Robust sequencing on a single machine

A. Marchetti-Spaccamela[1⋆], N. Megow[2], M. Skutella[3⋆⋆], and L. Stougie[4]

[1] University of Rome "La Sapienza", Italy. alberto.marchetti@dis.uniroma1.it
[2] Max-Planck-Institut für Informatik, Saarbrücken, Germany. nmegow@mpi-inf.mpg.de
[3] Technische Universität Berlin, Germany. skutella@math.tu-berlin.de
[4] Vrije Universiteit Amsterdam, CWI, Amsterdam, The Netherlands. stougie@cwi.nl

Abstract. We consider scheduling to minimize the weighted sum of completion times on a single machine that may experience unexpected changes in processing speed or even full breakdowns. We design a polynomial time deterministic algorithm that finds a robust prefixed scheduling sequence with a solution value within 4 times the value an optimal clairvoyant algorithm can achieve, knowing the disruptions in advance and even being allowed to interrupt jobs at any moment. A randomized version of this algorithm attains in expectation a ratio of e w.r.t. a clairvoyant optimum. We show that such a ratio can never be achieved by any deterministic algorithm by proving that the price of robustness of any such algorithm is at least $1+\sqrt{3} \approx 2.73205 > e$.
As a direct consequence of our results, the question whether a constant approximation algorithm exists for the problem with given machine unavailability periods is answered affirmatively. We complement this result by an FPTAS for the preemptive and non-preemptive special case with a single non-available period.

1 Introduction

In general if one offers a set of jobs to a machine then the production process on that machine is not usually influenced by the job owner. The machine may slow down due to simultaneous utilization by other users, it may break down completely for some time, or otherwise become unavailable for a particular user, etc. The only influence the job owner has is on the order in which he offers his jobs to be processed. Thus the quest for a schedule that is robust against machine calamities emerges. In this paper we study this scheduling problem when the objective is to minimize the sum of weighted completion times of the jobs. We aim to compute a robust scheduling sequence which performs well regardless of unexpected machine breakdowns or fluctuations in speed when comparing against an optimal clairvoyant algorithm.

More precisely, we are given a job set J with processing times $p_j \in \mathbb{R}^+$ and weights $w_j \in \mathbb{R}^+$ 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 of jobs π 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, unexpectedly, the machine may become unavailable or changes its processing speed. In case of a machine breakdown the currently running job is preempted and will be resumed processing at any later moment

⋆ Supported by EU project FRONTS and MUR FIRB project RBIN047MH9 Italy-Israel.
⋆⋆ Supported by the DFG Research Center MATHEON in Berlin.

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.

Our main results are a deterministic and a randomized robust, prefixed, order of the jobs, computable in polynomial time, such that scheduling the jobs in this order will always yield a solution that remains within multiplicative factor 4 from clairvoyant optimal for the deterministic order and within multiplicative factor e in expectation from optimal for the randomized order. We can adapt our algorithm to solve more general problem instances with certain types of precedence constraints without losing in the performance. These results are presented in Section 2.

We notice that in any version of the problem in which a prefixed sequence of jobs executed on machines with unknown behavior cannot guarantee to finish within the minimum makespan, an adversary would create an arbitrarily long breakdown at the moment that an optimal schedule has completed all jobs. For any such variation of our problem any (exponential time) algorithm will have an arbitrarily bad performance ratio. Examples of such variations are the problem with two or more machines instead of a single machine, the problem in which preempting or resuming a job requires (even the slightest amount of) extra work, or the problem in which jobs have individual release dates (even if all weights are equal).

To derive our results, we view the objective function as to minimize the total weight of uncompleted jobs at any time. A constant bound on the remaining weight when compared with the remaining weight of an optimal clairvoyant algorithm over all points in time gives a performance guarantee. We compute the job sequence iteratively from backwards: in each iteration we find a subset of jobs with largest total processing time such that their total weight stays below a certain weight bound that we can relate to an optimal value. We increase this bound in each iteration using the general idea of *doubling* which has proven to be a very useful method for designing approximation algorithms. It is not a precisely formulated technique but rather an idea that has been applied in various problem setting; for a collection of such examples we refer to [4].

