. scheduling, min-sum objective, single machine, unreliable machine, machine speed, universal solution, worst case guarantee

AMS subject classifications. 68M20, 68W25, 90B35

DOI. 10.1137/110844210

1. Introduction.

Traditional scheduling theory assumes in its standard models that jobs are processed on ideal machines that provide the same constant performance throughout time. While in some settings this is a good enough approximation of real life machine behavior, in other situations this assumption is decidedly unreasonable. A machine, for example, can be a server shared by multiple users; if other users suddenly increase their workload, this can cause a general slowdown, or even worse, the machine may become unavailable for a given user due to priority issues. In other cases, our machine may be a production unit that can break down altogether and remain offline for some time until it is repaired. In these cases, it is crucial to have schedules that take such unreliable machine behavior into account.

*Received by the editors August 12, 2011; accepted for publication (in revised form) March 9, 2012; published electronically May 31, 2012. An extended abstract with parts of this work appeared in *Proceedings of the 14th International Conference on Integer Programming and Combinatorial Optimization (IPCO XIV)*, Springer, New York, 2010.
http://www.siam.org/journals/sicomp/41-3/84421.html

†Department of Mathematics, University of Haifa, Haifa, Israel (lea@math.haifa.ac.il).

‡Faculty of Industrial Engineering and Management, The Technion, Haifa, Israel (levinas@ie.technion.ac.il).

§Department of Computer and System Sciences, Sapienza University of Rome, Rome, Italy (alberto.marchetti@dis.uniroma1.it). This author's work was supported by EU project 215270 FRONTS.

¶Max-Planck-Institut für Informatik, Saarbrücken, Germany (nmegow@mpi-inf.mpg.de).

‖School of Information Technologies, University of Sydney, Sydney, NSW, Australia (mestre@it.usyd.edu.au).

**Department of Mathematics, Technische Universität Berlin, Berlin, Germany (skutella@math.tu-berlin.de). This author's work was supported by DFG research center MATHEON in Berlin.

††Department of Econometrics and Operations Research, Vrije Universiteit Amsterdam & CWI, Amsterdam, The Netherlands (stougie@cwi.nl). This author's work was supported by the Tinbergen Institute.

Different machine behaviors will typically lead to very different optimal schedules. This creates a burden on the scheduler who has to periodically recompute the schedule from scratch. In some situations, recomputing the schedule may not even be feasible: when submitting a set of jobs to a server, a user can choose the order in which it presents these jobs but cannot alter this ordering later on. Therefore, it is desirable in general to have a fixed master schedule that will perform well regardless of the actual machine behavior. In other words, we aim for a *universal schedule* that, for any given machine behavior, has cost close to that of an optimal clairvoyant algorithm.

In this paper we initiate the study of universal scheduling by considering the problem of sequencing jobs on a single unreliable machine to minimize various completion time dependent cost functions. Our main result is an algorithm for computing a universal sequence that is always a constant factor away from an optimal schedule computed by a clairvoyant algorithm. In particular, it computes the same universal sequence independently of the cost function. Thus, we compute a universal solution that has the same small worst case guarantee for any machine behavior and any nondecreasing, nonnegative, differentiable cost function. We complement this by showing that our result is best possible among universal schedules for unreliable machines. In fact, we give a matching lower bound that holds even for any unbounded nonnegative cost function. Furthermore, we study the case in which jobs have individual release dates. Here we provide an almost logarithmic lower bound on the performance of universal schedules (again for any unbounded nonnegative cost function), thus showing a drastic difference with respect to the setting without release dates. Finally, we design a nontrivial algorithm with constant performance for the interesting special case of scheduling jobs with release dates and proportional weights.

Our hope is that our work stimulates the study of universal solutions (with respect to unreliable machine behavior or different cost functions) for other scheduling problems, and, more broadly, the study of more realistic scheduling models. In the rest of this section we introduce our model formally, discuss related work, and explain our contributions in detail.

1.1. The model. We are given a job set J with processing times $p_j \in \mathbb{Q}^+$ and weights $w_j \in \mathbb{Q}^+$ for each job $j \in J$. Using a standard scaling argument, we can assume w.l.o.g. that $w_j \geq 1$ for $j \in J$. The problem is to find a sequence π of jobs to be scheduled on a single machine that minimizes $\sum w_j f(C_j)$ for any nondecreasing, nonnegative, differentiable[1] cost function $f(C_j)$. The jobs are processed in the prefixed order π no matter how the machine may change its processing speed or whether it becomes unavailable. In case of a machine breakdown the currently running job is preempted and will resume processing at any later moment when the machine becomes available again. We may denote our problem as the universal scheduling variant of $1||\sum w_j f(C_j)$, using the standard scheduling notation [12]. We analyze the worst case performance of an algorithm by comparing the solution value it provides with that of an optimal clairvoyant algorithm that knows the machine behavior in advance, and that is even allowed to preempt jobs at any time.

We also consider the problem $1|r_j, pmtn|\sum w_j f(C_j)$ in which each job $j \in J$ has an individual release date $r_j \geq 0$, which is the earliest point in time when it can start processing. In this model, it is necessary to allow job preemption; otherwise no constant performance guarantee is possible, as simple examples show; see Example 2

[1] In fact, the assumption that f is differentiable is slightly stronger than necessary. However, it covers natural cost functions and avoids unnecessary technical details.

in section 4. We allow preemption in the actual scheduling procedure, however, as in the case without release dates, we aim for nonadaptive universal solutions. That is, our solution will still be a total ordering of jobs that we interpret as a priority order. At any point in time we work on the highest priority job that has not yet finished and that has already been released. This procedure is called *preemptive list scheduling* [14, 38]. Note that a newly released job will preempt the job that is currently running if it comes earlier than the current job in the ordering.

1.2. Related work. The concept of *universal* solutions, that perform well for every single input of a superset of possible inputs, has been used already decades ago in different contexts, such as in hashing [5] and routing [42]. The latter is also known as *oblivious routing* and has been studied extensively; see [36] for a state-of-the-art overview. Jia et al. [18] considered universal approximations for TSP, Steiner tree, and set cover problems. All this research falls broadly into the field of robust optimization [4]. The term *robust* is not used consistently in the literature. In particular, the term *robust scheduling* refers mainly to robustness against uncertain processing times; see, e.g., [26, Chap. 7] and [33]. Here, quite strong restrictions on the input or weakened notions of robustness are necessary to guarantee meaningful worst case solutions. We emphasize that our results in this paper are robust in the most conservative, classical notion of robustness originating in Soyster [40], also called *strict robustness* [31], and in this regard we follow the terminology of universal solutions.

Scheduling with limited machine availability, and more generally with arbitrarily varying machine capacity or speed, is a subfield of machine scheduling that has been studied for more than twenty years; see, e.g., the surveys [37, 29, 9, 32]. Different objective functions, stochastic fluctuations, as well as offline problems with known machine availability periods have been investigated. Nevertheless, only a few results are known for min-sum objectives, and none of these deal with release dates. They mainly address the objective of minimizing the total weighted completion time and the extreme case of full machine breakdowns. In the setting when a job can continue processing after a breakdown without restarting it and if all jobs have equal weights, a simple interchange argument shows that sequencing jobs in nondecreasing order of processing times is optimal for $\sum w_j C_j$, as it is in the setting with continuous machine availability [39]. Obviously, this result immediately transfers to the universal setting in which machine breakdowns or changes in processing speeds are not known beforehand. The natural generalization of this algorithm to the weighted setting, that is, scheduling jobs in nonincreasing order of ratios of weight over processing time, is known as *Smith's rule* [39] and yields an optimal schedule on an ideal machine. However, when the machine is unreliable, simple examples show that Smith's rule does not yield a constant performance guarantee. This is true even if there is just a single machine breakdown [28]. In fact, this special problem is weakly NP-hard in both models, where a breakdown causes the restart of a preempted job [1, 30] or does not [28]. Several approximation results have been derived for both models; see [28, 30, 43, 20, 34, 23]. Recently, these results were complemented by fully polynomial-time approximation schemes (FPTAS); see [24, 21, 10]. The special class of instances, in which the processing time of each job is proportional to its weight, has been studied in [43]. The authors showed that scheduling in nonincreasing order of processing times (or weights) yields a 2-approximation for preemptive scheduling without restarts. However, for the general problem with arbitrary job weights, it remained an open question [43] whether a polynomial-time algorithm with constant approximation ratio exists, even without release dates. Even in this restricted case,

the problem is strongly NP-hard [43].

The difficulties in handling full breakdowns are avoided when dealing with nondecreasing machine speed functions. Even for identical release dates and the objective of $\sum w_j C_j$ it is not clear if the problem is NP-hard. Stiller and Wiese [41] show that Smith's rule [39] yields a schedule—a universal schedule, in fact—with a performance guarantee of exactly $(\sqrt{3}+1)/2 < 1.37$. They also show that no universal solution can achieve a performance guarantee of 1.12 or less.

Recently more general objective functions have been considered. For the most general formulation of min-sum problems on a single machine with release dates and preemption, $1|r_j, pmtn| \sum f_j$, where each job may have its individual nondecreasing cost function f_j, Bansal and Pruhs [2] gave a randomized $\mathcal{O}(\log\log(nP))$ approximation, where $P = \max_{j\in J} p_j$. When all jobs have identical release dates, the approximation factor reduces to 16. This result was improved by Cheung and Shmoys [6], who give a deterministic primal-dual $(2+\epsilon)$-approximation. This result can be obtained also on a machine of varying speed. However, it requires the full knowledge of the speed function in advance and does not produce a universal sequence. Very recently, Höhn and Jacobs [16] studied the more restricted problem with a global cost function $1||\sum w_j f(C_j)$ that we consider in this paper. In particular, they analyze the performance of the universal sequence obtained by Smith's rule [39] and give tight guarantees for all convex and all concave functions f.

