Scheduling uniform machines on-line requires nondecreasing speed ratios

Citation for published version (APA):

Vestjens, A. P. A. (1994). *Scheduling uniform machines on-line requires nondecreasing speed ratios*. (Memorandum COSOR; Vol. 9435). Technische Universiteit Eindhoven.

Document status and date:

Published: 01/01/1994

Document Version:

Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

[Link to publication](#)

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy

If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

EINDHOVEN UNIVERSITY OF TECHNOLOGY
Department of Mathematics and Computing Science

Memorandum COSOR 94-35

**Scheduling uniform machines on-line
requires nondecreasing speed ratios**

A.P.A. Vestjens

Eindhoven, November 1994
The Netherlands

Scheduling uniform machines on-line requires nondecreasing speed ratios

Arjen Vestjens

Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O.Box 513, 5600 MB, Eindhoven
arjenv@win.tue.nl

Abstract

We consider the following on-line scheduling problem. We have to schedule n independent jobs, where n is unknown, on m uniform parallel machines so as to minimize the makespan; preemption is allowed. Each job becomes available at its release date, and this release date is not known beforehand; its processing requirement becomes known at its arrival. We show that if only a finite number of preemptions is allowed, then there exists an algorithm that solves the problem if and only if $s_{i-1}/s_i \leq s_i/s_{i+1}$ for all $i = 2, \dots, m-1$, where s_i denotes the i th largest machine speed. We also show that if this condition is satisfied, then $O(mn)$ preemptions are necessary, and we provide an example to show that this bound is tight.

Keywords: on-line algorithms, preemptive scheduling, uniform machines.

1 Introduction

We consider a scheduling problem in which n independent jobs $J_1, \dots, J_n$ have to be processed on m parallel machines $M_1, \dots, M_m$, where n is unknown. Each machine can handle at most one job at a time and each job can be executed on at most one machine at a time. Job j becomes available for processing at its release date r_j , which is not known beforehand; it has a processing requirement p_j , which becomes known at its release date. Preemption is allowed, i.e., the processing of any job may arbitrarily often be interrupted and resumed at the same time on a different machine or at a later time on any machine. The machines are assumed to be uniform, i.e., the speed s_i with which M_i executes the jobs may differ per machine; complete execution of J_j on M_i requires p_j/s_i time units. We assume that the machines are ordered according to nonincreasing speeds. Our objective is to minimize the maximum completion time; note that, since at any point in time only information is available about the jobs that have arrived, we have to resort to *on-line* algorithms.

In our model, all jobs have release dates, which is not the case in all of the on-line scheduling literature. Another situation that has been analyzed is the one in which the sequence of all jobs is available at the beginning, but the jobs are not known a priori, i.e., J_j remains unknown until J_{j-1} has been scheduled. As soon as J_j appears, it must irrevocably be assigned to one or more time slots of one or more machines (e.g., see [Bartel, Fiat, Karloff

& Vohra, 1992; Chen, Van Vliet & Woeginger, 1994]). Also the fact that the assignment is irrevocable is different from our model, because we assume that the remaining processing time of a job can entirely be rescheduled at any point in time after its arrival.

A special case of our variant concerns the case in which the machines are assumed to be identical and the objective is to schedule all jobs to meet a common deadline. Hong & Leung [1992] prove that for this variant it is possible to create a feasible schedule on-line, if such a schedule exists. Their algorithm can also be used to minimize the makespan, because it only uses the knowledge of the value of the deadline to stop the algorithm in an early stage if possible. Labetoulle, Lawler, Lenstra & Rinnooy Kan [1984] present a nearly on-line algorithm that solves the uniform machine case, where nearly on-line refers to the situation that it is known when the next job will arrive. The existence of this algorithm implies that if jobs can only arrive at given points in time, then we can solve our problem by just considering every such moment as a possible arrival time.