Part of our performance ratio is inherent to measuring the value of a schedule computed having only partial information against that of a clairvoyant schedule, in a similar way as the competitive ratio of online algorithms [3]. Additionally we require a prefixed solution which may not be changed after receiving more information on the machine behavior. As in competitive analysis, we use adversarial sequences in Section 3 to show that no deterministically computed prefixed order can remain within a multiplicative factor of $1+\sqrt{3}$ from optimal, regardless of the algorithm's running time. Since $1+\sqrt{3} \approx 2.73205 > e$, this shows that randomized algorithms may produce prefixed orders that have essentially better expected performance ratios than their deterministic counterparts. We complement this result by a lower bound of 2 on the performance ratio of any randomized algorithm. We notice that such lower bounds have been called the *price of robustness* of the problem in [14].

Robust optimization has become a separate field of research in optimization and is mostly concentrating on robust continuous optimization problems, see [2] for a survey. The term *robust scheduling* in the literature refers mainly to robustness against uncertain processing times; see e.g. [11, chap. 7]. Generally, the term *robust* is not used consistently and recently many attempts have been made to weaken the notion of robustness to make

it more applicable. We emphasize, that our results are robust in the most conservative, classical notion of robustness originating by Soyster [18], which is also called *strict robustness* [14].

It may seem rather surprising that for our robust problem it is possible to remain within constant multiplicative factors from optimal. This is even more since our results immediately answer a major open question in the area of scheduling with limited machine availability, a subfield of machine scheduling that has been studied for over twenty years; see, e.g., the survey by Schmidt [16]. The question was if there exists a polynomial time constant approximation algorithm for the offline version of our problem, in which the breakdown periods are given in advance. Clearly, our results answer this question affirmatively. In fact, this problem is known to be strongly NP-hard [19], and it is weakly NP-hard [12] if there is only one such non-available period. However, if all jobs have equal weights, a simple interchange argument shows that sequencing jobs in non-increasing order of processing times is optimal as it is in the setting with continuous machine availability [17]. Obviously, this result immediately transfers to the robust setting in which machine breakdowns or changes in processing speeds are not known beforehand. In contrast, when job weights are arbitrary, natural greedy strategies perform arbitrarily bad in both, the robust and the offline, problem setting.

Our last result in Section 4 is a fully polynomial time approximation scheme (FPTAS) for the special problem setting of scheduling on a machine with a single unavailable period that is known a priori. It also solves the non-preemptive problem variant which is also weakly NP-hard [1, 13]. Thereby we improve on the previously best known algorithms for both, the preemptive and the non-preemptive problem, which yield approximation factors $1.618+\varepsilon$ [15] and 2 [8], respectively.

We notice that for the offline versions of our problem in which preemption is not allowed or causes extra work, 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.

2 A robust sequencing algorithm

Given a single machine that runs on unit speed, the completion time C_j^π of job j according to sequence π is

$$C_j^\pi := \sum_{k:\pi(k)\leq\pi(j)} p_k. \tag{1}$$

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, i.e., $W^\pi(t) := \sum_{j:C_j^\pi>t} w_j$. Then,

$$\sum_{j\in J} w_j C_j^\pi = \int_0^\infty W^\pi(t)dt. \tag{2}$$

If breaks or fluctuations in speed occur on the machine, (1) no longer describes the true job completion times. The more general setting with breaks and/or speed fluctuations can be described by a non-decreasing continuous function $f : \mathbb{R}_+ \to \mathbb{R}_+$ where $f(t)$ denotes the aggregated amount of processing time available on the machine up to time t. We

refer to f as the *machine capacity function* of the machine. If the derivative of f at time t exists, it can be interpreted as the speed of the machine at that point in time.