1.3. Our results. Our main results are algorithms that compute deterministic and randomized universal sequences for jobs without release dates. They output a permutation of the jobs such that scheduling the jobs in this order will yield for any machine behavior and all considered cost functions f simultaneously a solution that remains within multiplicative factor 4 and within multiplicative factor e in expectation of any feasible schedule. Our upper bounds are best possible for universal sequencing on unreliable machines, even when the cost function is fixed. To show this, we establish an interesting connection between our problem and a certain online bidding problem [7]. Furthermore, our algorithms can be adapted such that they have running time polynomial in the input size and $1/\epsilon$, $\epsilon > 0$, at the cost of an ϵ-increase in the performance guarantee. Furthermore, we show that our algorithms can be adapted to solving more general problem instances with certain types of precedence constraints without losing performance quality.

It may seem rather surprising that universal sequences with constant performance guarantee should always exist. In fact, our results immediately answer affirmatively a major question that had remained open in the area of offline scheduling with limited machine availability subject to minimizing $\sum w_j C_j$: whether there exists a constant factor approximation algorithm that schedules jobs (not necessarily universally) on a machine having multiple unavailable periods that are known in advance.

To derive our results, we study the objective of minimizing the total weight of uncompleted jobs at any point in time. First, we show that the performance guarantee for scheduling on an unreliable machine is given directly by a bound on the ratio between the remaining weight of our algorithm and that of an optimal clairvoyant algorithm at every point in time on an ideal machine that is continuously processing at constant speed. Then, we devise an algorithm that computes the job sequence iteratively backwards: in each iteration we find a subset of jobs with largest total processing time subject to a bound on their total weight. The bound is doubled in each iteration. Our approach is related to, but not equivalent to, an algorithm of Hall et al. [14] for online scheduling on ideal machines—the doubling there happens in

the time horizon. Indeed, this type of *doubling* strategy has been applied successfully in the design of algorithms for various problems; the interested reader is referred to the excellent survey of Chrobak and Kenyon-Mathieu [8] for a collection of such examples.

The problem of minimizing the total weight of uncompleted jobs at any time was previously considered by Becchetti et al. [3] in the context of online scheduling to minimize flow time on a single machine; there, a constant approximation algorithm is presented with a worst case bound of 24. Our results imply an improved deterministic $(4+\epsilon)$-approximation for this problem, complemented by a randomized $(e+\epsilon)$-approximation. Furthermore, we show that the same guarantee holds for the setting with release dates. On an ideal machine, this implies a universal solution for minimizing $\sum w_j f(C_j)$ for any cost function f under consideration. Unfortunately, unlike in the case without release dates, this does not translate into the same performance guarantee for an unreliable machine. In fact, when jobs have individual release dates, the problem changes drastically.

In section 4 we show that in the presence of release dates, even if all weights are equal, there are instances for which the ratio between the value of any universal solution and that of an optimal schedule is $\Omega(\log n/\log\log n)$. This lower bound holds for any unbounded nonnegative cost function. Our proof relies on the classical theorem of Erdős and Szekeres [11] on the existence of long increasing/decreasing subsequences of a given sequence of numbers. Motivated by this hardness, we study the class of instances where the processing time of each job is proportional to its weight. We present a nontrivial algorithm and prove a performance guarantee of 5. This algorithm yields directly an $\mathcal{O}(\max_{k\in J}\frac{w_k}{p_k}/\min_{k\in J}\frac{w_k}{p_k})$-approximation when applied to general problem instances. Additionally, we give a lower bound of 3 for all universal solutions in the special case.

2. Preliminaries and key observations. We call a machine ideal if it runs continuously at constant speed. By a standard scaling argument we may assume that this is unit speed. Given an ideal machine and a sequence π, the completion time C_j^π of job j when applying preemptive list scheduling to π is uniquely defined. Its cost is

$$f(C_j^\pi) = \int_0^{C_j^\pi} f'(t)dt = \int_0^\infty \chi_j^\pi(t) f'(t)dt\,,$$

where $\chi_j^\pi(t)$ is the indicator function which is 1 if and only if job j is unfinished by time t in the schedule according to π. For some point in time $t \geq 0$ let $W^\pi(t)$ denote the total weight of jobs that are not yet completed by time t according to π, that is, $W^\pi(t) := \sum_{j\in J}\chi_j^\pi(t)w_j$. Then,

$$\sum_{j\in J} w_j f(C_j^\pi) = \int_0^\infty W^\pi(t) f'(t)dt. \tag{1}$$

Clearly, breaks or fluctuations in the speed of the machine affect the completion times. To describe a particular machine behavior, let $\mu : \mathbb{R}_+ \to \mathbb{R}_+$ be a nondecreasing continuous function, with $\mu(t)$ being the aggregated amount of processing time available on the machine up to time t. We refer to μ as the *machine capacity function*. If the derivative of μ at time t exists, it can be interpreted as the speed of the machine at that point in time. An ideal machine that is processing continuously at unit speed has a machine capacity function $\mu(t) = t,\ t \geq 0$.

For a given capacity function μ, let $S(\pi,\mu)$ denote the single machine schedule when applying preemptive list scheduling to permutation π, and let $C_j^{S(\pi,\mu)}$ denote the completion time of job j in this particular schedule. For some point in time $t \geq 0$, let $W^{S(\pi,\mu)}(t)$ denote the total weight of jobs that are not yet completed by time t in schedule $S(\pi,\mu)$. Then,

$$\sum_{j\in J} w_j f(C_j^{S(\pi,\mu)}) = \int_0^\infty W^{S(\pi,\mu)}(t) f'(t) dt\,.$$

For $t \geq 0$ let $W^{S^*(\mu)}(t) := \min_\pi W^{S(\pi,\mu)}(t)$.

OBSERVATION 1. *For a given machine capacity function μ and a cost function f,*

$$\int_0^\infty W^{S^*(\mu)}(t) f'(t) dt \tag{2}$$

is a lower bound on the objective value $\sum_{j\in J} w_j f(C_j^S)$ of any schedule S.

We aim for a universal sequence of jobs π such that, no matter how the machine behaves, the objective value of the corresponding schedule $S(\pi,\mu)$ is within a constant factor of the optimum.

LEMMA 1. *Let f be a fixed unbounded cost function and $c > 0$. Let π be a sequence of jobs with arbitrary release dates. The objective value $\sum_{j\in J} w_j f(C_j^{S(\pi,\mu)})$ is at most c times the optimum value for every machine capacity function μ if and only if*

$$W^{S(\pi,\mu)}(t) \leq c W^{S^*(\mu)}(t) \qquad \text{for all } t \geq 0. \tag{3}$$

Furthermore, condition (3) *is sufficient for guaranteeing the performance bound c for all (not necessarily unbounded) cost functions.*

Notice that condition (3) is independent of the cost function f. Thus it implies the same performance guarantee for sequence π for all considered cost functions f simultaneously.

Proof. The "if" part is clear for all cost functions, since for any machine capacity function μ we have

$$\sum_{j\in J} w_j f(C_j^{S(\pi,\mu)}) = \int_0^\infty W^{S(\pi,\mu)}(t) f'(t) dt \leq c \int_0^\infty W^{S^*(\mu)}(t) f'(t) dt\,,$$

and Observation 1 applies.

We prove the "only if" part by contradiction. Assume that $W^{S(\pi,\mu)}(t_0) > cW^{S^*(\mu)}(t_0)$ for some t_0 and μ. Consider the following machine capacity function:

$$\bar{\mu}(t) = \begin{cases} \mu(t) & \text{if } t \leq t_0, \\ \mu(t_0) & \text{if } t_0 < t < t_1, \\ \infty & \text{if } t \geq t_1, \end{cases}$$

which equals μ up to time t_0, then remains constant at value $\bar{\mu}(t) = \mu(t_0)$ for the time interval $[t_0, t_1)$, and then increases to infinity[2] such that all remaining jobs finish

[2]The extreme increase from t_1 onwards is not imperative. Any function that guarantees that the remaining processing volume can be finished by t_2 with $f(t_2) \leq (1+\epsilon) f(t_1)$, for $\epsilon > 0$, is sufficient but makes the presentation unnecessarily technical.

at t_1. The objective value for π under the modified machine behavior $\bar{\mu}$, is

$$\sum_{j\in J} w_j f(C_j^{S(\pi,\bar{\mu})}) = \sum_{\substack{j\in J:\\ C_j^{S(\pi,\mu)}\le t_0}} w_j f(C_j^{S(\pi,\mu)}) + f(t_1)\cdot W^{S(\pi,\mu)}(t_0). \tag{4}$$

On the other hand, let π^* be a sequence of jobs with $W^{S(\pi^*,\bar{\mu})}(t_0) = W^{S^*(\bar{\mu})}(t_0)$. Then,

$$\sum_{j\in J} w_j f(C_j^{S(\pi^*,\bar{\mu})}) = \sum_{\substack{j\in J:\\ C_j^{S(\pi^*,\mu)}\le t_0}} w_j f(C_j^{S(\pi^*,\mu)}) + f(t_1)\cdot W^{S^*(\mu)}(t_0). \tag{5}$$

As f is an unbounded, nonnegative, nondecreasing function, $f(t_1)$ tends to infinity for growing t_1. Then, the ratio of (4) and (5) is dominated by $W^{S(\pi,\mu)}(t_0)/W^{S^*(\mu)}(t_0)$, which is larger than c and thus gives a contradiction. □

In the case that all release dates are equal, we can strengthen the previous lemma and show that approximating the min-sum objective value on a machine with unknown processing behavior is equivalent to approximating the total remaining weight at any point in time on an ideal machine with $\mu(t) = t,\ t \ge 0$. To that end, we first observe that scheduling according to sequence π on an ideal machine yields for each j,

$$C_j^{\pi} := \sum_{k:\pi(k)\le\pi(j)} p_k\,.$$

The completion time under machine capacity function μ is

$$C_j^{S(\pi,\mu)} = \min\{t \mid \mu(t) \ge C_j^{\pi}\}.$$

Observation 2. *For any machine capacity function μ and any sequence π of jobs without release dates,*

$$W^{S(\pi,\mu)}(t) = W^{\pi}(\mu(t)) \qquad \text{for all } t \ge 0.$$

For $\mu(t) = t$ let $W^*(t) := W^{S^*(\mu)}(t)$. With Observation 2 we can significantly strengthen the statement of Lemma 1.