In Section 2 we show that if the machines are capable of *processor sharing*, then we can solve our problem without time-discretization. By processor sharing we mean that the processing capacity of a given subset of machines can be equally shared by a given subset of jobs; each job can still only be executed on at most one machine at a time. Note that processor sharing requests an infinite number of preemptions. In Section 3 we prove that, for each instance, all optimal schedules have the same structure at any time. This will be the basis of the rest of the paper. In Section 4, we give a necessary and sufficient condition on the machine speeds for which processor sharing is not needed to maintain the optimal structure and thereby an optimal schedule. Finally, in Section 5, we show that the number of preemptions necessary for optimality is $O(mn)$, if the conditions derived in Section 4 are satisfied. We also provide an example which shows that this bound is tight.

2 An algorithm with processor sharing

We mentioned above that if processor sharing is allowed, then the problem can be solved to optimality. This result is closely related to the nearly on-line model considered by Labetoulle et al. The key observation behind their analysis is that the remaining processing requirements should be as evenly distributed as possible at any arrival time. More specifically, their nearly on-line algorithm schedules the jobs such that *all* the partial sums $\sum_{j=1}^l p_j(t)$ ($1 \leq l \leq n$) are minimized at any arrival, where $p_j(t)$ denotes the processing requirement of J_j that still has to be performed at time t ; the jobs are assumed to be ordered such that $p_1(t) \geq p_2(t) \geq \dots \geq p_n(t)$ at any point in time t . Each time that a job arrives, the algorithm determines which part of each job has to be processed before the next arrival such that all partial sums are minimal when the next job arrives. After that, it schedules these parts in the interval between the two release dates by using the Gonzalez-Sahni algorithm for $Q|pmtn|C_{\max}$ [Gonzalez & Sahni, 1978]. The notation we use to describe scheduling problems is the same as Lawler, Lenstra, Rinnooy Kan & Shmoys [1993]. The optimality of this strategy follows from the well-known result of Horvath, Lam & Sethi [1977] that at time t the minimum time needed for completing the n currently available jobs is given by

$$\max_{1 \leq l \leq n} \frac{\sum_{j=1}^l p_j(t)}{\sum_{j=1}^l s_j}. \quad (1)$$

For simplicity of exposition, we assume that there are at least as many machines as there are jobs. This can be achieved by adding $\max\{0, n - m\}$ dummy machines with processing speed zero. Condition 1 implies that an on-line algorithm is optimal if it minimizes all partial

sums $\sum_{j=1}^l p_j(t)$ for all available jobs at any time t . This boils down to considering any time as a possible arrival time.

The on-line algorithm in which processor sharing is allowed, which from now on is denoted by $\mathcal{I}$, has the property that at any time it processes the k longest jobs exclusively on the k fastest machines. As a result of this property, algorithm $\mathcal{I}$ minimizes all partial sums at any time, and therefore it produces optimal schedules. The property also implies that the relative order of the jobs does not change, i.e., for every pair of jobs, the processing requirement of one of the jobs is greater than or equal to the processing requirement of the other job during its total execution. The key observation behind this algorithm is that it is necessary to determine when jobs become of equal size; from that moment on, they should share the machines they occupy. Horvath et al. make this observation for the corresponding off-line problem without release dates. They prove that an algorithm produces optimal schedules when based upon this rule. The algorithm $\mathcal{I}$ is the same as the first phase of their algorithm if all jobs have the same release date.

Algorithm $\mathcal{I}$ keeps jobs equally sized from the moment they become equal until they are completed. This is possible due to processor sharing. If processor sharing is not allowed, then it is impossible to keep jobs equally sized and thereby to minimize all the partial sums at any time. Hence, having no additional information about future requests may cause difficulties in determining an optimal schedule.

3 Structure of the optimal solution

In the literature, on-line algorithms are often studied in a sort of request-answer game. In such a game an adversary makes a sequence of requests, which are served one by one by the on-line algorithm. In this paper, we take the place of the adversary, who tries to create instances for which a given on-line algorithm fails to generate an optimal schedule. For creating such instances it is important to study the structure of optimal schedules. If no more jobs would arrive after time t , then the schedule created by algorithm $\mathcal{I}$ (for which processor sharing is allowed) has a staircase structure as presented in Figure 1. The schedule consists of q

Figure 1: Structure of an optimal schedule.

