

Universal sequencing on a single machine*

Leah Epstein[†] Asaf Levin[‡] Alberto Marchetti-Spaccamela[§] Nicole Megow[¶]
Julián Mestre^{||} Martin Skutella^{**} Leen Stougie^{††}

March 16, 2011

Abstract

We consider scheduling on an unreliable machine that may experience unexpected changes in processing speed or even full breakdowns. We aim for a universal solution that performs well without adaptation for any possible machine behavior. For the objective of minimizing the total weighted completion time, we design a polynomial time 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 disruptions in advance. A randomized version of this algorithm attains in expectation a ratio of e . We also show that both results are best possible among all universal solutions. As a direct consequence of our results, we answer affirmatively the question of whether a constant approximation algorithm exists for the offline version of the problem when machine unavailability periods are known in advance.

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. Motivated by this hardness, we study the special case when the processing time of each job is proportional to its weight. We present a non-trivial algorithm with a small constant performance guarantee.

Keywords: scheduling, single machine, unreliable machine behavior, universal solution, worst case guarantees

1 Introduction

Traditional scheduling theory assumes in their standard models that jobs are processed on ideal machines that provide the same constant performance throughout time. While in some settings this is a

*A preliminary version of this work appeared in *Proceedings of the 14th International Conference on Integer Programming and Combinatorial Optimization (IPCO XIV)*, Springer, 2010.

[†]Department of Mathematics, University of Haifa, Israel. Email: lea@math.haifa.ac.il

[‡]Chaya fellow. Faculty of Industrial Engineering and Management, The Technion, Haifa, Israel. Email: levinas@ie.technion.ac.il

[§]Department of Computer and System Sciences, Sapienza University of Rome, Italy. Email: alberto.marchetti@dis.uniroma1.it. Supported by EU project 215270 FRONTS.

[¶]Max-Planck-Institut für Informatik, Saarbrücken, Germany. Email: nmegow@mpi-inf.mpg.de

^{||}School of Information Technologies, University of Sydney, Australia. Email: mestre@it.usyd.edu.au.

^{**}Department of Mathematics, Technische Universität Berlin, Germany. Email: skutella@math.tu-berlin.de. Supported by DFG research center MATHEON in Berlin.

^{††}Department of Econometrics and Operations Research, Vrije Universiteit Amsterdam & CWI, Amsterdam, The Netherlands. Email: stougie@cwi.nl. Supported by the Dutch BSIK-BRIKS project.

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.

Different machine behaviors will typically lead to very different optimal schedules. This creates a burden on the scheduler who would have 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 machine to minimize average completion times. Our main result is an algorithm for computing a universal sequence that is always a constant factor away from an optimal clairvoyant algorithm. We complement this by showing that our upper bound is best possible among universal schedules. We also consider the case when jobs have release dates. Here we provide an almost logarithmic lower bound on the performance of universal schedules, thus showing a drastic difference with respect to the setting without release dates. Finally, we design an algorithm with constant performance for the interesting case of scheduling jobs with release dates and proportional weights.

Our hope is that these results stimulate the study of universal solutions 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.

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 the total sum of weighted completion times. 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 be resumed processing at any later moment when the machine becomes available again. We analyze the worst case performance by comparing the solution value provided by an algorithm 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 more general problem in which each job $j \in J$ has its 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 in Section 4. We allow preemption in the actual scheduling procedure, however, as in the case without release dates, we aim for non-adaptive 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 finished yet and that has already been released. This procedure is called *preemptive list scheduling* [11, 34]. 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.

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, as e.g. in hash-

ing [4] and routing [38]. The latter is also known as *oblivious routing* and has been studied extensively; see [32] for a state-of-the-art overview. Jia et al. [14] considered universal approximations for TSP, Steiner Tree, and Set Cover Problems. All this research falls broadly into the field of robust optimization [3]. 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. [22, chap. 7] and [29]. 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 [36], also called *strict robustness* [28], and in this regard, we follow the terminology of universal solutions.