Lemma 2. *Let f be a fixed unbounded cost function and $c > 0$. Let π be a sequence of jobs with equal release dates. Then, the objective value $\sum_{j\in J} w_j f(C_j^{S(\pi,\mu)})$ is at most c times the optimum for all machine capacity functions μ if and only if*

$$W^{\pi}(t) \le cW^*(t) \qquad \text{for all } t \ge 0.$$

The weight condition implies the performance guarantee c for all (not necessarily unbounded) cost functions.

Notice that it is crucial for Lemma 2 that all release dates are equal; otherwise, Observation 2 is simply not true. We illustrate this fact by a small example.

Example 1. *At time 0 we release $n-1$ jobs with $p_j = 1$ and $w_j = 1$, for $j = 1,\dots,n-1$. At time $n-1$ we release job n with $p_n = 1$ and $w_n = 2^n$. Consider the sequence $\pi = 1, 2, \dots, n-1, n$ and time $t = n$. It is easy to check that $W^{\pi}(t) = W^*(t)$ for any t. In particular, for $t' = n-1$ we have $W^{\pi}(t') = W^*(t') = w_n$.*

However, when considering an unreliable machine with machine capacity function μ and the corresponding schedule $S(\pi,\mu)$, the situation changes drastically. Let μ

be such that there is a full breakdown at $[(n-2),(n-1)]$ *followed by a second long breakdown beginning at time* n. *At time* $t = n$, *our solution* π *has remaining weight* $W^{S(\pi,\mu)}(t) = w_n$, *which equals* $W^{\pi}(\mu(t))$ *with* $\mu(t) = t'$. *However, an optimal solution executes job* n *in* $[n-1,n)$, *instead of job* $n-1$, *and has remaining weight* $W^{S^*(\mu)}(t) = 1 \neq W^*(\mu(t))$. *Obviously, Observation* 2 *does not hold if jobs have arbitrary release dates.*

3. Universal scheduling without release dates. In what follows we study the universal scheduling problem for jobs without release dates.

3.1. Upper bounds. In what follows we use for a subset of jobs $J' \subseteq J$ the notation $p(J') := \sum_{j\in J'} p_j$ and $w(J') := \sum_{j\in J'} w_j$. Based on key Lemma 2, we aim at approximating the minimum total weight of uncompleted jobs at any point in time on an ideal machine; that is, we approximate the value of $W^*(t)$ for all values of $t \leq p(J)$ for a machine with capacity function $\mu(t) = t,\ t \geq 0$. In our algorithm we do so by solving the problem to find the set of jobs that has maximum total processing time and total weight within a given bound. By sequentially doubling the weight bound, a sequence of job sets is obtained. Jobs in job sets corresponding to smaller weight bounds are to come later in the schedule, breaking ties arbitrarily.

Algorithm Double:

1. For $i \in \{0,1,\ldots,\lceil \log w(J)\rceil\}$, find a subset J_i^* of jobs of maximum total processing time $p(J_i^*)$ such that the total weight satisfies $w(J_i^*) \leq 2^i$. Notice that $J^*_{\lceil \log w(J)\rceil} = J$.
2. Construct a permutation π as follows. Start with an empty sequence of jobs. For $i = \lceil \log w(J)\rceil$ down to 0, append the jobs in $J_i^* \setminus \bigcup_{k=0}^{i-1} J_k^*$ in any order at the end of the sequence.

Theorem 1. *For every scheduling instance,* Double *produces a permutation* π *such that the objective value* $\sum_{j\in J} w_j f(C_j^{S(\pi,\mu)})$ *is less than* 4 *times the optimum for all machine capacity functions* μ *and all considered cost functions* f *simultaneously.*

Proof. Using Lemma 2, it is sufficient to show that $W^{\pi}(t) < 4W^*(t)$ for all $t \geq 0$. Let $t \geq 0$, and let i be minimal such that $p(J_i^*) \geq p(J) - t$. By construction of π, only jobs j in $\bigcup_{k=0}^{i} J_k^*$ have a completion time $C_j^{\pi} > t$. Thus,

$$W^{\pi}(t) \leq \sum_{k=0}^{i} w(J_k^*) \leq \sum_{k=0}^{i} 2^k = 2^{i+1} - 1. \tag{6}$$

In case $i = 0$, the claim is trivially true since $w_j \geq 1$ for any $j \in J$, and thus $W^*(t) = W^{\pi}(t)$. Suppose $i \geq 1$; then by our choice of i, it holds that $p(J_{i-1}^*) < p(J) - t$. Therefore, in any sequence π', the total weight of jobs completing after time t is larger than 2^{i-1}, because otherwise we get a contradiction to the maximality of $p(J_{i-1}^*)$. That is, $W^*(t) > 2^{i-1}$. Together with (6) this concludes the proof. ∎

Notice that the algorithm takes exponential time since finding the subsets of jobs J_i^* is a knapsack problem and, thus, NP-hard [22]. On the other hand, job sets J_i^* can be found in pseudopolynomial time by straightforward dynamic programming. We can reduce the running time to polynomial time at the cost of slightly increasing the performance bound.

We adapt the algorithm by computing, instead of J_i^*, a subset of jobs J_i of total weight $w(J_i) \leq (1+\epsilon)2^i$ and processing time

$$p(J_i) \geq \max\{p(J') \mid J' \subseteq J \text{ and } w(J') \leq 2^i\}.$$

This can be done in time polynomial in the input size and $1/\epsilon$, adapting, e.g., the FPTAS by Ibarra and Kim [17] for knapsack in the following way. In iteration i, let $B := 2^i$ denote the bound on the total weight. For a given $\epsilon > 0$, fix a scaling parameter $K = \epsilon B/n$ and round job weights as well as the weight bound such that $w'_j = \lfloor w_j/K \rfloor$, for any $j \in J$, and $B' = \lfloor B/K \rfloor$. Notice that $w'_j > w_j/K - 1$. Now, we apply a standard dynamic program for knapsack on the modified instance that computes a solution set $J' \subseteq J$ in running time $\mathcal{O}(nB') = \mathcal{O}(n^2/\epsilon)$. The total weight of J' is

$$\sum_{j \in J'} w_j < \sum_{j \in J'} K(w'_j + 1) \leq KB' + Kn \leq B + \epsilon B\,.$$

The optimal solution of the scaled instance has total processing time $p(J') \geq p(J_i^*)$ since the scaling only weakened the capacity constraint.

The subsets J_i obtained in this way are turned into a sequence π' as in Algorithm Double.

Theorem 2. *Let $\epsilon > 0$. For every scheduling instance, we can construct a permutation π in time polynomial in the input size and $1/\epsilon$ such that the objective value $\sum_{j \in J} w_j f(C_j^{S(\pi,\mu)})$ is less than $4+\epsilon$ times the optimum for all machine capacity functions μ and all considered cost functions f simultaneously.*

Proof. Again, by Lemma 2 it is sufficient to prove that $W^{\pi}(t) < (4+\epsilon)W^*(t)$ for all $t \geq 0$. Instead of inequality (6) we get the slightly weaker bound

$$W^{\pi'}(t) \leq \sum_{k=0}^{i} w(J_k) \leq \sum_{k=0}^{i} \left(1 + \frac{\epsilon}{4}\right) 2^k = \left(1 + \frac{\epsilon}{4}\right)(2^{i+1} - 1) < (4+\epsilon)\, 2^{i-1}.$$

Moreover, the lower bound $W^*(t) > 2^{i-1}$ still holds. ∎

We can improve Theorem 1 by adding randomization to our algorithm in a quite standard fashion. Instead of the fixed bound of 2^i on the total weight of job set J_i^* in iteration $i \in \{0, 1, \ldots, \lceil \ln w(J) \rceil\}$ we use the randomly chosen bound Xe^i, where $X = e^Y$ and Y is picked uniformly at random from $[0, 1]$ before the first iteration.

Notice that the same arguments as in Lemma 2 hold for randomized algorithms and their expected values of remaining weight and their min-sum objective values.

Corollary 1. *Let f be a fixed unbounded cost function and $c > 0$. Let π be a random sequence of jobs. Then, the expected objective value $\mathbb{E}[\sum_{j \in J} w_j f(C_j^{S(\pi,\mu)})]$ is at most c times the optimum value for all machine capacity functions μ if and only if $\mathbb{E}[W^{\pi}(t)] \leq cW^*(t)$ for all $t \geq 0$. The weight condition implies the performance guarantee c also for bounded cost functions.*

Theorem 3. *For every scheduling instance, the randomized algorithm produces a random permutation $\pi(X)$ such that $\mathbb{E}[W^{\pi(X)}(t)] < eW^*(t) - 1$ for all $t \geq 0$.*

Proof. Given X and t, let $i \in \mathbb{N}$ be minimal such that $p(J_i^*) \geq p(J) - t$. For $i = 0$ the claim is trivially true. Consider the case $i \geq 1$. By the same arguments as in the proof of Theorem 1, we have $W^*(t) > Xe^{i-1}$, and therefore $i < \lfloor \ln(W^*(t)/X) \rfloor + 1$.