blocks $B_1(t), \dots, B_q(t)$, where block $B_k(t)$, consisting of the jobs $J_{b_{k-1}+1}, \dots, J_{b_k}$, occupies the machines $M_{b_{k-1}+1}, \dots, M_{b_k}$; the index b_k of the last job and the length t_k of block $B_k(t)$

are determined by

$$t_k = \max_{b_{k-1}+1 \leq l \leq n} \frac{\sum_{j=b_{k-1}+1}^l p_j(t)}{\sum_{j=b_{k-1}+1}^l s_j} \text{ and } b_k = \max \left\{ l \mid \frac{\sum_{j=b_{k-1}+1}^l p_j(t)}{\sum_{j=b_{k-1}+1}^l s_j} = t_k \right\}.$$

These values can be determined recursively in the order $t_1, b_1, t_2, \dots, b_q$, and the number of blocks, q , in which the schedule is partitioned, follows from this recursion scheme. If we represent a feasible schedule by a vector of finish times of the machines, where the machines are ordered according to nonincreasing finish times, then it is possible to order the schedules lexicographically. The schedule created by algorithm $\mathcal{I}$ is the smallest schedule among all feasible schedules, using this lexicographical order; this follows directly from the fact that $\mathcal{I}$ minimizes all partial sums $\sum_{j=1}^l p_j(t)$ at any time. Lemma 1 states that in order to solve $Q|pmtn, r_j|C_{\max}$ to optimality any on-line algorithm should produce a schedule that possesses the same structure as the one created by algorithm $\mathcal{I}$ at any point in time.

Lemma 1. *If there exists a time t for which the structure of the schedule created by an arbitrary on-line algorithm $\mathcal{A}$ differs from the structure of the schedule created by algorithm $\mathcal{I}$, then the adversary can force the algorithm $\mathcal{A}$ to get a non-optimal solution.*

Proof. Define the workload of a machine at any point in time as the total amount of work that still has to be executed on this machine if no more jobs were to arrive. Let $\bar{L}_1(t), \dots, \bar{L}_m(t)$ and $L_1(t), \dots, L_m(t)$ denote the workloads at time t in the schedule created by algorithm $\mathcal{I}$ and algorithm $\mathcal{A}$, respectively. Notice that the total amount of processing still to be carried out by algorithm $\mathcal{I}$ at time t is always less than or equal to the amount that has to be processed by algorithm $\mathcal{A}$, since $\mathcal{I}$ minimizes the total amount of processing left at time t . This implies that if there is a machine to which $\mathcal{I}$ assigns a bigger workload than $\mathcal{A}$, then there is also a machine for which the opposite holds.

Suppose that $L_1(t) = \bar{L}_1(t), \dots, L_{k-1}(t) = \bar{L}_{k-1}(t)$, and $L_k(t) > \bar{L}_k(t)$. From the construction of the schedule by algorithm $\mathcal{I}$ it follows that at time $t_0 = \bar{L}_k(t)/s_k$ only the machines M_1 up to M_{k-1} have workloads greater than zero. In the schedule created by algorithm $\mathcal{A}$, however, not only the workloads on the first $k-1$ machines are equal to the workloads in the schedule created by $\mathcal{I}$, but also M_k has a workload greater than zero. This implies that the total amount of processing still to be carried out by algorithm $\mathcal{A}$ is greater than the total amount of processing still to be carried out by $\mathcal{I}$ at time t_0 . If at time t_0 , m jobs arrive with sizes $p_j(t_0) = (\bar{L}_1(t_0)/s_1)s_j - \bar{L}_j(t_0)$ for $1 \leq j \leq m$, then the length of the schedule created by $\mathcal{I}$ remains the same, whereas the length of the schedule created by $\mathcal{A}$ increases.

Now suppose that $L_1(t) = \bar{L}_1(t), \dots, L_{k-1}(t) = \bar{L}_{k-1}(t)$, and $L_k(t) < \bar{L}_k(t)$. Then there exists a machine M_l with $l > k$ for which $L_l(t)/s_l > \bar{L}_k(t)/s_k$, since otherwise algorithm $\mathcal{I}$ would not create the smallest feasible schedule. Again, if we wait until time $t_0 = \bar{L}_k(t)/s_k$, then $L_1(t_0) = \bar{L}_1(t_0), \dots, L_{k-1}(t_0) = \bar{L}_{k-1}(t_0)$, $L_l(t_0) > \bar{L}_l(t_0)$, and all other loads in the schedule created by $\mathcal{I}$ are equal to zero. Analogously to the foregoing part, the adversary can cause algorithm $\mathcal{A}$ to fail. $\square$

Lemma 1 proves that the structure of the schedule constructed by algorithm $\mathcal{I}$ is the only possible structure to ensure that optimality is maintained at all times. Hence, the adversary can cause an on-line algorithm to fail only if it can force the algorithm to create a schedule with a different structure. Because of this, we refer to the structure of the schedule constructed by algorithm $\mathcal{I}$ as the optimal structure.