Let $S(\pi)$ denote the single machine schedule in which jobs are sequenced according to permutation π. The completion time of job j in this schedule is then given by

$$C_j^{S(\pi)} := \min\{t \mid f(t) \geq C_j^{\pi}\}. \tag{3}$$

For some point in time $t \geq 0$ let $W^{S(\pi)}(t)$ denote the total weight of jobs that are not yet completed by time t in schedule $S(\pi)$. Notice that $W^{S(\pi)}(t) = W^{\pi}(f(t))$. Then,

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

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

Observation 1 *For any given machine capacity function f,*

$$\int_0^{\infty} W^*(f(t))dt \tag{4}$$

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

We will present how to find a robust sequence of jobs π such that, no matter how the single machine behaves, the objective value of the corresponding schedule $S(\pi)$ is within a constant factor of the optimum.

Lemma 1. *Let π be a sequence of jobs and $c > 0$. Then, the objective value $\sum_{j \in J} w_j C_j^{S(\pi)}$ is at most c times the optimum for all machine capacity functions f if and only if*

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

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

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

We prove the "only if" part by contradiction. Assume that $W^{\pi}(t_0) > cW^*(t_0)$ for some t_0. For any $t_1 > t_0$ consider the following machine capacity function f:

$$f(t) = \begin{cases} t & \text{if } t \leq t_0, \\ t_0 & \text{if } t_0 < t \leq t_1, \\ t - t_1 + t_0 & \text{if } t > t_1\,, \end{cases}$$

which describes a breakdown of the machine in the time interval $[t_0, t_1]$. Then,

$$\sum_{j \in J} w_j C_j^{S(\pi)} = \sum_{j \in J} w_j C_j^{\pi} + (t_1 - t_0)W^{\pi}(t_0). \tag{5}$$

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

$$\sum_{j \in J} w_j C_j^{S(\pi^*)} = \sum_{j \in J} w_j C_j^{\pi^*} + (t_1 - t_0)W^*(t_0). \tag{6}$$

If t_1 goes to infinity, the ratio of (5) and (6) goes to $W^{\pi}(t_0)/W^*(t_0) > c$. This is a contradiction and concludes the proof. □

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 the lemma, we aim at approximating the minimum total weight of uncompleted jobs at any point in time, i.e., approximating the value of $W^*(t)$ for all values of $t \leq p(J)$. 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.

Algorithm D: For $i \in \{0, 1, \ldots, \lceil \log w(J) \rceil\}$ let J_i^* be a subset of jobs of total weight $w(J_i^*) \leq 2^i$ and maximum total processing time $p(J_i^*)$. Notice that $J^*_{\lceil \log w(J) \rceil} = J$. Construct a permutation π as follows. Start with an empty sequence of jobs. For $i = \lceil \log w(J) \rceil$ down to 1, append the jobs in $J_i^* \setminus \bigcup_{k=0}^{i-1} J_k^*$ in any order at the end of the sequence. Finally append the jobs in J_0^* in any order.

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

Proof. Using Lemma 1 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,

$$W^\pi(t) \leq \sum_{k=0}^{i} w(J_k^*) \leq \sum_{k=0}^{i} 2^k = 2^{i+1} - 1. \tag{7}$$

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 (7) 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 [9]. However, we adapt the algorithm by, instead of J_i^*, computing a subset of jobs J_i of total weight $w(J_i) \leq (1+\varepsilon)2^i$ 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 $1/\varepsilon$ adapting, e.g., the FPTAS in [6] for KNAPSACK. The subsets J_i obtained in this way are turned into a sequence π' as in Algorithm D.

Theorem 2. *Let $0 < \varepsilon \leq 1/(2^{\lceil \log w(J) \rceil + 1})$. For every scheduling instance, we can construct a permutation π in time polynomial in the input size and $1/\varepsilon$ such that the objective value $\sum_{j \in J} w_j C_j^{S(\pi)}$ is at most 4 times the optimum for all machine capacity functions f.*

Proof. Again, by Lemma 1 it is sufficient to prove that $W^{\pi}(t) < 4W^*(t)$ for all $t \geq 0$. Instead of inequality (7) we get the slightly weaker bound

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

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

The following example and Lemma 1 show that there are instances for which our algorithm yields a solution of value arbitrarily close to 3 times the optimal value. Since in the proof of the two theorems job weights are counted twice in case of overlapping job sets, it is not unlikely that the correct price of robustness is strictly less than 4.