Similar to (6) we have, for any t,

$$\begin{aligned}\mathbb{E}\left[W^{\pi(X)}(t)\right] &\le \mathbb{E}\left[\sum_{k=0}^{i} X e^i\right] = \mathbb{E}\left[X\cdot\frac{e^{i+1}-1}{e-1}\right] < \mathbb{E}\left[X\cdot\frac{e^{\lfloor \ln(W^*(t)/X)\rfloor+2}-1}{e-1}\right]\\ &= \frac{1}{e-1}\cdot\mathbb{E}\left[X\cdot e^{\lfloor \ln(W^*(t)/X)\rfloor-\ln(W^*(t)/X)+2}\cdot\frac{W^*(t)}{X} - X\right]\\ &= \frac{e}{e-1}\cdot W^*(t)\cdot\mathbb{E}\left[e^{\lfloor \ln(W^*(t)/X)\rfloor-\ln(W^*(t)/X)+1}\right] - \frac{1}{e-1}\cdot\mathbb{E}[X]\\ &= \frac{e}{e-1}\cdot W^*(t)\cdot\mathbb{E}\left[e^{\lfloor \ln(W^*(t)/X)\rfloor-\ln(W^*(t)/X)+1}\right] - 1\,.\end{aligned}$$

Here, we used the fact that $\mathbb{E}[X] = \int_0^1 e^x dx = e-1$. With $\ln(W^*(t)/X) = \ln W^*(t) - Y$, we let Z denote the exponent of the exponential function, i.e., $Z := 1-((\ln W^*(t)-Y)-\lfloor \ln W^*(t)-Y\rfloor)$, which is 1 minus the fractional part of $\ln W^*(t)-Y$. Then Z is a random variable distributed like Y uniformly in $[0,1]$. Thus, $\mathbb{E}\left[e^Z\right] = \mathbb{E}[X] = e-1$, and Corollary 1 completes the proof. □

The algorithm can be adapted in the same way as the deterministic algorithm to run in polynomial time; see the proof of Theorem 2. This gives the following improved result.

THEOREM 4. *Let $\epsilon > 0$. For every scheduling instance, randomized* DOUBLE *constructs a permutation π in time that is polynomial in the input size and $1/\epsilon$ such that $\sum_{j\in J} w_j f(C_j^{S(\pi,\mu)})$ is in expectation less than $e+\epsilon$ times the optimum value for all machine capacity functions μ and all considered cost functions f simultaneously.*

3.2. Lower bounds. In this section we show a connection between the performance guarantee for sequencing jobs on a single machine without release dates and an online bidding problem investigated by Chrobak et al. [7]. This allows us to prove tight lower bounds for our problem.

In online bidding we are given a universe $\mathcal{U} = \{1,\dots,n\}$ of possible target values. A bidding algorithm has to submit bids one after another until it gives a number at least as large as the unknown target value $T \in \mathcal{U}$. The cost of the solution is the sum of all submitted bids. An algorithm producing the bid set $\mathcal{B} \subseteq \mathcal{U}$ is said to be α-competitive if

$$\sum_{b\in\mathcal{B}\,:\,b<T} b \;+\; \min_{b\in\mathcal{B}\,:\,b\ge T} b \;\le \alpha\, T \qquad \text{for all } \; T\in\mathcal{U}. \tag{7}$$

Chrobak et al. [7] gave lower bounds of $4-\epsilon$ and $e-\epsilon$, for any $\epsilon > 0$, for deterministic and randomized algorithms, respectively.

THEOREM 5. *For any $\epsilon > 0$ and any unbounded cost function f, there exists an instance of the universal scheduling problem on which the performance guarantee of any deterministic schedule is at least $4-\epsilon$. The performance ratio of any randomized schedule is at least $e-\epsilon$ with respect to an oblivious adversary.*

Proof. Take an instance of the online bidding problem and create the following instance of the scheduling problem: For each $j \in \mathcal{U}$ create job j with weight $w_j = j$ and processing time $p_j = j^j$. Consider any permutation π of the jobs in $\mathcal{U}$. For any $j \in \mathcal{U}$, let $k(j)$ be the largest index such that $\pi_{k(j)} \ge j$. Since the total processing time of jobs $\{\pi_k \,|\, k = k(j)+1,\dots,n\}$ is at most $\sum_{i=1}^{j-1} p_i < p_j$, at time $t = p(\mathcal{U}) - p_j$ we have $W^\pi(t) = \sum_{k=k(j)}^{n} w_{\pi_k}$, while $W^*(t) = w_j$. If sequence π yields a performance

guarantee of α, then it holds by Lemma 2 that

$$\sum_{k=k(j)}^{n} \pi_k \le \alpha\, j \qquad \text{for all } j \in \mathcal{U}. \tag{8}$$

From sequence π we extract another sequence of jobs:

$$\begin{aligned} \mathcal{W}_1 &= \pi_n, \\ \mathcal{W}_k &= \operatorname*{argmax}_{i \in \mathcal{U}} \left\{ \pi^{-1}(i) \mid i > \mathcal{W}_{k-1} \right\}. \end{aligned}$$

Define the bid set $\mathcal{W} = \{\mathcal{W}_1, \mathcal{W}_2, \ldots\}$. By definition $\mathcal{W}_{i+1} > \mathcal{W}_i$, and all j with $\pi^{-1}(\mathcal{W}_{i+1}) < \pi^{-1}(j) < \pi^{-1}(\mathcal{W}_i)$ have weight less than $\mathcal{W}_i$. Therefore, for all $j \in \mathcal{U}$ we have $\{i \in \mathcal{W} \mid i < j\} \cup \min\{i \in \mathcal{W} \mid i \ge j\} \subset \{\pi_{k(j)}, \ldots, \pi_n\}$. Hence, if π achieves a performance guarantee of α, then

$$\sum_{i \in \mathcal{W} \,:\, i<j} i \;+\; \min_{i \in \mathcal{W} \,:\, i \ge j} i \;\le\; \sum_{k=k(j)}^{n} \pi_k \;\le\; \alpha\, j \qquad \text{for all } j \in \mathcal{U}; \tag{9}$$

that is, the bid set $\mathcal{W}$ induced by the sequence π must be α-competitive. Since there is a lower bound of $4 - \epsilon$ for the competitiveness of deterministic strategies for online bidding, the same bound follows for the performance guarantee of deterministic universal schedules.

The same approach yields the lower bound for randomized strategies. In this case, for online bidding, $\mathcal{B}$ is a probability distribution over all subsets of $\mathcal{U}$. Analogous to (7), $\mathcal{B}$ is said to be α-competitive if

$$\mathrm{E}\left[\sum_{b \in \mathcal{B} \,:\, b<T} b + \min_{b \in \mathcal{B} \,:\, b \ge T} b \right] \le \alpha\, T \qquad \text{for all } T \in \mathcal{U}. \tag{10}$$

In the scheduling setting, analogous to (8), if the random permutation π yields performance guarantee α, then

$$\mathrm{E}\left[\sum_{k=k(j)}^{n} \pi_k \right] \le \alpha\, j \qquad \text{for all } j \in \mathcal{U}. \tag{11}$$

In the same way a single schedule induces a single bid set, a random sequence of jobs π induces a probability distribution $\mathcal{W}$ over bid sets, and if π has performance guarantee α, then

$$\mathrm{E}\left[\sum_{i \in \mathcal{W} \,:\, i<j} i + \min_{i \in \mathcal{W} \,:\, i \ge j} i \right] \le \mathrm{E}\left[\sum_{k=k(j)}^{n} \pi_k \right] \le \alpha\, j \qquad \text{for all } j \in \mathcal{U}. \tag{12}$$

The lower bound of $e - \epsilon$ for randomized strategies for online bidding implies the same lower bound for the performance guarantee of randomized universal schedules. ∎

3.3. Universal scheduling with precedence constraints. A natural generalization of the universal sequencing problem requires that jobs must be sequenced in compliance with given precedence constraints. These constraints define a partial order $(J, \prec)$ on the set of jobs J.

Clearly, the lower bounds above hold also in the more general setting. Furthermore, we can adapt our algorithm to a certain extent. To handle precedence constraints we need to adapt the knapsack-related subroutine of our algorithm to the problem with a given partial order of jobs. This subproblem coincides with the *partially ordered knapsack problem* (POK) which is strongly NP-hard [19] and also hard to approximate [13]. On the positive side, FPTASes exist for several POK problems with special partial orders, including directed out-trees, two-dimensional orders, and the complement of chordal bipartite orders [19, 25].

THEOREM 6. *Let $\epsilon > 0$. Consider the universal sequencing problem with precedence constraints $(J, \prec)$. If there is an FPTAS for POK for partial orders of the same type, then for any $\epsilon > 0$ we can construct a permutation π respecting $(J, \prec)$ in time polynomial in the input size and $1/\epsilon$ such that the objective value $\sum_{j \in J} w_j f(C_j^{S(\pi,\mu)})$ is less than $4 + \epsilon$ times the optimum for all machine capacity functions μ and all cost functions f simultaneously. A randomized algorithm finds a sequence with expected objective value bounded by $e + \epsilon$ times the optimum value in the same running time.*

Proof. We make use of the following trivial observation: Let $(N, \prec)$ be a partial order and $(N, \prec')$ be the reverse partial order. Then, given a linear extension of $(N, \prec)$, the reverse of this ordering is a feasible linear extension of $(N, \prec')$.

Now consider the universal sequencing problem with a given partial order $(J, \prec)$ and its reverse order $(J, \prec')$. We apply a slightly modified version of DOUBLE. To compute subsets J_i^* with bounded total weight and maximal processing time, we use the FPTAS for the corresponding special case of POK for $(J, \prec')$. Obviously, the sequence of sets $0, 1, 2 \ldots$ respects the given partial order $(J, \prec')$. The algorithm appends the sets in the reverse order, and therefore the final sequence is a linear extension of $(J, \prec)$ if the jobs of each set are appended accordingly. Now, the analysis given in section 3.1 applies directly. □

4. Universal scheduling with release dates. In this section we study the universal scheduling problem for jobs with arbitrary release dates. As mentioned in the introduction, we cannot hope for a universal sequence of jobs that, when processed nonpreemptively in exactly this order, yields a constant performance guarantee for all machine capacity functions μ. This is true not only when competing with an optimal solution that may preempt jobs, but also when the optimum is not allowed to preempt. We visualize this by the following simple example.