Scheduling with limited machine availability is a subfield of machine scheduling that has been studied for more than twenty years; see, e.g., the surveys [7, 25, 33]. Different objective functions, stochastic breakdowns, as well as the offline problem with known machine availability periods have been investigated. Nevertheless, only few results are known on the problem of scheduling to minimize the total weighted completion time, and none of these deal with release dates. If all jobs have equal weights, a simple interchange argument shows that sequencing jobs in non-decreasing order of processing times is optimal as it is in the setting with continuous machine availability [35]. 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 to the weighted setting, that is, scheduling jobs in non-increasing order of ratios weight over processing time, also known as *Smith's Rule* [35], does not yield a constant performance guarantee. This is true even if there is just a single machine breakdown as is shown e.g. in [24]. In fact, this special problem is weakly NP-hard in both the preemptive [24] and the non-preemptive variant [1, 26], and has stimulated a major line of research. Several approximation results have been derived, see [16, 19, 24, 26, 30, 39], which were recently complemented by fully polynomial-time approximation schemes, see [8, 17, 20].

The special class of instances, in which the processing time of each job is proportional to its weight, has been studied in [39]. The authors showed that scheduling in non-increasing order of processing times (or weights) yields a 2-approximation for preemptive scheduling. However, for the general problem with arbitrary job weights, it remained an open question [39] if a polynomial time algorithm with constant approximation ratio exists, even without release dates. In this case, the problem is strongly NP-hard [39].

Our results. Our main results are algorithms that compute deterministic and randomized universal sequences for jobs without release dates. These algorithms run in polynomial time and output a permutation of the jobs such that scheduling the jobs in this order will always yield a solution that remains within multiplicative factor 4 and within multiplicative factor e in expectation from any given schedule. Furthermore, we show that our algorithms can be adapted to solve more general problem instances with certain types of precedence constraints without losing performance quality. We also show that our upper bounds are best possible for universal sequencing. This is done by establishing an interesting connection between our problem and a certain online bidding problem [5].

It may seem rather surprising that universal sequences with constant performance guarantee should always exist. In fact, our results immediately answer affirmatively an open question in the area of offline scheduling with limited machine availability: whether there exists a constant factor approximation algorithm for scheduling jobs 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 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 unit 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. [11] 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 [6] 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. [2] in the context of on-line 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, best possible $(4 + \epsilon)$ -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; unfortunately, unlike in the case without release dates, this does not translate into the same performance guarantee for universal solutions 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 $\Omega(\log n / \log \log n)$. Our proof relies on the classical theorem of Erdős and Szekeres [9] 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 non-trivial 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

Given a single machine that runs continuously at unit speed (ideal machine), the completion time C_j^π of job j when applying preemptive list scheduling to sequence π is uniquely defined. 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 sequence π , that is, $W^\pi(t) := \sum_{j: C_j^\pi > t} w_j$. Then,

$$\sum_{j \in J} w_j C_j^\pi = \int_0^\infty W^\pi(t) dt. \quad (1)$$

Clearly, breaks or fluctuations in the speed of the machine delay the completion times. To describe a particular machine behavior, let $f : \mathbb{R}_+ \rightarrow \mathbb{R}_+$ be a non-decreasing continuous function, with $f(t)$ being the aggregated amount of processing time available on the machine up to time t . We refer to f as the *machine capacity function*. If the derivative of f 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 $f(t) = t, t \geq 0$.

For a given capacity function f , let $S(\pi, f)$ denote the single machine schedule when applying preemptive list scheduling to permutation π , and let $C_j^{S(\pi, f)}$ denote the completion time of job j in this particular schedule. For some point in time $t \geq 0$, let $W^{S(\pi, f)}(t)$ denote the total weight of jobs

that are not yet completed by time t in schedule $S(\pi, f)$. Then,

$$\sum_{j \in J} w_j C_j^{S(\pi, f)} = \int_0^\infty W^{S(\pi, f)}(t) dt.$$

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

Observation 1. For a given machine capacity function f ,

$$\int_0^\infty W^{S^*(f)}(t) dt \tag{2}$$

is a lower bound on the objective function of any schedule.

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

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

$$W^{S(\pi, f)}(t) \leq c W^{S^*(f)}(t) \quad \text{for all } t \geq 0.$$

Proof. The ‘‘if’’ part is clear, since by Observation 1

$$\sum_{j \in J} w_j C_j^{S(\pi, f)} = \int_0^\infty W^{S(\pi, f)}(t) dt \leq c \int_0^\infty W^{S^*(f)}(t) dt.$$