Example 1. For a given set of n jobs with $w_j = p_j = 2^j$ for $j = 1, \ldots, n-1$ and $w_n = 2^{n-1}+2$ and $p_n = 2^n$, our algorithm computes the sequence $\pi = n, n-1, \ldots, 1$. Let $t = 2^n$. Observe that

$$W^{\pi}(t) = \sum_{j=1}^{n} w_j = \sum_{j=1}^{n-1} 2^j + 2^{n-1} + 2 = 3 \cdot 2^{n-1}.$$

Any other sequence π' with final job n has left at time t only n because $\sum_{j=1}^{n-1} p_i = 2^n - 2 < t$. Thus, $W^{\pi'} = w_n = 2^{n-1} + 2$. If n goes to infinity, the ratio $W^{\pi}(t)/W^{\pi'}(t)$ tends to 3.

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 $i \in \{0, 1, \ldots, \lceil \log w(J) \rceil\}$ 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 first, that the same arguments as in Lemma 1 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 $\mathbb{E}\left[\sum_{j \in J} w_j C_j^{S(\pi)}\right]$ is at most c times the optimum 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 $\mathbb{E}\left[W^{\pi(X)}(t)\right] < eW^*(t) - 1$ 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 (7) we can bound the expected total remaining weight of sequence $\pi(X)$ at time t by

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

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

The algorithm can be adapted in the same way as the deterministic algorithm to run in polynomial time, see the proof of Theorem 2.

Corollary 2 *Let $0 < \varepsilon \leq 1/e^{\lceil \log w(J) \rceil + 1}$. For every scheduling instance, the adapted randomized algorithm constructs a permutation π in time that is polynomial in the input size and $1/\varepsilon$ such that the objective value $\sum_{j \in J} w_j C_j^{S(\pi)}$ is in expectation at most e times the optimum for all machine capacity functions f.*

A natural generalization of the robust 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. To generalize our robust algorithm to scheduling instances with precedence constraints we need to find a way of adapting 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 [7] and even hard to approximate [5]. On the positive side, several POK problems with underlying partial orders that have a special structure can be approximated arbitrarily well. Such special cases are, e.g., directed outtrees, two dimensional orders, and the complement of chordal bipartite orders for which FPTASes are known [7, 10].

Theorem 4. *Let $\varepsilon > 0$. Consider the robust 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 we can choose an appropriate ε and construct a permutation π respecting $(J, \prec)$ in time polynomial in the input size and $1/\varepsilon$ such that the objective value is at most 4 times the optimum for all machine capacity functions f. A randomized algorithm finds a sequence with expected objective value bounded by e 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 robust sequencing problem with a given partial order $(J, \prec)$ and its reverse order $(J, \prec')$. We apply a slightly modified version of Algorithm D. 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. □

3 Lower bounds on the price of robustness

We construct a lower bound on the price of robustness of $1 + \sqrt{3} \approx 2.73 > e$ for any deterministic sequence of jobs. That is, independent of the sequencing of the jobs there exists an adversarial breakdown strategy such that the order yields an objective value which is at least $1 + \sqrt{3}$ times that of an order chosen optimally with respect to the breakdown strategy. This result, together with Corollary 2 implies that randomization yields intrinsically better performance ratios than deterministic algorithms.

Theorem 5. *No deterministic algorithm can produce a sequence of jobs paying a price of robustness less than $1 + \sqrt{3}$.*

Proof. Consider a sequence of 9 jobs with processing times $p_j = 2^{j-1}$, $j = 1,\ldots,9$, and weights $w_1,\ldots,w_9$ given by the sequence

$$1,\ 1+\sqrt{3},\ 3+2\sqrt{3},\ 7+4\sqrt{3},\ 14+8\sqrt{3},\ 26+15\sqrt{3},\ 45+26\sqrt{3},\ 71+41\sqrt{3},\ 97+56\sqrt{3}\,.$$