EXAMPLE 2. *The instance has two jobs: a long job with $r_1 = 0$, $p_1 = L$, and $w_1 = \epsilon$ and a short job with $r_2 = \epsilon$ and $w_2 = p_2 = 1$. By Lemma 1, any algorithm must start working on the first job immediately at time 0, for otherwise the jobs cannot be completed by time $L + 1$. On the other hand, if we start working on the first job at time 0 and there is no machine breakdown, the solution has cost $L(1 + \epsilon) + 1$. The optimal cost is $L\epsilon + (1 + \epsilon)^2$, when the optimal solution is not allowed to preempt, and $L\epsilon + 1 + 2\epsilon$ otherwise. In both cases, when ϵ is small, the ratio of the costs is growing linearly in L and thus arbitrarily large.*

Therefore, we allow preemption in the actual scheduling procedure, although, as in the case without release dates, we aim for nonadaptive universal solutions. Thus, our universal sequence specifies a priority order of jobs which is executed by a preemptive list scheduling procedure: At any point in time we work on the job of highest priority that has not finished yet and that has already been released.

Algorithm DOUBLE, which aims to minimize the total remaining weight, can be adapted to the setting with release dates with the same performance guarantee, as we show in section 4.1. This implies by Lemma 1 that, on an ideal machine, one se-

quence of jobs yields the same performance guarantee for all considered cost functions simultaneously. Unfortunately, this does not hold on unreliable machines. In contrast to the setting without release dates, approximation ratios on an ideal machine do not translate directly into a performance guarantee of the universal sequencing strategy for an unreliable machine; see section 2. In fact, universal scheduling with release dates cannot be approximated within a constant ratio, as we show in section 4.2. In section 4.3, we consider the special case in which jobs have proportional weights. We provide a nontrivial algorithm with small constant performance guarantee accompanied with lower bounds. This algorithm yields an $\mathcal{O}(\max_{k\in J} \frac{w_k}{p_k} / \min_{k\in J} \frac{w_k}{p_k})$-approximate universal solution when applied to the general scheduling problem with arbitrary weights.

4.1. Universal schedules on an ideal machine. Consider scheduling on an ideal machine with constant speed.

THEOREM 7. *There is a deterministic polynomial-time algorithm that produces for any set of jobs J with individual release dates a sequence π such that the total cost $\sum_{j\in J} w_j f(C_j^{S(\pi)})$ when applying preemptive list scheduling according to π is less than $4+\epsilon$ times the optimum value for any cost function f. A randomized algorithm yields the performance guarantee $e+\epsilon$.*

By Lemma 1 and its natural extension to randomized algorithms, it is sufficient to give an algorithm that yields the desired performance guarantee for minimizing the total remaining weight at any time. In the remainder of this section we discuss such an algorithm.

The following algorithm is an adaptation of Algorithm DOUBLE to the setting with release dates and preemptive list scheduling.

ALGORITHM DOUBLE-R:

1. Compute the earliest possible completion time T.
2. For $i \in \{0, 1, \ldots, \lceil \log w(J) \rceil\}$, find a feasible schedule S_i for J and a maximum value Δ_i such that the total weight of late jobs, $J_i^* = \{j \in J \mid C_j^{S_i} > T - \Delta_i\}$, completing after due date $T - \Delta_i$, satisfies $w(J_i^*) \le 2^i$. Notice that $J^*_{\lceil \log w(J) \rceil} = J$.
3. Construct a permutation π as follows. Start with an empty sequence of jobs. For $i = \lceil \log w(J) \rceil$ down to 0, append the jobs in $J_i^* \setminus \bigcup_{k=0}^{i-1} J_k^*$ in any order at the end of the sequence.

As in DOUBLE, we want to find for any weight bound 2^i a subset of jobs that, when being placed last in the priority list, covers the longest continuous processing period at the end of a feasible preemptive schedule. In contrast to the setting without release dates, we cannot simply consider the total processing time of the last jobs, because some of these jobs may have been processed earlier in the actual schedule and have less processing remaining at the end of the schedule. We take this into account in step 2 of DOUBLE-R by maximizing Δ_i, which corresponds to maximizing the total processing time of the last jobs in step 2 of DOUBLE. In fact, DOUBLE-R computes the same permutation of jobs as DOUBLE when all release dates are zero.

Lawler [27] provides a pseudopolynomial-time algorithm for preemptively scheduling jobs with release dates and due dates on a single machine to minimize the total weight of late jobs. Hence, we can solve step 2 in DOUBLE-R by binary search over the parameter $\Delta_i \in [\Delta_{i-1}, T]$ using Lawler's algorithm.

THEOREM 8. *For every scheduling instance,* DOUBLE-R *produces a sequence π such that $W^{S(\pi)}(t) < 4W^{S^*}(t) - 1$ for all $t \ge 0$ on an ideal machine.*

Proof. Given a schedule $S := S(\pi)$ and some $t \geq 0$, let i be minimal such that $T - \Delta_i \leq t$. By construction, only jobs j in $\bigcup_{k=0}^{i} J_k^*$ have a completion time $C_j^S > t$. To see that, consider a job $j \in J \setminus \bigcup_{k=0}^{i} J_k^*$. By definition, there exists a feasible schedule S_i such that for all $k \in J \setminus J_i^*$ (including j) we find $C_k^{S_i} \leq T - \Delta_i$. Applying preemptive list scheduling allows only jobs in $J \setminus \bigcup_{k=0}^{i} J_k^*$ to be considered earlier than j. Since there exists a feasible schedule such that all those jobs can be completed by $T - \Delta_i$, preemptive list scheduling will also find such a schedule.

Therefore,

$$W^S(t) \leq \sum_{k=0}^{i} w(J_k^*) \leq \sum_{k=0}^{i} 2^k = 2^{i+1} - 1. \tag{13}$$

In case $i = 0$, the claim is trivially true since $w_j \geq 1$ for any $j \in J$, and thus $W^{S^*}(t) = W^S(t)$. Suppose $i \geq 1$; then by our choice of i, it holds that $T - \Delta_{i-1} > t$. In any schedule S', the total weight of jobs completing after time t is larger than 2^{i-1}, because otherwise we get a contradiction to the maximality of Δ_{i-1}. Hence, $W^{S^*}(t) > 2^{i-1}$. Together with (13), this concludes the proof. ∎

As in section 3, randomization on the choice of weight bounds improves the performance.

COROLLARY 2. *For every scheduling instance, the randomized version of* DOUBLE-R *produces a random sequence* $\pi(X)$ *such that* $\mathbb{E}[\, W^{S(\pi(X))}(t)\,] < eW^{S^*}(t) - 1$ *for all* $t \geq 0$ *on an ideal machine.*

Lawler's algorithm used in step 2 runs in pseudopolynomial time. However, Pruhs and Woeginger [35] turned it into an FPTAS. Using this algorithm, we get a running time polynomial in the input size and $1/\epsilon$ and slightly increased cost.

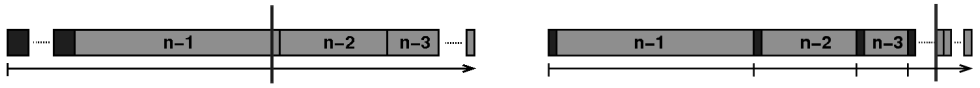
THEOREM 9. *Let* $0 < \epsilon$. *For every scheduling instance with release dates,* DOUBLE-R *constructs a permutation* π *in time that is polynomial in the input size and* $1/\epsilon$ *such that* $W^{S(\pi)}(t) < (4+\epsilon)W^{S^*}(t)$ *for all* $t \geq 0$ *on an ideal machine. A randomized variant yields a random permutation* $\pi(X)$ *with* $\mathbb{E}[\, W^{S(\pi(X))}(t)\,] < (e+\epsilon)W^{S^*}(t)$ *for all* $t \geq 0$.

4.2. Lower bound. We give a lower bound on the performance guarantee of universal schedules for jobs with arbitrary release dates and unbounded cost functions.

THEOREM 10. *There exists an instance with* n *jobs with release dates, where the performance guarantee of any universal schedule is* $\Omega(\log n / \log\log n)$ *for any unbounded cost function* f, *even if all weights are equal.*

In our lower bound instance each job j has weight $w_j = 1$, $j = 0, 1, \ldots, n-1$. The processing times of the jobs form a geometric series $p_j = 2^j$, $j = 0, 1, \ldots, n-1$, and they are released in reversed order $r_j = \sum_{i>j}^{n-1} 2^i = \sum_{i>j} p_i$, $j = 0, 1, \ldots, n-1$. On an ideal machine, each job can start running at its release date, and it will have finished processing by the release time of the next job. Therefore, preemptive list scheduling produces in this case the same schedule for any priority order.

To get some intuition, we briefly discuss two extreme cases, which are visualized in Figure 1. First, consider the universal job sequence $\pi_a = n-1, n-2, \ldots, 0$. If the machine is unavailable from time 0 on until all jobs are released, we arrive in a setting without release dates. Suppose that the machine becomes available at time 2^n and then works at full speed. Then at time $2^n + p_{n-1} - 1 = 2^n + 2^{n-1} - 1$ the universal schedule still has n uncompleted jobs, whereas an optimal solution has completed $0, 1, \ldots, n-2$, yielding by Lemma 1 a performance guarantee of $\Omega(n)$.



(a) Worst case for sequence π_a. (b) Worst case for sequence π_b.