We prove the ‘‘only if’’ part by contradiction. Assume that $W^{S(\pi, f)}(t_0) > c W^{S^*(f)}(t_0)$ for some t_0 and f . For any $t_1 > t_0$ consider the following machine capacity function

$$f'(t) = \begin{cases} f(t) & \text{if } t \leq t_0, \\ f(t_0) & \text{if } t_0 < t \leq t_1, \\ f(t - t_1 + t_0) & \text{if } t > t_1 \end{cases}$$

which equals f up to time t_0 and then remains constant at value $f'(t) = f(t_0)$ for the time interval $[t_0, t_1]$. Hence,

$$\sum_{j \in J} w_j C_j^{S(\pi, f')} = \sum_{j \in J} w_j C_j^{S(\pi, f)} + (t_1 - t_0) W^{S(\pi, f)}(t_0). \tag{3}$$

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

$$\sum_{j \in J} w_j C_j^{S(\pi^*, f')} = \sum_{j \in J} w_j C_j^{S(\pi^*, f)} + (t_1 - t_0) W^{S^*(f')}(t_0). \tag{4}$$

As t_1 tends to infinity, the ratio of (3) and (4) tends to $W^{S(\pi, f)}(t_0) / W^{S^*(f)}(t_0) > c$, a contradiction. \square

In case that all release dates are equal, approximating the sum of weighted completion times 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 $f(t) = t$, $t \geq 0$. Scheduling according to sequence π on such a machine yields for each j ,

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

The completion time under machine capacity function f is

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

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

$$W^{S(\pi, f)}(t) = W^\pi(f(t)) \quad \text{for all } t \geq 0.$$

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

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

$$W^\pi(t) \leq cW^*(t) \quad \text{for all } t \geq 0.$$

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 $\hat{t} = n - 1$ we have $W^\pi(\hat{t}) = W^*(\hat{t}) = w_n$.*

However, when considering an unreliable machine with machine capacity function f and the corresponding schedule $S(\pi, f)$, then the situation changes drastically. Let f 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 $W^{S(\pi, f)}(t) = w_n$, which equals $W^\pi(f(t))$ with $f(t) = \hat{t}$. However, an optimal solution executes job n in $[n - 1, n)$, instead of job $n - 1$, and has remaining weight $W^{S^(f)}(t) = 1 \neq W^*(f(t))$. Obviously, Observation 2 does not hold if jobs have arbitrary release dates.*

3 Universal scheduling without release dates

In this section we study the universal scheduling problem for jobs without release dates.

3.1 Upper bounds

In the sequel 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 $f(t) = t, t \geq 0$. 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 $i \in \{0, 1, \dots, \lceil \log w(J) \rceil\}$, 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 $\sum_{j \in J} w_j C_j^{S(\pi, f)}$ is less than 4 times the optimum for all machine capacity functions f .*

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,

$$W^\pi(t) \leq \sum_{k=0}^i w(J_k^*) \leq \sum_{k=0}^i 2^k = 2^{i+1} - 1. \quad (5)$$

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 (5) this concludes the proof. \square

Notice that the algorithm takes exponential time since finding the subsets of jobs J_i^* is a KNAPSACK problem and, thus, NP-hard [18]. On the other hand, job sets J_i^* can be found in pseudo-polynomial 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, instead of J_i^* , computing 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 [13] 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 $\sum_{j \in J} w_j C_j^{S(\pi, f)}$ is less than $4 + \epsilon$ times the optimum for all machine capacity functions f .*

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 (5) we get the slightly weaker bound

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

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

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 $i \in \{0, 1, \dots, \lceil \ln w(J) \rceil\}$ 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 total weighted completion time.

Corollary 1. *Let π be a random sequence of jobs and $c > 0$. Then, the expected objective value $\mathbb{E} \left[\sum_{j \in J} w_j C_j^{S(\pi, f)} \right]$ is at most c times the optimum value for all machine capacity functions f if and only if $\mathbb{E} [W^\pi(t)] \leq cW^*(t)$ for all $t \geq 0$.*

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 $i < \lceil \ln(W^*(t)/X) \rceil$. Similar to (5) we have for any t ,

$$\begin{aligned} \mathbb{E} [W^{\pi(X)}(t)] &\leq \mathbb{E} \left[\sum_{k=0}^i Xe^k \right] = \mathbb{E} \left[X \frac{e^{i+1} - 1}{e - 1} \right] < \mathbb{E} \left[X \frac{e^{\lceil \ln(W^*(t)/X) \rceil + 1} - 1}{e - 1} \right] \\ &= \frac{e}{e - 1} W^*(t) \mathbb{E} \left[e^{\lceil \ln(W^*(t)/X) \rceil - \ln(W^*(t)/X)} \right] - 1. \end{aligned}$$

Let $Z := \lceil \ln W^*(t) \rceil - Y - (\ln W^*(t) - Y)$ which is 1 minus the fractional part of Z . Then Z is a random variable distributed like Y uniformly in $[0, 1]$. Thus, $\mathbb{E} [e^Z] = \mathbb{E} [X] = e - 1$, which concludes the proof by Corollary 1. \square

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 the objective value $\sum_{j \in J} w_j C_j^{S(\pi, f)}$ is in expectation less than $e + \epsilon$ times the optimum value for all machine capacity functions f .*

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. [5]. This allows us to prove tight lower bounds for our problem.