4 Influence of speed ratios

In this section, we show that there is a necessary and sufficient condition on the machine speeds, for which there exists an on-line algorithm that cannot be made to fail in finding an optimal solution, even if only a polynomial number of preemptions is allowed. This excludes processor sharing.

As we showed in Section 3, every optimal schedule has the same structure at any time. Hence, if an on-line algorithm creates a schedule that possesses this structure at every moment, then it is guaranteed to produce an optimal schedule. This implies that an algorithm produces optimal schedules if the arrival of jobs does not cause a deviation from the optimal structure.

Given a schedule created by an on-line algorithm, the adversary has to find a time t at which the arrival of a specific job will cause the algorithm to deviate from the structure obtained by algorithm $\mathcal{I}$. As Lemma 2 shows, it is not always possible for the adversary to do this.

Lemma 2. *There exists an on-line algorithm that solves $Q|pmtn, r_j|C_{\max}$ to optimality without processor sharing if*

$$\frac{s_{l-1}}{s_l} \leq \frac{s_l}{s_{l+1}} \quad \text{for all } 1 < l < m. \quad (2)$$

Proof. Consider an arbitrary schedule with the optimal structure after n jobs have arrived. We want to determine how the structure changes if at an arbitrary time t a job of size p arrives. Suppose that the k th block is the first one that is going to differ from the current structure. The change is caused by the new job so the length of the block is equal to

$$\max_{b_{k-1}+1 \leq l \leq n+1} \frac{p + \sum_{j=b_{k-1}+1}^{l-1} p_j(t)}{\sum_{j=b_{k-1}+1}^l s_j},$$

where $s_{n+1} = 0$ if $n+1 > m$. Suppose that the largest index for which the maximum is reached is l^* . Then we clearly have that

$$\begin{aligned} \frac{p + \sum_{j=b_{k-1}+1}^{l^*-1} p_j(t)}{\sum_{j=b_{k-1}+1}^{l^*} s_j} &\geq \frac{p + \sum_{j=b_{k-1}+1}^{l_1-1} p_j(t)}{\sum_{j=b_{k-1}+1}^{l_1} s_j} \quad \text{for all } l_1, b_{k-1}+1 \leq l_1 \leq l^*, \text{ and} \\ \frac{p + \sum_{j=b_{k-1}+1}^{l^*-1} p_j(t)}{\sum_{j=b_{k-1}+1}^{l^*} s_j} &> \frac{p + \sum_{j=b_{k-1}+1}^{l_2-1} p_j(t)}{\sum_{j=b_{k-1}+1}^{l_2} s_j} \quad \text{for all } l_2, l^* < l_2 \leq n+1. \end{aligned} \quad (3)$$

These conditions impose upper and lower bounds on the size p of the new job. As we show in Appendix A, the conditions for the existence of a job for which Condition 3 holds can be written as

$$\sum_{j=l_1+1}^{l^*} s_j \sum_{j=l^*}^{l_2-1} p_j(t) < \sum_{j=l^*+1}^{l_2} s_j \sum_{j=l_1}^{l^*-1} p_j(t) \quad \text{for all } b_{k-1}+1 \leq l_1 \leq l^* < l_2 \leq n+1. \quad (4)$$

Now suppose that $b_{k_1-1}+1 < l^* \leq b_{k_1}$ for some $k_1 \geq k$, which implies that the block contains only a part of block k_1 of the current structure. Then Condition 4 should certainly hold for