The processing times are chosen such that $\sum_{j=1}^{k-1} p_j < p_k$, for $k = 1,\ldots,9$. Any algorithm that chooses job i, $i = 1,\ldots,8$, to be the first job that is not selected to be amongst the i jobs to be scheduled last, will at time $\sum_{j=1}^{9} p_j - p_i$ face an adversarial breakdown of the machine and has at least a total weight of $\sum_{j=1}^{i-1} w_j + w_{i+1}$ of jobs to be completed, whereas the adversary has left only job i. It is a matter of simple calculations to verify that, for each $i = 1,\ldots,8$ the ratio $(\sum_{j=1}^{i-1} w_j + w_{i+1})/w_i \geq 1+\sqrt{3}$. If the algorithm decides otherwise, it schedules the jobs in sequence $9,8,\ldots,2,1$ and will face an adversarial machine breakdown at time $\sum_{j=1}^{9} p_j - p_9$ having a total weight of jobs to be completed of $\sum_{j=1}^{9} w_j$ against w_9 for the adversary. Again simple calculations show that $(\sum_{j=1}^{9} w_j)/w_9 \geq 1+\sqrt{3}$. □

We also design a first lower bound on the price of robustness for randomized algorithms.

Theorem 6. *No randomized algorithm can find a job sequence at a price of robustness less than* 2*.*

Proof. We use Yao's principle [20] to derive the lower bound. Our randomized instance consists of n jobs $J = \{1,\ldots,n\}$ with $p_j = w_j = 2^j$, for $j \in J$. There is a single huge breakdown of the machine occurring at a random point in time. With probability $\mathrm{pr}_j = 1/2^j$ the breakdown happens at time $p(J) - p_j$, for any $j = 1,\ldots,n-1$ (*scenario j*), and at time $p(J) - p_n$ with probability $\mathrm{pr}_n = 1/2^{n-1}$ (*scenario n*). Obviously $\sum_{j=1}^{n} \mathrm{pr}_j = 1$.

We assume that the machine breakdown lasts for a huge amount of time such that the objective function value for any sequence of jobs is completely dominated by the length of the breakdown times the total weight of jobs completed after the breakdown (see Lemma 1 and its proof). Thus, in the following we refer to the (expected) total remaining weight of a sequence at the time of the random machine breakdown simply as the *(expected) value* of that sequence.

If the breakdown starts at time $p(J) - p_j$, then in an optimal sequence for this scenario, job j is the final job with a total remaining weight w_j at that time. Thus, the expected value of an optimal (scenario-dependent) sequence is

$$\sum_{j=1}^{n-1} \frac{1}{2^j}\, 2^j + \frac{1}{2^{n-1}}\, 2^n \;=\; n+1\,.$$

We claim that the sequence $n, n-1,\ldots,2,1$ gives the minimum expected value for any prefixed deterministic sequence. Suppose this was true. Then the remaining weight at time $p(J) - p_j$ is $\sum_{k\leq j} p_k$ for any $j = 1,\ldots,n$ which gives an expected value of

$$\sum_{j=1}^{n-1} \frac{1}{2^j} \sum_{k=1}^{j} 2^k + \frac{1}{2^{n-1}} \sum_{k=1}^{n} 2^k \;=\; \sum_{j=1}^{n-1} \frac{2^{j+1}-2}{2^j} + \frac{2^{n+1}-2}{2^{n-1}} \;=\; 2n\,.$$

This gives the claimed lower bound since the ratio between the expected values of the best deterministic algorithm and an optimal offline solution tends to 2 when n goes to infinity.

It remains to prove the claim. We use a simple exchange argument. Assume there is an optimal sequence π which differs from sequence $n, n-1, \ldots, 2, 1$. Let $k < n$ be the job with smallest index that is not at the $(n-k+1)$-th position in π. Thus, the final subsequence of π is given by jobs $\ell, k-1, k-2, \ldots, 2, 1$ for some $\ell > k$. We remove job k from its current position and insert it between ℓ and $k-1$ and show that this change does not increase the expected objective value of the sequence. Denote the modified sequence by π'.