In online bidding, we are given a universe $\mathcal{U} = \{1, \dots, n\}$. A bid set is just a subset of \mathcal{U} . A given bid set \mathcal{B} is said to be α -competitive if

$$\sum_{b \in \mathcal{B}: b < T} b + \min_{b \in \mathcal{B}: b \geq T} b \leq \alpha T \quad \text{for all } T \in \mathcal{U}. \quad (6)$$

Chrobak et al. [5] gave lower bounds of $4 - \epsilon$ and $e - \epsilon$, for any $\epsilon > 0$, for deterministic and randomized algorithms, respectively.

Theorem 5. *For any $\epsilon > 0$, there exists an instance of the universal scheduling problem on which the performance ratio 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 \mathcal{U} . For any $j \in \mathcal{U}$, let $k(j)$ be the largest index such that $\pi_{k(j)} \geq j$. Since $p_j > \sum_{i=1}^{j-1} p_i$, 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 ratio of α then, it holds by Lemma 2

$$\sum_{k=k(j)}^n \pi_k \leq \alpha j \quad \text{for all } j \in \mathcal{U}. \quad (7)$$

From sequence π we extract another sequence of jobs:

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

Define the bid set $\mathcal{W} = \{\mathcal{W}_1, \mathcal{W}_2, \dots\}$. 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 $\{i \in \mathcal{W} \mid i < j\} \cup \min \{i \in \mathcal{W} \mid i \geq j\} \subset \{\pi_{k(j)}, \dots, \pi_n\}$. Hence, if π achieves a performance ratio of α then

$$\sum_{i \in \mathcal{W}: i < j} i + \min_{i \in \mathcal{W}: i \geq j} i \leq \sum_{k=k(j)}^n \pi_k \leq \alpha j \quad \text{for all } j \in \mathcal{U}, \quad (8)$$

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 ratio 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 (6), \mathcal{B} is said to be α -competitive if

$$\mathbb{E} \left[\sum_{b \in \mathcal{B}: b < T} b + \min_{b \in \mathcal{B}: b \geq T} b \right] \leq \alpha T \quad \text{for all } T \in \mathcal{U}. \quad (9)$$

In the scheduling setting, analogous to (7), if the random permutation π yields performance ratio α then

$$\mathbb{E} \left[\sum_{k=k(j)}^n \pi_k \right] \leq \alpha j \quad \text{for all } j \in \mathcal{U}. \quad (10)$$

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

$$\mathbb{E} \left[\sum_{i \in \mathcal{W}: i < j} i + \min_{i \in \mathcal{W}: i \geq j} i \right] \leq \mathbb{E} \left[\sum_{k=k(j)}^n \pi_k \right] \leq \alpha j \quad \text{for all } j \in \mathcal{U}. \quad (11)$$

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. \square

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 [15] and also hard to approximate [10]. 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 [15, 21].

Theorem 6. *Let $\epsilon > 0$. Consider the universal sequencing problem with precedence constraints (J, \prec) . If there is an FPTAS for the partially ordered knapsack problem 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 $\sum_{j \in J} w_j C_j^{S(\pi, f)}$ is less than $4 + \epsilon$ times the optimum for all machine capacity functions f . 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 $0, 1, 2, \dots$ 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. \square

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 non-preemptively in exactly this order, yields a constant performance guarantee. 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 cost 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 non-adaptive 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 at minimizing the total remaining weight, can be adapted to the setting with release dates, as we show in Section 4.1. Unfortunately, this is only meaningful for an ideal machine. In the presence of release dates approximation ratios on an ideal machine do not translate directly to 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 non-trivial 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 Minimizing remaining weight on an ideal machine

Consider scheduling on an ideal single machine with the objective to minimize the total remaining weight at any time. 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 $i \in \{0, 1, \dots, \lceil \log w(J) \rceil\}$, 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 $w(J_i^*) \leq 2^i$. 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.
-

Lawler [23] provides a pseudo-polynomial 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 $\Delta_i \in [0, \Delta_{i-1}]$ using Lawler's algorithm.