$l_1 = b_{k_1-1} + 1$ and $l_2 = b_{k_1} + 1$. For these specific l_1 and l_2 the condition becomes

$$\sum_{j=b_{k_1-1}+2}^{l^*} s_j \sum_{j=l^*}^{b_{k_1}} p_j(t) < \sum_{j=l^*+1}^{b_{k_1}+1} s_j \sum_{j=b_{k_1-1}+1}^{l^*-1} p_j(t). \quad (5)$$

This condition is satisfied most easily if within the current block B_{k_1} the partial sum of the first $l^* - b_{k_1} - 1$ jobs is maximal and the partial sum of the other jobs in this block is minimal, i.e., if

$$\sum_{j=b_{k_1}+1}^{l^*-1} p_j(t) = t_{k_1} \sum_{j=b_{k_1}+1}^{l^*-1} s_j \text{ and } \sum_{j=l^*}^{b_{k_1}} p_j(t) = t_{k_1} \sum_{j=l^*}^{b_{k_1}} s_j,$$

where t_{k_1} is the length of block k_1 . After inserting these values for the $p_j(t)$'s, Condition 5 becomes

$$\frac{\sum_{j=b_{k_1-1}+1}^{l^*-1} s_j}{\sum_{j=b_{k_1-1}+2}^{l^*} s_j} > \frac{\sum_{j=l^*}^{b_{k_1}} s_j}{\sum_{j=l^*+1}^{b_{k_1}+1} s_j}.$$

But Condition 2 implies that

$$\frac{\sum_{j=b_{k_1-1}+1}^{l^*-1} s_j}{\sum_{j=b_{k_1-1}+2}^{l^*} s_j} \leq \dots \leq \frac{s_{l^*-2} + s_{l^*-1}}{s_{l^*-1} + s_{l^*}} \leq \frac{s_{l^*-1}}{s_{l^*}} \leq \frac{s_{l^*}}{s_{l^*+1}} \leq \frac{s_{l^*} + s_{l^*+1}}{s_{l^*+1} + s_{l^*+2}} \leq \dots \leq \frac{\sum_{j=l^*}^{b_{k_1}} s_j}{\sum_{j=l^*+1}^{b_{k_1}+1} s_j},$$

which is a contradiction. It follows that the maximum is reached at the end of a block of the current structure, i.e., $l^* = b_{k_1-1} + 1$ for some $k_1 \geq k$. Since this is true for all possible $p_j(t)$ values within the bounds induced by the current structure, we know that the block that includes the new job is the same as the one in the schedule constructed by $\mathcal{I}$.

For the rest of the schedule to be equal to the one constructed by $\mathcal{I}$, it is sufficient to prove that all new blocks only include total current blocks and never just a part of a current block. Note that for the jobs from every current block $k_2 \geq k_1 + 1$ we have

$$\frac{\sum_{j=b_{k_2}+1}^{l-1} p_j(t)}{\sum_{j=b_{k_2}+2}^l s_j} = \frac{\sum_{j=b_{k_2}+1}^{l-1} p_j(t)}{\sum_{j=b_{k_2}+1}^{l-1} s_j} \frac{\sum_{j=b_{k_2}+1}^{l-1} s_j}{\sum_{j=b_{k_2}+2}^l s_j} \leq \frac{\sum_{j=l}^{b_{k_2}} p_j(t)}{\sum_{j=l}^{b_{k_2}} s_j} \frac{\sum_{j=l}^{b_{k_2}} s_j}{\sum_{j=l+1}^{b_{k_2}+1} s_j} = \frac{\sum_{j=l}^{b_{k_2}} p_j(t)}{\sum_{j=l+1}^{b_{k_2}+1} s_j}.$$

This implies directly that the new blocks are composed of complete blocks of the current structure. Hence, under Condition 2 it is possible for an on-line algorithm to maintain the optimal structure without the need of processor sharing. $\square$

Lemma 2 shows that processor sharing is not always necessary for an on-line algorithm to create optimal schedules. Lemma 3 shows that Condition 2 is not only sufficient, but also necessary to guarantee optimality without processor sharing.