The change in the expected value of the sequence depends only on the possible breakdowns that have their start times $t_j := p(J) - p_j$, for $j = 1, \ldots, n$, within the time interval $[t, t')$ with $t := C_k^{\pi}$ and $t' := C_\ell^{\pi}$. We express the change in the expected value via the remaining weight at times $t_j \in [t, t')$. Notice that $t_j \notin [t, t')$ for $j = 1, \ldots, k-1$. The insertion of job k adds w_k to the remaining weight at any $t_j \in [t, t')$; on the other hand, the contribution of job ℓ with weight $w_\ell \geq 2w_k$ at t_k is removed. Thus the change in the expected value is

$$\sum_{j:t_j \in [t,t')} \mathrm{pr}_j \left(W^{\pi'}(t_j) - W^{\pi}(t_j) \right) = \sum_{j:t_j \in [t,t')} \mathrm{pr}_j w_k - \mathrm{pr}_k w_\ell \leq \sum_{j:t_j \in [t,t')} \mathrm{pr}_j w_k - \mathrm{pr}_k 2w_k$$
$$\leq w_k \left(\sum_{j=k}^{n} \mathrm{pr}_j - 2\,\mathrm{pr}_k \right) = 0\,.$$

Applying this argument iteratively proves the claim and the theorem. □

4 The offline problem

Clearly, the approximation results in Section 2 hold in the offline version of our problem in which machine breakdowns and changes in speed are known in advance. Thus, we provide the first constant factor approximation algorithms for preemptive scheduling with fixed non-availability periods as a byproduct. In this section we consider the special case of the offline problem in which the machine has a single non-availability interval $[s,t]$ for $1 \leq s < t$. We derive a fully polynomial time approximation scheme (FPTAS) for the preemptive and non-preemptive problem variant which improve on the currently best known approximation results $1.618 + \varepsilon$ [15] and 2 [8]. Due to space restrictions we focus on the non-preemptive case in this extended abstract.

Non-preemptive dynamic program (DP). Given a non-available time interval $[s,t]$ we compute an optimal non-preemptive schedule for a given set of n jobs. The jobs must be partitioned into jobs that complete before s and jobs that complete after t. Clearly, the jobs in each individual set are scheduled in WSPT order, that is, in non-increasing order of ratios w_j/p_j. This order is known to be optimal [17] on a continuously processing machine. Let the jobs be indexed in WSPT order, and assume that the total processing time exceeds the length of the available period before the break, i.e., $\sum_{j=1}^{n} p_j > s$.

The dynamic program generates a state $[k, v, y]$ if there is a feasible schedule of jobs $1, \ldots, k$ with total value $v := \sum_{j=1}^{k} w_j C_j$ and in which the amount of processing time used in the interval before the break, $[0, s]$, is y. The dynamic program starts with the state $[0, 0, 0]$ and computes all states by moving from any state $[j-1, v, y]$ to one or two

new states $[j, v', y']$. The first possibility is to schedule job j before the break, that is,

$$v' = v + (y + p_j)w_j \quad \text{and} \quad y' = y + p_j, \tag{8}$$

provided that $y' \leq s$. The second possibility is to schedule j after the break $[s,t]$, that is,

$$v' = v + (t + \sum_{i=1}^{j} p_i - y)w_j \quad \text{and} \quad y' = y. \tag{9}$$

An optimal schedule can be obtained by finding a state $[n, v, y]$ with minimum v and backtracking from that state. Since the v-values are bounded by $V := \sum_{j=1}^{n} w_j(t + \sum_{k=1}^{j} p_k)$ and the y-values are bounded by s, the running time of this dynamic programming algorithm is $\mathcal{O}(nVs)$.

Non-preemptive FPTAS. In a fully polynomial time algorithm, we can neither afford to consider all possible objective values v, nor can we consider all possible total processing times before the break, y. Using standard rounding techniques, the number of occurring v-values can be reduced to a number which is polynomially bounded in the input size and $1/\varepsilon$, at the cost of increasing the v-values occurring in the dynamic program by a factor at most $(1+\varepsilon)$. (For a given parameter $\varepsilon > 0$, round up objective values to the nearest multiple of $\varepsilon Z_{LB}/n$ for some reasonable lower bound Z_{LB}; we skip further details.)