Theorem 7. *For every scheduling instance DOUBLE-R produces a sequence π such that $W^\pi(t) < 4W^*(t) - 1$ for all $t \geq 0$ on an ideal machine.*

Proof. Given a schedule S obtained by preemptive list scheduling of sequence π 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) holds $C_k^{S_i} \leq T - \Delta_i$. 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,

$$W^S(t) \leq \sum_{k=0}^i w(J_k^*) \leq \sum_{k=0}^i 2^k = 2^{i+1} - 1. \quad (12)$$

In case $i = 0$, the claim is trivially true since $w_j \geq 1$ for any $j \in J$, and thus, $W^*(t) = W^S(t)$. 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, $W^*(t) > 2^{i-1}$. Together with (12) this concludes the proof. \square

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 $\mathbb{E} [W^{\pi(X)}(t)] < eW^*(t) - 1$ for all $t \geq 0$ on an ideal machine.*

Lawler's algorithm used in Step 2 runs in pseudopolynomial time. However, Pruhs and Woeginger [31] 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 8. *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 $W^\pi(t) < (4 + \epsilon)W^*(t)$ for all $t \geq 0$ on an ideal machine. A randomized variant yields a random permutation $\pi(X)$ with $\mathbb{E} [W^{\pi(X)}(t)] < (e + \epsilon)W^*(t)$ 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.

Theorem 9. *There exists an instance with n jobs with release dates, where the performance guarantee of any universal schedule is $\Omega(\log n / \log \log n)$, even if all weights are equal.*

In our lower bound instance each job j has weight $w_j = 1$, $j = 0, 1, \dots, n - 1$. The processing times of the jobs form a geometric series $p_j = 2^j$, $j = 0, 1, \dots, n - 1$, and they are released in reversed order $r_j = \sum_{i>j}^{n-1} 2^i = \sum_{i>j} p_i$, $j = 0, 1, \dots, n - 1$. 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.

To get some intuition, consider the universal job sequence $n - 1, n - 2, \dots, 0$. If the machine is unavailable from time 0 on until all jobs are released, we arrive in a setting without release dates.

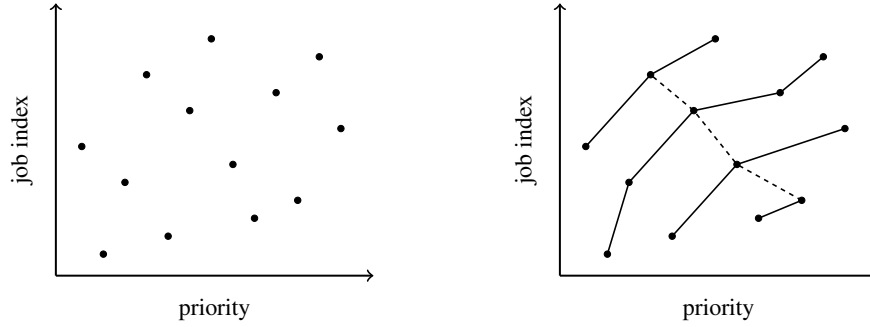


Figure 1: Two-dimensional visualization of a universal sequence. On the left: Each point represents one job; x-coordinates correspond to positions in the sequence and y-coordinates correspond to job indices. On the right: The points have been decomposed into four increasing subsequences. And there is a decreasing subsequence of length four.

Suppose that the machine becomes available at time 2^n and works then at full speed. Then at time $2^n + p_{n-1} - 1 = 2^n + 2^{n-1} - 1$ the universal schedule has still n uncompleted jobs, whereas an optimal solution has completed $0, 1, \dots, n - 2$, yielding by Lemma 1 a performance guarantee of $\Omega(n)$.

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

To show that there is no way to “interpolate” between these two extremes, we rely on a classic theorem of Erdős and Szekeres [9] or, more precisely, on Hammersley’s proof [12] of this result.