FIG. 1. *For the sequence* $\pi_a = n-1, n-2, \ldots, 0$ *and its reverse counterpart* $\pi_b = 0, 1, \ldots, n-1$, *we give two breakdown patterns* (a) *and* (b) *such that each sequence is optimal for one of them and arbitrarily bad for the other one.*

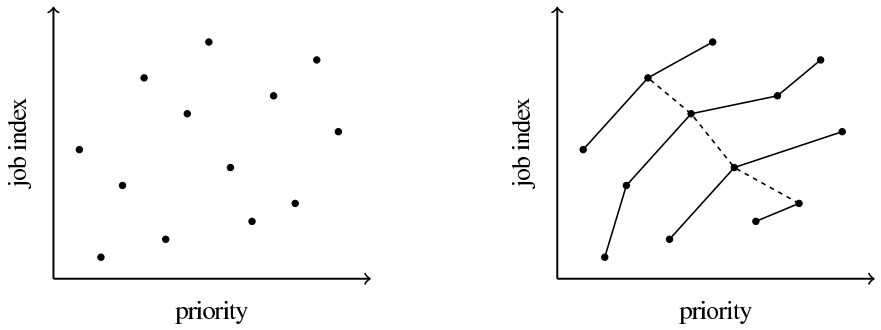


FIG. 2. *Two-dimensional visualization of a universal sequence. Left: Each point represents one job; x-coordinates correspond to positions in the sequence, and y-coordinates correspond to job indices. Right: The points have been decomposed into four increasing subsequences. There is a decreasing subsequence of length* 4.

On the other hand, the sequence $\pi_b = 0, 1, \ldots, n-1$ sees adversarial breakdowns in the intervals $[r_j, r_j + \epsilon]$, $j = 0, 1, \ldots, n-1$, resulting in n uncompleted jobs at time $2^n - 1$, whereas the schedule following the sequence π_a has job 0 as the only uncompleted job. Hence a long breakdown of the machine starting at this time yields again a performance guarantee $\Omega(n)$.

In the following we show that there is no way to "interpolate" between these two extremes. To that end, we rely on a classic theorem of Erdős and Szekeres [11] or, more precisely, on Hammersley's proof [15] of this result.

LEMMA 3 (Hammersley [15]). *Any given sequence of* n *distinct numbers* $\pi_1, \pi_2, \ldots, \pi_n$ *can be decomposed into* k *increasing subsequences* $\ell_1, \ell_2, \ldots, \ell_k$, *for some* $k \geq 1$, *with the following properties:*

(i) *There exists a decreasing subsequence of length* k.
(ii) *If* π_j *belongs to* ℓ_i, *then for all* $j' > j$, *if* $\pi_{j'} < \pi_j$, *then* $\pi_{j'}$ *belongs to* $\ell_{i'}$ *with* $i' < i$.

The idea is now to view any universal sequence as a permutation of job indices $\{0, 1, \ldots, n-1\}$ and to decompose it into k increasing subsequences according to Lemma 3. Figure 2 provides a two-dimensional visualization. The following two lemmas give lower bounds on the performance guarantee of a universal sequence as functions of the lengths of increasing and decreasing subsequences in the decomposition. The proofs are based on breakdown patterns, similar to the ones in the two extreme cases above.

Let $|\ell|$ denote the length of a sequence ℓ.

LEMMA 4. *The performance guarantee of a universal schedule that has a decreasing subsequence* ℓ *is at least* $|\ell|$.

Proof. Let j be the first job in ℓ, that is, the job with highest priority and smallest

release date in ℓ. The machine has breakdowns $[r_j, r_0]$ and $[r_0 + 2^j - 1, L]$ for large L. By time r_0 all jobs are released. Then, $2^j - 1$ time units later, at the start of the second breakdown, all jobs in ℓ belong to the set of jobs uncompleted by the universal schedule, whereas an optimal solution can complete all jobs except j. Choosing L large enough implies the lemma. □

LEMMA 5. *Let $\ell_1, \ell_2, \ldots, \ell_k$ be the decomposition of a universal schedule into increasing subsequences as described in Lemma 3. Then for all $i = 1, \ldots, k$ the performance guarantee is at least $\frac{|\ell_i|+|\ell_{i+1}|+\cdots+|\ell_k|}{1+|\ell_{i+1}|+\cdots+|\ell_k|}$.*

Proof. For each job j in ℓ_i there is a breakdown $[r_j, r_j + \epsilon]$. For each job j in $\ell_{i+1}, \ldots, \ell_k$ there is a breakdown $[r_j, r_j + p_j] = [r_j, r_j + 2^j]$.

By induction, at any release time t the following is true for all jobs that were released before t: a job has finished if it belongs to subsequences $\ell_1, \ldots, \ell_{i-1}$, has left ϵ units of processing if it belongs to ℓ_i, and has not been processing at all if it belongs to $\ell_{i+1}, \ldots, \ell_k$. Consider the first point in time, r_{j_0}, when a job j_0 of subsequences $\ell_i, \ldots, \ell_k$ is released. This is the first time when a machine breakdown is incurred. All jobs j' that had been released earlier belong to subsequences $\ell_1, \ldots, \ell_{i-1}$ and have completed. Then j_0 has highest priority and can start whenever the machine is available before the next release date. With the given breakdown pattern, j_0 can process $p_j - \epsilon$ units if it belongs to ℓ_i, and it does not run at all if it belongs to $\ell_{i+1}, \ldots, \ell_k$. Now consider some release time t. Let j from subsequence ℓ_a be the last job released before t. Property (ii) of Lemma 3 states that all jobs with higher priority than j and smaller release date (i.e., larger index) than j belong to some sequence ℓ_b with $b < a$. Combined with the inductive hypothesis, this implies for $a \in \{1, \ldots, i\}$ that j has highest priority because all available higher priority jobs have finished processing. With the defined breakdown patterns, j fully completes processing by time t if it belongs to $\ell_1, \ldots, \ell_{i-1}$, whereas ϵ time units remain if $a = i$. If $a \in \{i + 1, \ldots, k\}$, then the machine is unavailable until time t, which proves the claim.

As a consequence, at time $2^n - 1$ the universal schedule has all jobs in ℓ_i and all jobs in $\ell_{i+1}, \ldots, \ell_k$ uncompleted, whereas a schedule exists that leaves the last job of ℓ_i and all jobs in $\ell_{i+1}, \ldots, \ell_k$ uncompleted. Therefore, a breakdown $[2^n - 1, L]$ for L large enough implies the lemma. □

Proof of Theorem 10. Consider an arbitrary universal scheduling solution and its decomposition into increasing subsequences $\ell_1, \ldots, \ell_k$ as in Lemma 3, and let α be its performance guarantee.

Using Lemma 5, one can easily prove by backward induction that $|\ell_i| \leq \alpha^{k-i+1}$. Since $\ell_1, \ldots, \ell_k$ is a partition of all jobs, we have

$$n = \sum_{i=1}^{k} |\ell_i| \leq \sum_{i=1}^{k} \alpha^{k-i+1} \leq \alpha^{k+1}.$$

By Lemma 4, it follows that $k \leq \alpha$. Therefore $\log n = \mathcal{O}(\alpha \log \alpha)$ and $\alpha = \Omega(\frac{\log n}{\log \log n})$. □

4.3. Jobs with proportional weights. Motivated by the negative result in the previous section, we turn our attention to the special case where jobs have weights that are proportional to their processing times; that is, there exists a fixed $\gamma \in \mathbb{Q}^+$ such that $w_j = \gamma p_j$ for all $j \in J$. Using a standard scaling argument, we can assume w.l.o.g. that $p_j = w_j$ for all j. We provide an algorithm with performance guarantee 5 and prove a lower bound of 3 on the performance guarantee of any universal

scheduling algorithm. This algorithm applied to the unconstrained problem version yields an $\mathcal{O}(\max_{k\in J} \frac{w_k}{p_k} / \min_{k\in J} \frac{w_k}{p_k})$-approximate universal solution. In this case, we ignore the actual weights and assume them to equal processing times.

4.3.1. Upper bounds. We propose and analyze the following algorithm.

ALGORITHM SORTCLASS:

1. Partition the set of jobs into $z := \lceil \log \max_{j\in J} w_j \rceil$ classes such that j belongs to class J_i, for $i \in 1, 2, \ldots, z$, if and only if $p_j \in (2^{i-1}, 2^i]$.
2. Construct a permutation π as follows. Start with an empty sequence of jobs. For $i = z$ down to 1, append the jobs of class J_i in nondecreasing order of release dates at the end of π.

THEOREM 11. *For every scheduling instance,* SORTCLASS *produces a permutation π such that the objective value $\sum_{j\in J} w_j f(C_j^{S(\pi,\mu)})$ is at most 5 times the optimum for all machine capacity functions μ and all considered cost functions f simultaneously. This bound is tight.*

Proof. Let π be the job sequence computed by SORTCLASS. We can assume w.l.o.g. that the schedule $S(\pi, \mu)$ obtained from π for some machine capacity function μ has no idle time. The reason is that idle time can occur only when no unfinished job is available, because we run preemptive list scheduling. Thus, the schedule $S(\pi, \mu')$ for the machine behavior μ' that equals μ with additional breakdowns during the idle time periods in $S(\pi, \mu)$ is free of idle times and has exactly the same cost as $S(\pi, \mu)$. By Lemma 1, it is sufficient to prove

$$W^{S(\pi,\mu)}(t) \;\le\; 5W^{S^*(\mu)}(t) \quad \text{for all } t > 0, \text{ for all } \mu. \tag{14}$$

Take any time t, and let $j \in J_i$ be the job being processed at time t according to the schedule $S(\pi, \mu)$. We say that a job other than job j is *in the stack* at time t if it was processed for a positive amount of time but has not completed before t. The algorithm needs to complete all jobs in the stack, job j, and jobs that did not start before t, which have a total weight of at most $p(J) - \mu(t)$, the amount of remaining processing time at time t to be done by the algorithm.