Lemma 3. *There does not exist an on-line algorithm that solves $Q|pmt_n, r_j|C_{\max}$ to optimality in case processor sharing is not allowed if for some index l ($1 < l < m$)*

$$\frac{s_{l-1}}{s_l} > \frac{s_l}{s_{l+1}}. \quad (6)$$

Proof. Suppose that l^* is the first index for which $s_{l-1}/s_l > s_l/s_{l+1}$. Consider the following instance:

$$p_j = \begin{cases} rs_j & \text{for all } 1 \leq j \leq l^* - 2, \\ \frac{1}{2}(s_{l^*-1} + s_{l^*}) & \text{for all } l^* - 1 \leq j \leq l^*, \end{cases}$$

where r is some constant satisfying

$$r > \frac{s_{l^*} + s_{l^*+1}}{2s_{l^*+1}}.$$

Algorithm $\mathcal{I}$ assigns job J_j to machine M_j ($j = 1, \dots, l^* - 2$), and it assigns the jobs J_{l^*-1} and J_{l^*} to both machine M_{l^*-1} and M_{l^*} . Any on-line algorithm $\mathcal{A}$ has to make the same assignment to solve the problem to optimality. But since processor sharing is not allowed, algorithm $\mathcal{A}$ cannot decrease the processing requirements of jobs J_{l^*-1} and J_{l^*} equally fast. Hence, from a certain time t_0 onwards, M_{l^*-1} will only process J_{l^*-1} , and M_{l^*} will only process J_{l^*} . If we denote the remaining processing times of the jobs in the schedule created by $\mathcal{I}$ and $\mathcal{A}$ by $\bar{p}_j(t_0)$ and $p_j(t_0)$, respectively, then

$$\bar{p}_j(t_0) = \begin{cases} (r - t_0)s_j & \text{for all } 1 \leq j \leq l^* - 2, \\ \frac{1}{2}(1 - t_0)(s_{l^*-1} + s_{l^*}) & \text{for all } l^* - 1 \leq j \leq l^*, \end{cases}$$

and

$$p_j(t_0) = \begin{cases} (r - t_0)s_j & \text{for all } 1 \leq j \leq l^* - 2, \\ (1 - t_0)s_j & \text{for all } l^* - 1 \leq j \leq l^*. \end{cases}$$

At time t_0 , a job of size $p = (1 - t_0) \frac{(s_{l^*-1} + s_{l^*})s_{l^*+1}}{s_{l^*} + s_{l^*+1}}$ arrives. If we determine the structure of both schedules, then it follows that

$$\max_{1 \leq l \leq l^*+1} \left\{ \frac{\max\{p_l(t_0), p\} + \sum_{j=1}^{l-1} p_j(t_0)}{\sum_{j=1}^l s_j} \right\} = (r - t_0) = \max_{1 \leq l \leq l^*+1} \left\{ \frac{\max\{\bar{p}_l(t_0), p\} + \sum_{j=1}^{l-1} \bar{p}_j(t_0)}{\sum_{j=1}^l s_j} \right\}.$$

Both maxima are reached for $l = l^* - 2$, and the new job is not included in the first block. But it also follows that

$$\begin{aligned} \max_{l^*-1 \leq l \leq l^*+1} \left\{ \frac{\max\{p_l(t_0), p\} + \sum_{j=l^*-1}^{l-1} p_j(t_0)}{\sum_{j=l^*-1}^l s_j} \right\} &= \frac{p + p_{l^*-1}(t_0)}{s_{l^*-1} + s_{l^*}} > \\ \frac{p + \bar{p}_{l^*-1}(t_0) + \bar{p}_{l^*}(t_0)}{s_{l^*-1} + s_{l^*} + s_{l^*+1}} &= \max_{l^*-1 \leq l \leq l^*+1} \left\{ \frac{\max\{\bar{p}_l(t_0), p\} + \sum_{j=l^*-1}^{l-1} \bar{p}_j(t_0)}{\sum_{j=l^*-1}^l s_j} \right\}. \end{aligned}$$

Hence, the structure of the schedule created by algorithm $\mathcal{A}$ deviates from the structure of the optimal schedule, and according to Lemma 1, algorithm $\mathcal{A}$ cannot solve the problem $Q|pmt_n, r_j|C_{\max}$ to optimality. $\square$

Lemma 2 and Lemma 3 are combined in Theorem 4, which specifies the borderline between the instances of $Q|pmt_n, r_j|C_{\max}$ that are or are not solvable to optimality on-line without processor sharing.