Lemma 3 (Hammersley [12]). *Any given sequence of n distinct numbers x_1, x_2, \dots, x_n can be decomposed into k increasing subsequences $\ell_1, \ell_2, \dots, \ell_k$ with the properties:*

1. *There exists a decreasing subsequence of length k .*
2. *If x_i belongs to ℓ_a then for all $j > i$ if $x_j < x_i$ then x_j belongs to ℓ_b and $b > a$.*

The idea is now to view any universal sequence as a permutation of $\{0, 1, \dots, n - 1\}$ and use Lemma 3 to decompose the sequence into k increasing subsequences. Figure 1 provides a two-dimensional visualization. This decomposition is then used to design a breakdown pattern that will yield Theorem 9. 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. \square

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

Proof. For each job j in ℓ_i there is a breakdown $[r_j, r_j + \epsilon]$. For each job j in $\ell_{i+1}, \dots, \ell_k$ there is a breakdown $[r_j, r_j + p_j] = [r_j, r_j + 2^j]$. As a consequence, at time $2^n - 1$ the universal schedule has all jobs in ℓ_i and all jobs in $\ell_{i+1}, \dots, \ell_k$ uncompleted, whereas, a schedule exists that leaves the last job of ℓ_i and all jobs in $\ell_{i+1}, \dots, \ell_k$ uncompleted. Therefore, a breakdown $[2^n - 1, L]$ for L large enough implies the lemma. \square

Proof of Theorem 9. Consider an arbitrary universal scheduling solution and its decomposition into increasing subsequences ℓ_1, \dots, ℓ_k as in Lemma 3 and let α be its performance guarantee.

Using Lemma 5, one can easily prove by induction that $|\ell_i| \leq \alpha^{k-i+1}$. Since ℓ_1, \dots, ℓ_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 $\alpha = \Omega\left(\frac{\log n}{\log \log n}\right)$. \square

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

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 $i \in 1, 2, \dots, z$, 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 non-decreasing order of release dates at the end of π .
-

Theorem 10. *The performance guarantee of SORTCLASS for universal scheduling of jobs with proportional weights and release dates is exactly 5.*

Proof. Let π be the job sequence computed by SORTCLASS. By Lemma 1, it is sufficient to prove

$$W^{S(\pi, f)}(t) \leq 5W^{S^*(f)}(t) \quad \text{for all } t > 0. \quad (13)$$

Take any time t and any machine capacity function f . Let $j \in J_i$ be the job being processed at time t according to the schedule $S(\pi, f)$. 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 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 $p(J) - f(t)$, 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}, \dots, 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,

$$W^{S(\pi, f)}(t) < 2^{i+1} + p(J) - f(t). \quad (14)$$

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

$$W^{S^*(f)}(t) \geq p(J) - f(t). \quad (15)$$

For another lower bound, let t' be the last time before t in which the machine is available but it is either 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 $S(\pi, f)$, 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} :

$$W^{S^*(f)}(t) \geq 2^{i-1}. \quad (16)$$

Combining (14), (15), and (16) yields (13) 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 $j = 1, \dots, k$. 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 $r_a = \sum_{i \leq k} p_i$, $r_b = r_a + \epsilon$, and $r_c = r_b + \epsilon$, for some $0 < \epsilon < 1$.

SORTCLASS transforms the sequence $\pi = 1, 2, \dots, k-1, k, a, b, c$ of release date order into $a, b, k, c, k-1, k-2, \dots, 2, 1$. To show the lower bound, we give a break of length ϵ at each release date r_j for $j = 1, 2, \dots, k$. Thus, the jobs $j = 1, \dots, k$ 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 $1, 2, \dots, k-1, k, b, c, a$ 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. \square

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 10 gives directly the following guarantee.

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

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 11. *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 R such that $R < 3$. We define the following sequence: $a_1 = 1$, $a_2 = R$ and for $i \geq 3$, $a_i = (R + 1)(a_{i-1} - a_{i-2})$. Let $S_i = \sum_{j=1}^i a_j$. Note that an alternative definition for a_i , $i \geq 3$, is $a_i = R \cdot a_{i-1} - S_{i-2}$. Indeed, using the definition of the sequence, yields

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

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

Note that by letting $C = R + 1$, we get exactly the well known sequence defined by the recurrence $b_i = C(b_{i-1} - b_{i-2})$; see, e.g., [5]. 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 $b_1 < b_2 < \dots < b_n$, while $0 < b_{n+1} \leq b_n$. 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 R and we next characterize the permutation which the algorithm must use.