Since jobs within a class are ordered by release times, there is at most one job per class in the stack at any point in time. Since jobs in higher classes have higher priority and job $j \in J_i$ is processed at time t, there are no jobs in $J_{i+1}, \ldots, J_z$ in the stack at time t. Thus the weight of the jobs in the stack together with the weight of job j is at most $\sum_{k=1}^{i} 2^k = 2^{i+1} - 1$. Hence,

$$W^{S(\pi,\mu)}(t) \;< 2^{i+1} + p(J) - \mu(t)\,. \tag{15}$$

A first obvious lower bound on the remaining weight of any schedule at time t is

$$W^{S^*(\mu)}(t) \;\ge p(J) - \mu(t)\,. \tag{16}$$

For another lower bound, let t' be the last time before t in which the machine is available but either it is idle or a job of a class $J_{i'}$ with $i' < i$ is being processed. Note that t' is well defined. By definition, all jobs processed during the time interval $[t', t]$ are in classes with index at least i, but also they are released in the interval $[t', t]$ since at t' a job of a lower class was processed or the machine was idle. Since at time t at least one of these jobs is unfinished in $S(\pi, \mu)$, even though the machine

continuously processed only those jobs, no algorithm can complete all these jobs. Thus, at time t, an optimal schedule also still needs to complete at least one job with weight at least 2^{i-1}:

$$W^{S^*(\mu)}(t) \geq 2^{i-1}. \tag{17}$$

Combining (15), (16), and (17) yields (14) and thus the upper bound of the theorem.

To see that the analysis is tight, consider the following instance with $k+3$ jobs. We have k main jobs of geometrically increasing weights and processing times $w_j = p_j = 2^{j-1}$ and release dates $r_j = \sum_{i<j} p_i$ for $j = 1, \ldots, k$. We have three additional jobs a, b, and c with $w_a = p_a = 2^{k-1} + k\epsilon$, $w_b = p_b = 2^k$, and $w_c = p_c = 2^{k-1} - \epsilon$ and release dates $r_a = \sum_{i \leq k} p_k, r_b = r_a + \epsilon$ and $r_c = r_b + \epsilon$ for some $0 < \epsilon < 1$.

SortClass transforms the sequence $\pi = 1, 2, \ldots, k-1, k, a, b, c$ of release date order into $a, b, k, c, k-1, k-2, \ldots, 2, 1$. To show the lower bound, we give a break of length ϵ at each release date r_j for $j = 1, 2, \ldots, k$. Thus, the jobs $j = 1, \ldots, k$ start processing one after another but get preempted an ϵ time unit before finishing because of the release of a higher priority job. At time r_a, job a starts processing and finishes without interruption; it is followed by job b, which gets interrupted at time $t = r_a + p_a + p_b - \epsilon$ by a huge breakdown. At this time, the only job that has completed in this schedule is job a. Thus, the remaining weight of unfinished jobs at time t is $5 \cdot 2^{k-1} - 1 - \epsilon$. In contrast, scheduling the sequence $1, 2, \ldots, k-1, k, b, c, a$ under such machine breakdowns leaves only job a with weight $2^{k-1} + k\epsilon$ unfinished at time t. The lower bound of 5 follows immediately. ∎

We may apply Algorithm SortClass also to general instances with arbitrary job weights: we simply ignore the actual weights and assume them to equal processing times. In the case that $w_j \geq p_j$ for all $j \in J$, we underestimate the cost of a schedule by at most a factor of $\max_{k \in J} \frac{w_k}{p_k}$. In the case that $w_j < p_j$ for some $j \in J$, we first multiply all weights by the factor $\max_{k \in J} \frac{p_k}{w_k} = 1/\min_{k \in J} \frac{w_k}{p_k}$ to guarantee that $w_j \geq p_j$ for all $j \in J$. Then we lose in total a factor of at most $\max_{k \in J} \frac{w_k}{p_k} / \min_{k \in J} \frac{w_k}{p_k}$. Thus, Theorem 11 directly gives the following guarantee.

Corollary 3. *The performance guarantee of* SortClass *for universal scheduling is* $\mathcal{O}(\frac{\max_{k \in J} w_k/p_k}{\min_{k \in J} w_k/p_k})$.

4.3.2. Lower bound. We complement this result by a lower bound of 3 on the performance guarantee of any universal scheduling algorithm for the proportional weight case.

Theorem 12. *For any unbounded cost function f there exists an instance with n jobs with release dates and $w_j = p_j$, for all $j \in J$, where the performance guarantee of any universal sequence is at least* 3.

Proof. Assume by contradiction that there is an algorithm that finds a universal sequence with performance guarantee strictly smaller than c such that $c < 3$. We define a sequence of jobs that must be scheduled by any universal c-approximate algorithm in a specific order for one specific machine breakdown scenario. But this permutation will then be worse than c-approximate for another breakdown scenario.

We start by defining and analyzing a number sequence, which we will use later as the sequence of processing times of the desired scheduling instance. Let $a_1 = 1$, $a_2 = c$, and for $i \geq 3$, $a_i = (c+1)(a_{i-1} - a_{i-2})$. Let $S_i = \sum_{j=1}^{i} a_j$. Note that an alternative definition for a_i, $i \geq 3$, is $a_i = c \cdot a_{i-1} - S_{i-2}$. Indeed, using the definition

of the sequence yields

$$\begin{aligned} a_i + S_{i-1} - (c+1) = S_i - (c+1) = S_i - a_1 - a_2 &= \sum_{j=3}^{i} a_j = \sum_{j=3}^{i} (c+1)(a_{j-1} - a_{j-2}) \\ &= (c+1)\sum_{j=2}^{i-1} a_j - (c+1)\sum_{j=1}^{i-2} a_j = (c+1)(a_{i-1} - a_1) \\ &= (c+1)(a_{i-1} - 1), \end{aligned}$$

which gives $a_i = (c+1)a_{i-1} - S_{i-1} = ca_{i-1} - S_{i-2}$. When $S_0 = 0$, the alternative definition holds for $i = 2$ as well.

Note that by letting $c' = c+1$, we get exactly the well-known sequence defined by the recurrence $b_i = c'(b_{i-1} - b_{i-2})$; see, e.g., [7]. For this sequence, it is known that since $c' < 4$, no matter what the initial conditions are exactly (but $0 < b_1 < b_2$), there exists an integer n such that $b_1 < b_2 < \cdots < b_n$, while $0 < b_{n+1} \le b_n$. Therefore, this property holds for the sequence a_i, and we use this value of n in our proof.

We consider a set of n jobs as follows. For job j, $p_j = w_j = a_j$. The release time of job j is $r_j = S_{j-1} - (j-1)\epsilon$, where $\epsilon < \frac{1}{n}$. Let $T = S_n$, which is the total size of all jobs.

By our assumption, the performance guarantee of the algorithm is smaller than c, and we next characterize the permutation which the algorithm must use.

Consider the time $T - \epsilon$. If the machine works continuously until this time, and since an optimal solution can run the jobs as follows—job 1 during the time $[0, S_1 - \epsilon]$ and $[T-\epsilon, T]$, and job $k > 1$ during the time $[S_{k-1} - \epsilon, S_k - \epsilon]$—then it must be the case that the total size of the jobs which the algorithm did not complete until time $T - \epsilon$ is less than ca_1. Since $c = a_2 < a_3 < \cdots < a_n$, the only such job can be job 1. We next prove by induction that if the last jobs in the permutation are jobs $k-1, k-2, \ldots, 1$, then the job before $k-1$ must be job k (for any $2 \le k \le n-1$). Consider the time $T - k\epsilon$ and an optimal schedule which runs job j, where $j < k$, during the time slot $[S_{j-1}, S_j]$, the job k during the time slots $[S_{k-1}, S_k - k\epsilon]$ and $[T - k\epsilon, T]$, and any job $j > k$ during the time slot $[S_{j-1} - k\epsilon, S_j - k\epsilon]$. Since the jobs $1, 2, \ldots, k-1$ have a lower priority than jobs $k, k+1, \ldots, n$, and among the set $\{1, 2, \ldots, k-1\}$ jobs of lower indices have a lower priority, job $j-1$ is preempted upon the release of job j, for $j = 2, 3, \ldots, k-1$, leaving a part of length ϵ of each such job incomplete. This gives a total of $(k-1)\epsilon$, which means that there is an additional incomplete job. However, the total size of the incomplete jobs at time $T - k\epsilon$ must be smaller than $c \cdot a_k$. Therefore, since $a_{k+1} + S_{k-1} = c \cdot a_k$ and $a_{k+1} = \min\{a_{k+1}, a_{k+2}, \ldots, a_n\}$, the only additional incomplete job must be job k.

Since the $n-1$ last jobs in the permutation must be $n-1, n-2, \ldots, 1$, the first job in the permutation is job n.

Consider now the time $T - n\epsilon$. An optimal solution can run each job j during the time slot $[S_{j-1}, S_j]$, and thus at time $S_n - n\epsilon > S_{n-1}$ it runs job n. However, the algorithm preempts each job in favor of the newly released job, and hence none of the jobs is completed by this time. This gives a ratio of $\frac{S_n}{a_n}$. Since $a_{n+1} < a_n$, we have $a_{n+1} = (c+1)(a_n - a_{n-1}) \le a_n$ or $c \cdot a_n \le (c+1)a_{n-1}$. Moreover, $a_n = ca_{n-1} - S_{n-2}$, so $\frac{S_n}{a_n} = \frac{a_n + a_{n-1} + S_{n-2}}{a_n} = \frac{(c+1)a_{n-1}}{a_n} \ge c$, which is a contradiction. ∎

5. Concluding remarks. In section 4 we have shown that the performance of universal scheduling algorithms may deteriorate drastically when release dates are added to the universal scheduling problem with a min-sum objective function. Other

generalizations do not admit any (even exponential-time) algorithm with bounded performance guarantee: If a nonadaptive algorithm cannot guarantee to finish within the minimum makespan, then an adversary creates an arbitrarily long breakdown at the moment that an optimal schedule has completed all jobs. Examples of such variations are the problem with two or more machines instead of a single machine, or the problem in which preempting or resuming a job requires (even the slightest amount of) extra work. In these problem variants, the worst case examples require rather artificial machine behaviors in which all machines break down simultaneously. Reasonably restricted settings might still allow for a universal sequence with constant performance guarantee.