The main challenge here is to discretize the range of values y in an appropriate way. Notice that we cannot afford to round y-values since they contain critical information on how much processing time remains before the break. Perturbing this information causes a considerable change in the set of feasible schedules. In this way one might lose optimal schedules or introduce infeasible schedules with too much processing before the break. Both effects cannot be controlled easily, and thus, must be avoided.

The intuition behind the following algorithm is to reduce the number of states by removing those with the same (rounded) objective value and nearly the same total processing time before the break. Among them, we want to store those with smallest amount of processing before the break in order to make sure that enough space remains for further jobs that need to be scheduled there.

Algorithm F:

1. For an arbitrary given $\varepsilon > 0$ let $\delta := \varepsilon/n$.
2. Partition the interval $[0,s]$ into sub-intervals I_i of length $s\delta$ for $i = 1, \ldots, n/\varepsilon$; the last interval may be smaller.
3. Run the dynamic program DP with the following modification. Among the states for the same job set and the same (rounded) objective value v, we store at most one for each interval I_i, namely the one with currently minimum y-value within I_i.

Lemma 2. *Suppose the algorithm DP on an instance with n jobs finds a chain of states*[5] $[0,0,0], [1, v_1^*, y_1^*], \ldots, [n, v_n^*, y_n^*]$. *Then Algorithm F finds for each* $j = 1, \ldots, n$ *a state* $[j, v_j, y_j]$ *with*

$$y_j^* - js\delta \leq y_j \leq y_j^* \qquad \text{and} \qquad v_j \leq (1 + (j-1)\delta)\, v_j^*. \tag{10}$$

[5] *Chain* of states means that, for $j = 0, \ldots, n-1$, state $[j+1, v_{j+1}^*, y_{j+1}^*]$ is obtained from $[j, v_j^*, y_j^*]$ by adding job $j+1$ according to (8) or (9).

In (10) we state an upper and a lower bound on y_j in terms of y_j^*. This will turn out to be important when proving the bound on v_j in terms of v_j^* in the following analysis.

Proof. We give a proof by induction. For $j = 1$ we store at most two states which obviously fulfill both conditions in (10).

Suppose the lemma is true for $j = i$. Consider state $[i+1, v_{i+1}^*, y_{i+1}^*]$ that was obtained from $[i, v_i^*, y_i^*]$ according to (8) or (9). We distinguish the two cases.

First case: If state $[i+1, v_{i+1}^*, y_{i+1}^*]$ was obtained from $[i, v_i^*, y_i^*]$ by adding job $i+1$ before the break, then Algorithm F when doing the same while processing $[i, v_i, y_i]$ yields $y = y_i + p_{i+1}$ and $v_{i+1} = v_i + (y_i + p_{i+1})w_{i+1}$. However, we cannot guarantee that this state will survive, because we might find a partial solution with the same objective value v_{i+1} but smaller y_{i+1} within the same subinterval I_k that contains y. But in this case $y - y_{i+1}$ is bounded from above by the length of interval I_k and thus by $s\delta$. Thus,

$$y_{i+1} \geq y_i + p_{i+1} - s\delta \geq y_i^* - is\delta + p_{i+1} - s\delta = y_{i+1}^* - (i+1)s\delta \ .$$

Moreover,

$$v_{i+1} = v_i + (y_i + p_{i+1})w_{i+1} \leq (1 + (i-1)\delta)\, v_i^* + (y_i^* + p_{i+1})w_{i+1} \leq (1 + i\delta)\, v_{i+1}^* \, .$$