Consider the time $T - \epsilon$. If the machine works continuously till 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 Ra_1 . Since $R = a_2 < a_3 < \dots < a_n$, the only such job can be job 1. We next prove by induction that if the last jobs in the permutation are jobs $k - 1, k - 2, \dots, 1$, then the job before $k - 1$ must be job k (for any $2 \leq k \leq n - 1$). 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 $1, 2, \dots, k - 1$ have a lower priority than jobs $k, k + 1, \dots, n$, and among the set $\{1, 2, \dots, k - 1\}$, jobs of lower indices have a lower priority, job $j - 1$ is preempted upon the release of job j , for $j = 2, 3, \dots, k - 1$, 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 $R \cdot a_k$. Therefore, since $a_{k+1} + S_{k-1} = R \cdot a_k$ and $a_{k+1} = \min\{a_{k+1}, a_{k+2}, \dots, a_n\}$, the only additional incomplete job must be job k .

Since the $n - 1$ last jobs in the permutation must be $n - 1, n - 2, \dots, 1$, 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 $a_{n+1} = (R + 1)(a_n - a_{n-1}) \leq a_n$ or $R \cdot a_n \leq (R + 1)a_{n-1}$. Moreover, $a_n = Ra_{n-1} - S_{n-2}$, so $\frac{S_n}{a_n} = \frac{a_n + a_{n-1} + S_{n-2}}{a_n} = \frac{(R+1)a_{n-1}}{a_n} \geq R$, which is a contradiction. \square

5 Further remarks

In Section 4 we have shown that the performance of universal scheduling algorithms may deteriorate drastically when generalizing the universal scheduling problem slightly. Other generalizations do not admit any (exponential time) algorithm with bounded performance guarantee. If a non-adaptive 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.

To the best of our knowledge, the approximation results we give for universal scheduling are also the best currently known results for the offline version of our problem. Notice, that any offline variant (even without release dates) in which preemption is not allowed or causes extra work is not approximable in polynomial time; a reduction from the 2-PARTITION problem shows that the problem with two or more non-available periods is not approximable, unless $P=NP$, even if all jobs have equal weight. A reduction in that spirit has been used in [40] for a scheduling problem with some jobs having a fixed position in the schedule. Similarly, we can rule out constant approximation factors for any preemptive version of the problem in which the makespan of the jobs cannot be computed exactly in polynomial time. This is shown by simple reductions from the corresponding decision version of the makespan minimization problem. Such variations of our problem are scheduling with general precedence constraints (even if all jobs have unit processing times) [37] and scheduling with in-tree precedence constraints and release dates [27].

References

- [1] I. Adiri, J. Bruno, E. Frostig, and A. Rinnooy Kan. Single machine flow-time scheduling with a single breakdown. *Acta Informatica*, 26(7):679–696, 1989.
- [2] L. Becchetti, S. Leonardi, A. Marchetti-Spaccamela, and K. Pruhs. Online weighted flow time and deadline scheduling. *Journal of Discrete Algorithms*, 4(3):339–352, 2006.
- [3] A. Ben-Tal and A. Nemirovski. Robust solutions of linear programming problems contaminated with uncertain data. *Mathematical Programming*, 88:411–424, 2000.
- [4] J. Carter and M. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18:143–154, 1979.
- [5] M. Chrobak, C. Kenyon, J. Noga, and N. E. Young. Incremental medians via online bidding. *Algorithmica*, 50(4):455–478, 2008.
- [6] M. Chrobak and C. Kenyon-Mathieu. Sigact news online algorithms column 10: Competitive-ness via doubling. *SIGACT News*, 37(4):115–126, 2006.

- [7] 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*, pages 50–64. Springer, 2009.
- [8] 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)*, volume 6080 of *Lecture Notes in Computer Science*, pages 230–243. Springer Heidelberg, 2010.
- [9] P. Erdős and G. Szekeres. A combinatorial problem in geometry. *Compositio Mathematica*, 2:463–470, 1935.
- [10] M. Hajiaghayi, K. Jain, L. Lau, I. Mandoiu, A. Russell, and V. Vazirani. Minimum multicolored subgraph problem in multiplex PCR primer set selection and population haplotyping. In *Proceedings of Second IWBRA*, volume 3992 of *LNCS*, pages 758–766. Springer, 2006.
- [11] L. Hall, A. S. Schulz, D. Shmoys, and J. Wein. Scheduling to minimize average completion time: off-line and on-line approximation algorithms. *Mathematics of Operations Research*, 22:513–544, 1997.
- [12] J. Hammersley. A few seedlings of research. In *Proceedings Sixth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1 of *University of California Press*, pages 345–394, 1972.
- [13] O. H. Ibarra and C. E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *Journal of the ACM*, 22(4):463–468, 1975.
- [14] 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)*, pages 386–395, 2005.
- [15] D. S. Johnson and K. A. Niemi. On knapsacks, partitions, and a new dynamic programming technique for trees. *Mathematics of Operations Research*, 8(1):1–14, 1983.
- [16] I. Kacem. Approximation algorithm for the weighted flow-time minimization on a single machine with a fixed non-availability interval. *Computers & Industrial Engineering*, 54(3):401–410, 2008.
- [17] 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. *Computers & Industrial Engineering*, 56(4):1708–1712, 2009.
- [18] R. M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations (Proc. Sympos., IBM Thomas J. Watson Res. Center)*, pages 85–103. Plenum, 1972.
- [19] 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 Journal of Operational Research*, 199(1):111–116, 2009.
- [20] H. Kellerer and V. A. Strusevich. Fully polynomial approximation schemes for a symmetric quadratic knapsack problem and its scheduling applications. *Algorithmica*, 57(4):769–795, 2010.