We hope to initiate the study of universal sequencing also for other scheduling problems. As above, problem variants with different objective functions, e.g., minimizing the makespan on multiple machines, must avoid the extreme machine behavior of simultaneous unavailability. The more complex machine environment of a flowshop seems particularly interesting. A significant amount of research addresses flowshop scheduling on two machine stages with limited machine availability; see, e.g., the recent survey [32]. While several algorithms have been shown to perform very well (even arbitrarily well, depending on the specific setting) when the machine availability is known in advance, we are not aware of investigations on universal solutions. Finally, notice that we focused in our present work more on universal solutions for unreliable machines and less on universal solutions for different cost functions. Even on an ideal machine, it would be interesting to explore which properties of cost functions and scheduling constraints are sufficient to admit good universal solutions.

Acknowledgments. We thank two anonymous referees whose comments helped to improve the presentation of the paper.

REFERENCES

[1] I. Adiri, J. Bruno, E. Frostig, and A.H.G. Rinnooy Kan, *Single machine flow-time scheduling with a single breakdown*, Acta Inform., 26 (1989), pp. 679–696.

[2] N. Bansal and K. Pruhs, *The geometry of scheduling*, in Proceedings of the 51th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2010), IEEE Computer Society, Piscataway, NJ, 2010, pp. 407–414.

[3] L. Becchetti, S. Leonardi, A. Marchetti-Spaccamela, and K. Pruhs, *Online weighted flow time and deadline scheduling*, J. Discrete Algorithms, 4 (2006), pp. 339–352.

[4] A. Ben-Tal and A. Nemirovski, *Robust solutions of linear programming problems contaminated with uncertain data*, Math. Program., 88 (2000), pp. 411–424.

[5] J.L. Carter and M.N. Wegman, *Universal classes of hash functions*, J. Comput. System Sci., 18 (1979), pp. 143–154.

[6] M. Cheung and D. Shmoys, *A primal-dual approximation algorithm for min-sum single-machine scheduling problems*, in Proceedings of the 14th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX 2011), Lecture Notes in Comput. Sci. 6845, Springer, New York, 2011, pp. 135–146.

[7] M. Chrobak, C. Kenyon, J. Noga, and N.E. Young, *Incremental medians via online bidding*, Algorithmica, 50 (2008), pp. 455–478.

[8] M. Chrobak and C. Kenyon-Mathieu, *SIGACT news online algorithms column* 10: *Competitiveness via doubling*, SIGACT News, 37 (2006), pp. 115–126.

[9] F. Diedrich, K. Jansen, U.M. Schwarz, and D. Trystram, *A survey on approximation algorithms for scheduling with machine unavailability*, in Algorithmics of Large and Complex Networks: Design, Analysis, and Simulation, Springer, New York, 2009, pp. 50–64.

[10] L. Epstein, A. Levin, A. Marchetti-Spaccamela, N. Megow, J. Mestre, M. Skutella, and L. Stougie, *Universal sequencing on a single machine*, in Proceedings of the 14th Conference on Integer Programming and Combinatorial Optimization (IPCO 2010), Lecture Notes in Comput. Sci. 6080, Springer, Heidelberg, 2010, pp. 230–243.

[11] P. Erdős and G. Szekeres, *A combinatorial problem in geometry*, Compos. Math., 2 (1935), pp. 463–470.
[12] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan, *Optimization and approximation in deterministic sequencing and scheduling: A survey*, Ann. Discrete Math., 5 (1979), pp. 287–326.
[13] M.T. Hajiaghayi, K. Jain, L.C. Lau, I.I. Mandoiu, A. Russell, and V.V. Vazirani, *Minimum multicolored subgraph problem in multiplex PCR primer set selection and population haplotyping*, in Proceedings of the Second IWBRA, Lecture Notes in Comput. Sci. 3992, Springer, New York, 2006, pp. 758–766.
[14] 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 (1997), pp. 513–544.
[15] J.M. Hammersley, *A few seedlings of research*, in Proceedings of the Sixth Berkeley Symposium on Mathematical Statistics and Probability, vol. 1, University of California Press, Berkeley, CA, 1972, pp. 345–394.
[16] W. Höhn and T. Jacobs, *On the performance of Smith's rule in single-machine scheduling with nonlinear cost*, in Proceedings of the 10th Latin American Symposium on Theoretical Informatics (LATIN'12), Lecture Notes in Comput. Sci. 7256, Springer, New York, 2012, pp. 482–493.
[17] O.H. Ibarra and C.E. Kim, *Fast approximation algorithms for the knapsack and sum of subset problems*, J. ACM, 22 (1975), pp. 463–468.
[18] L. Jia, G. Lin, G. Noubir, R. Rajaraman, and R. Sundaram, *Universal approximations for TSP, Steiner tree, and set cover*, in Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC'05), ACM, New York, 2005, pp. 386–395.
[19] D.S. Johnson and K.A. Niemi, *On knapsacks, partitions, and a new dynamic programming technique for trees*, Math. Oper. Res., 8 (1983), pp. 1–14.
[20] I. Kacem, *Approximation algorithm for the weighted flow-time minimization on a single machine with a fixed non-availability interval*, Comput. Indust. Engrg., 54 (2008), pp. 401–410.
[21] I. Kacem and A.R. Mahjoub, *Fully polynomial time approximation scheme for the weighted flow-time minimization on a single machine with a fixed non-availability interval*, Comput. Indust. Engrg., 56 (2009), pp. 1708–1712.
[22] R.M. Karp, *Reducibility among combinatorial problems*, in Complexity of Computer Computations (Proc. Sympos., IBM Thomas J. Watson Res. Center, Yorktown Heights, NY), Plenum, New York, 1972, pp. 85–103.
[23] H. Kellerer, M.A. Kubzin, and V.A. Strusevich, *Two simple constant ratio approximation algorithms for minimizing the total weighted completion time on a single machine with a fixed non-availability interval*, European J. Oper. Res., 199 (2009), pp. 111–116.
[24] H. Kellerer and V.A. Strusevich, *Fully polynomial approximation schemes for a symmetric quadratic knapsack problem and its scheduling applications*, Algorithmica, 57 (2010), pp. 769–795.
[25] S.G. Kolliopoulos and G. Steiner, *Partially ordered knapsack and applications to scheduling*, Discrete Appl. Math., 155 (2007), pp. 889–897.
[26] P. Kouvelis and G. Yu, *Robust Discrete Optimization and Its Applications*, Springer, New York, 1997.
[27] E.L. Lawler, *A dynamic programming algorithm for preemptive scheduling of a single machine to minimize the number of late jobs*, Ann. Oper. Res., 26 (1990), pp. 125–133.
[28] C.-Y. Lee, *Machine scheduling with an availability constraint*, J. Global Optim., 9 (1996), pp. 395–416.
[29] C.-Y. Lee, *Machine scheduling with availability constraints*, in Handbook of Scheduling, J.Y.-T. Leung, ed., CRC Press, Boca Raton, FL, 2004.
[30] C.-Y. Lee and S.D. Liman, *Single machine flow-time scheduling with scheduled maintenance*, Acta Inform., 29 (1992), pp. 375–382.
[31] C. Liebchen, M.E. Lübbecke, R.H. Möhring, and S. Stiller, *The concept of recoverable robustness, linear programming recovery, and railway applications*, in Robust and Online Large-Scale Optimization, R.K. Ahuja, R.H. Möhring, and Chr. Zaroliagis, eds., Lecture Notes in Comput. Sci. 5868, Springer-Verlag, Berlin, 2009, pp. 1–27.
[32] Y. Ma, C. Chu, and C. Zuo, *A survey of scheduling with deterministic machine availability constraints*, Comput. Indust. Engrg., 58 (2010), pp. 199–211.
[33] M. Mastrolilli, N. Mutsanas, and O. Svensson, *Approximating single machine scheduling with scenarios*, in Proceedings of the 11th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX 2008), Lecture Notes in Comput. Sci. 5171, Springer, New York, 2008, pp. 153–164.

[34] N. Megow and J. Verschae, *Note on Scheduling on a Single Machine with One Non-availability Period*, unpublished manuscript, 2008.
[35] K. Pruhs and G.J. Woeginger, *Approximation schemes for a class of subset selection problems*, Theoret. Comput. Sci., 382 (2007), pp. 151–156.
[36] H. Räcke, *Survey on oblivious routing strategies*, in Mathematical Theory and Computational Practice, Proceedings of the 5th Conference on Computability in Europe (CiE), K. Ambos-Spies, B. Löwe, and W. Merkle, eds., Lecture Notes in Comput. Sci. 5635, Springer, New York, 2009, pp. 419–429.
[37] G. Schmidt, *Scheduling with limited machine availability*, European J. Oper. Res., 121 (2000), pp. 1–15.
[38] A.S. Schulz and M. Skutella, *The power of α-points in preemptive single machine scheduling*, J. Sched., 5 (2002), pp. 121–133.
[39] W.E. Smith, *Various optimizers for single-stage production*, Naval Res. Logist. Quart., 3 (1956), pp. 59–66.
[40] A. Soyster, *Convex programming with set-inclusive constraints and applications to inexact linear programming*, Oper. Res., 21 (1973), pp. 1154–1157.
[41] S. Stiller and A. Wiese, *Increasing speed scheduling and flow scheduling*, in Proceedings of the 21st Symposium on Algorithms and Computation (ISAAC 2010), Lecture Notes in Comput. Sci. 6507, Springer, New York, 2010, pp. 279–290.
[42] L.G. Valiant and G.J. Brebner, *Universal schemes for parallel communication*, in Proceedings of the 13th Annual ACM Symposium on Theory of Computation (STOC'81), ACM, New York, 1981, pp. 263–277.
[43] G. Wang, H. Sun, and C. Chu, *Preemptive scheduling with availability constraints to minimize total weighted completion times*, Ann. Oper. Res., 133 (2005), pp. 183–192.