Second case: If state $[i+1, v_{i+1}^*, y_{i+1}^*]$ was obtained from $[i, v_i^*, y_i^*]$ by adding $i+1$ after the break, then $y_{i+1}^* = y_i^*$ and Algorithm F finds a state $[i+1, y, v]$ such that $y = y_i$. If this state is later replaced by state $[i+1, y_{i+1}, v_{i+1}]$, where $y_{i+1} < y$ belongs to the same interval I_k as y, we still get

$$y_{i+1}^* \geq y_{i+1} \geq y_i - s\delta \geq y_{i+1}^* - (i+1)s\delta \ .$$

Moreover,

$$\begin{aligned} v_{i+1} &= v_i + (t + \sum_{j=1}^{i+1} p_j - y_i)w_{i+1} \leq (1 + (i-1)\delta)\, v_i^* + (t + \sum_{j=1}^{i+1} p_j - (y_i^* - is\delta))w_{i+1} \\ &\leq (1 + (i-1)\delta)\, v_i^* + (t + \sum_{j=1}^{i+1} p_j - y_i^* + it\delta)w_{i+1} \\ &\leq (1 + (i-1)\delta)\, v_i^* + (1 + i\delta)(t + \sum_{j=1}^{i+1} p_j - y_i^*)w_{i+1} \ \leq \ (1 + i\delta)\, v_{i+1}^* \, . \end{aligned}$$

The last inequality holds since $v_{i+1}^* = v_i^* + (t + \sum_{j=1}^{i+1} p_j - y_i^*)w_{i+1}$. □

As a consequence of Lemma 2, we can now state the main result of this section.

Theorem 7. *There exists an FPTAS for non-preemptive scheduling to minimize $\sum w_j C_j$ on a single machine that is not available during a given time interval $[s,t]$.*

The algorithm can be extended to an FPTAS for the preemptive problem. W.l.o.g. we can assume that in an optimal solution there is at most one job interrupted by the break $[s,t]$ and it resumes processing as soon as the machine is available again. Denote this job as the *split job*. Generally speaking, we extend the Algorithm F in such a way that the dynamic program in Step 3 is started for each potential split job $k = 1 \ldots, n$ and any of its possible positions. That is for any completion time $C_k \in (t, t + p_k)$ which defines a new non-available period $[s', t']$ with $s' = C_k - (t - s) - p_k$ and $t' = C_k$. Standard rounding of possible completion times to the nearest power of $1 + \varepsilon'$ gives the desired new FPTAS.

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. A. Ben-Tal and A. Nemirovski. Robust solutions of linear programming problems contaminated with uncertain data. *Mathematical Programming*, 88:411–424, 2000.
3. A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
4. M. Chrobak and C. Kenyon-Mathieu. Sigact news online algorithms column 10: Competitiveness via doubling. *SIGACT News*, 37(4):115–126, 2006.
5. 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.
6. 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.
7. 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.
8. 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.
9. 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.
10. S. G. Kolliopoulos and G. Steiner. Partially ordered knapsack and applications to scheduling. *Discrete Applied Mathematics*, 155(8):889–897, 2007.
11. P. Kouvelis and G. Yu. *Robust Discrete Optimization and Its Applications*. Springer, 1997.
12. C.-Y. Lee. Machine scheduling with an availability constraint. *Journal of Global Optimimization*, 9:395–416, 1996.
13. C.-Y. Lee and S. D. Liman. Single machine flow-time scheduling with scheduled maintenance. *Acta Informatica*, 29(4):375–382, 1992.
14. C. Liebchen, M. Lübbecke, R. H. Möhring, and S. Stiller. Recoverable robustness. Technical report ARRIVAL-TR-0066, ARRIVAL Project, 2007.
15. N. Megow and J. Verschae. Note on scheduling on a single machine with one non-availability period. Unpublished, 2008.
16. G. Schmidt. Scheduling with limited machine availability. *European Journal of Operational Research*, 121(1):1–15, 2000.
17. W. E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3:59–66, 1956.
18. A. Soyster. Convex programming with set-inclusive constraints and applications to inexact linear programming. *Operations Research*, 21(4):1154–1157, 1973.
19. 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.
20. A. C.-C. Yao. Probabilistic computations: toward a unified measure of complexity (extended abstract). In *Proceedings of the 18th FOCS*, pages 222–227, 1977.