- [21] S. G. Kolliopoulos and G. Steiner. Partially ordered knapsack and applications to scheduling. *Discrete Applied Mathematics*, 155(8):889–897, 2007.
- [22] P. Kouvelis and G. Yu. *Robust Discrete Optimization and Its Applications*. Springer, 1997.
- [23] E. L. Lawler. A dynamic programming algorithm for preemptive scheduling of a single machine to minimize the number of late jobs. *Annals of Operations Research*, 26:125–133, 1990.
- [24] C.-Y. Lee. Machine scheduling with an availability constraint. *Journal of Global Optimization*, 9:395–416, 1996.
- [25] C.-Y. Lee. Machine scheduling with availability constraints. In J. Y.-T. Leung, editor, *Handbook of scheduling*. CRC Press, 2004.
- [26] C.-Y. Lee and S. D. Liman. Single machine flow-time scheduling with scheduled maintenance. *Acta Informatica*, 29(4):375–382, 1992.
- [27] J. K. Lenstra. Unpublished. Cited by P. Brucker and S. Knust in Complexity results for scheduling problems at <http://www.mathematik.uni-osnabrueck.de/research/OR/class>.
- [28] C. Liebchen, M. E. Lübbecke, R. H. Möhring, and S. Stiller. The concept of recoverable robustness, linear programming recovery, and railway applications. In R. Ahuja, R. Möhring, and C. Zaroliagis, editors, *Robust and Online Large-Scale Optimization*, volume 5868 of *Lecture Notes in Computer Science*, pages 1–27. Springer-Verlag, Berlin, 2009.
- [29] 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)*, volume 5171 of *LNCS*, pages 153–164, 2008.
- [30] N. Megow and J. Verschae. Note on scheduling on a single machine with one non-availability period. Unpublished manuscript, 2008.
- [31] K. Pruhs and G. J. Woeginger. Approximation schemes for a class of subset selection problems. *Theoretical Computer Science*, 382(2):151–156, 2007.
- [32] H. Räcke. Survey on oblivious routing strategies. In K. Ambos-Spies, B. Löwe, and W. Merkle, editors, *Mathematical Theory and Computational Practice, Proceedings of 5th Conference on Computability in Europe (CiE)*, volume 5635 of *LNCS*, pages 419–429. Springer, 2009.
- [33] G. Schmidt. Scheduling with limited machine availability. *European Journal of Operational Research*, 121(1):1–15, 2000.
- [34] A. S. Schulz and M. Skutella. The power of α -points in preemptive single machine scheduling. *Journal of Scheduling*, 5:121–133, 2002.
- [35] W. E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3:59–66, 1956.
- [36] A. Soyster. Convex programming with set-inclusive constraints and applications to inexact linear programming. *Operations Research*, 21(4):1154–1157, 1973.

- [37] J. Ullman. Complexity of sequencing problems. In J. Bruno, J. E.G. Coffman, R. Graham, W. Kohler, R. Sethi, K. Steiglitz, and J. Ullman, editors, *Computer and Job/Shop Scheduling Theory*, pages 139–164. John Wiley & Sons Inc., New York, 1976.
- [38] 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)*, pages 263–277, 1981.
- [39] G. Wang, H. Sun, and C. Chu. Preemptive scheduling with availability constraints to minimize total weighted completion times. *Annals of Operations Research*, 133:183–192, 2005.
- [40] J. Yuan, Y. Lin, C. Ng, and T. Cheng. Approximability of single machine scheduling with fixed jobs to minimize total completion time. *European Journal of Operational Research*, 178(1):46–56, 2007.