Theorem 4. *There exists an on-line algorithm that solves $Q|pmtn, r_j|C_{\max}$ to optimality without processor sharing if and only if*

$$\frac{s_{l-1}}{s_l} \leq \frac{s_l}{s_{l+1}} \quad \text{for all } 1 < l < m.$$

□

In the next section, we present an on-line algorithm that solves $Q|pmtn, r_j|C_{\max}$ to optimality without processor sharing if Condition 2 holds. Furthermore, we answer the single remaining question of how many preemptions are needed to create an optimal schedule if Condition 2 is satisfied.

5 An algorithm without processor sharing

To guarantee optimality, the on-line algorithm must at any arrival determine the block structure as defined in Section 3. Then it schedules each of the blocks with the Gonzalez-Sahni algorithm for $Q|pmtn|C_{\max}$. Since the Gonzalez-Sahni algorithm produces no more than $2(m-1)$ preemptions, and since there are only n arrivals, we know that an upper bound on the total number of preemption is $O(mn)$.

An instance that requires $\Theta(mn)$ preemptions is given by m machines with speeds $s_1, \dots, s_m$ and n jobs with $r_j = j-1$, and $p_j = \sum_{i=1}^m s_i$ ($j = 1, \dots, n$). Figure 2 shows an example of an optimal schedule of such an instance consisting of four machines and seven jobs. It is not so difficult to see that, to maintain the optimal structure at the arrival of J_j , we have to assign J_j to M_1 and all other jobs to a machine with index one higher than the current machine they are assigned to. This implies that at each arrival every job that is being processed is

M_1	1	2	3	4	5	6	7
M_2		1	2	3	4	5	6
M_3			1	2	3	4	5
M_4				1	2	3	4

Figure 2: Example of an instance requiring $O(mn)$ preemptions.

preempted. Hence, the first $n-m$ jobs are preempted $m-1$ times, and the $(n-m+j)$ th job is preempted $m-j$ times. If we add these numbers, then we get a total of $(n-\frac{1}{2}m)(m-1)$ preemptions that are necessary for optimality.

References

- BARTEL, Y., A. FIAT, H. KARLOFF, AND R. VOHRA [1992], New algorithms for an ancient scheduling problem, *Proceedings of the 24th ACM Symposium on Theory of Computing*, 51–58.
- CHEN, B., A. VAN VLIET, AND G.J. WOEGINGER [1994], *An Optimal Algorithm for Preemptive On-Line Scheduling*, Technical Report 9404/A, Erasmus University Rotterdam.
- GONZALEZ, T., AND S. SAHNI [1978], Preemptive scheduling of uniform processor systems, *Journal of the ACM* **25**, 92–101.
- HONG, K.S., AND J.Y.-T. LEUNG [1992], On-line scheduling of real-time tasks, *IEEE Transactions on Computers* **41**, 1326–1331.
- HORVATH, E.C., S. LAM, AND R. SETHI [1977], A level algorithm for preemptive scheduling, *Journal of the ACM* **24**, 32–43.
- LABETOULLE, J., E.L. LAWLER, J.K. LENSTRA, AND A.H.G. RINNOOY KAN [1984], Preemptive scheduling of uniform machines subject to release dates, in: W.R. Pulleyblank (ed.), *Progress in combinatorial optimization*, Academic Press, Toronto, 245–261, ISBN 0-12-566780-9.

- LAWLER, E. L., J.K. LENSTRA, A. RINNOOY KAN, AND D.B. SHMOYS [1993], Sequencing and scheduling: Algorithms and complexity, in: S.C. Graves, A.H.G. Rinnooy Kan, and P.H. Zipkin (eds.), *Logistics of Production and Inventory*, Handbooks in OR & MS 4, Elsevier Science Publishers B.V., North-Holland, Amsterdam, Chapter 9, 445-522, ISBN 0-444-87472-0.
- SAHNI, S., AND Y. CHO [1979], Nearly on-line scheduling of a uniform processor system with release times, *SIAM Journal on Computing* 8, 275-285.

A Derivation of Condition 4

We have that p must satisfy the constraints

$$\frac{p + \sum_{j=b_{k-1}+1}^{l^*-1} p_j(t)}{\sum_{j=b_{k-1}+1}^{l^*} s_j} \geq \frac{p + \sum_{j=b_{k-1}+1}^{l_1-1} p_j(t)}{\sum_{j=b_{k-1}+1}^{l_1} s_j} \quad \text{for all } b_{k-1} + 1 \leq l_1 \leq l^*,$$

$$\frac{p + \sum_{j=b_{k-1}+1}^{l^*-1} p_j(t)}{\sum_{j=b_{k-1}+1}^{l^*} s_j} > \frac{p + \sum_{j=b_{k-1}+1}^{l_2-1} p_j(t)}{\sum_{j=b_{k-1}+1}^{l_2} s_j} \quad \text{for all } l^* < l_2 \leq n+1.$$

Rewriting these conditions, we get

$$p \sum_{j=l_1+1}^{l^*} s_j \leq \sum_{j=b_{k-1}+1}^{l_1} s_j \sum_{j=b_{k-1}+1}^{l^*-1} p_j(t) - \sum_{j=b_{k-1}+1}^{l^*} s_j \sum_{j=b_{k-1}+1}^{l_1-1} p_j(t) \quad \text{for all } b_{k-1} + 1 \leq l_1 \leq l^*,$$

$$p \sum_{j=l^*+1}^{l_2} s_j > \sum_{j=b_{k-1}+1}^{l^*} s_j \sum_{j=b_{k-1}+1}^{l_2-1} p_j(t) - \sum_{j=b_{k-1}+1}^{l_2} s_j \sum_{j=b_{k-1}+1}^{l^*-1} p_j(t) \quad \text{for all } l^* < l_2 \leq n+1.$$

There exists a job, the arrival of which will cause the structures to deviate if and only if the interval of possible values for p is not empty. Hence, we need for all l_1, l_2 with $b_{k-1} + 1 \leq l_1 \leq l^* < l_2 \leq n+1$ that

$$\begin{aligned} & \sum_{j=l_1+1}^{l^*} s_j \left(\sum_{j=b_{k-1}+1}^{l^*} s_j \sum_{j=b_{k-1}+1}^{l_2-1} p_j(t) - \sum_{j=b_{k-1}+1}^{l_2} s_j \sum_{j=b_{k-1}+1}^{l^*-1} p_j(t) \right) \\ & < \sum_{j=l^*+1}^{l_2} s_j \left(\sum_{j=b_{k-1}+1}^{l_1} s_j \sum_{j=b_{k-1}+1}^{l^*-1} p_j(t) - \sum_{j=b_{k-1}+1}^{l^*} s_j \sum_{j=b_{k-1}+1}^{l_1-1} p_j(t) \right) \\ \Leftrightarrow & \sum_{j=b_{k-1}+1}^{l^*} s_j \left(\sum_{j=l_1+1}^{l^*} s_j \sum_{j=b_{k-1}+1}^{l_2-1} p_j(t) - \sum_{j=b_{k-1}+1}^{l_2} s_j \sum_{j=b_{k-1}+1}^{l^*-1} p_j(t) \right) \\ & < - \sum_{j=b_{k-1}+1}^{l^*} s_j \left(\sum_{j=b_{k-1}+1}^{l_1} s_j \sum_{j=b_{k-1}+1}^{l^*-1} p_j(t) + \sum_{j=l^*+1}^{l_2} s_j \sum_{j=b_{k-1}+1}^{l_1-1} p_j(t) \right) \\ \Leftrightarrow & \sum_{j=l_1+1}^{l^*} s_j \sum_{j=b_{k-1}+1}^{l_2-1} p_j(t) + \sum_{j=l^*+1}^{l_2} s_j \sum_{j=b_{k-1}+1}^{l_1-1} p_j(t) < \sum_{j=l_1+1}^{l_2} s_j \sum_{j=b_{k-1}+1}^{l^*-1} p_j(t) \\ \Leftrightarrow & \sum_{j=l_1+1}^{l^*} s_j \sum_{j=l^*}^{l_2-1} p_j(t) < \sum_{j=l^*+1}^{l_2} s_j \sum_{j=l_1}^{l^*-1} p_j(t). \end{aligned}$$