Semi-Online Preemptive Scheduling: One Algorithm for All Variants

Tomáš Ebenlendr* Jiří Sgall*

Abstract: We present a unified optimal semi-online algorithm for preemptive scheduling on uniformly related machines with the objective to minimize the makespan. This algorithm works for all types of semi-online restrictions, including the ones studied before, like sorted (decreasing) jobs, known sum of processing times, known maximal processing time, their combinations, and so on. Based on the analysis of this algorithm, we derive some global relations between various semi-online restrictions and tight bounds on the approximation ratios for a small number of machines.

Keywords: Algorithms, scheduling.

1 Introduction

We study online scheduling on *uniformly related machines*, which means that the time needed to process a job with processing time p on a machine with speed s is p/s. *Preemption* is allowed, which means that each job may be divided into several pieces, which can be assigned to different machines in disjoint time slots. The objective is to minimize the *makespan*, i.e., the length of a schedule. In the *online* problem, jobs arrive one-by-one and we need to assign each incoming job without any knowledge of the jobs that arrive later. When a job arrives, its assignment at all times must be given and we are not allowed to change this assignment later. In other words, the online nature of the problem is given by the ordering of the input sequence and it is not related to possible preemptions and the time in the schedule.

*Institute of Mathematics, AS CR, Žitná 25, CZ-11567 Praha 1, Czech Republic. Partially supported by Institutional Research Plan No. AV0Z10190503, by Inst. for Theor. Comp. Sci., Prague (project 1M0545 of MŠMT ČR) and grant IAA1019401 of GA AV ČR. Email: {ebik,sgall}@math.cas.cz.

We focus on *semi-online* algorithms. This term encompasses algorithms that are essentially online, but some partial information about the input is given to the scheduler in advance. The main motivation behind this approach is the observation that the classical competitive analysis is too pessimistic compared to practical results, or, in other words, the adversary who may arbitrarily determine the input sequence is too powerful. In practice, the inputs are not completely arbitrary, and it may be reasonable to restrict the set of inputs. In scheduling, numerous semi-online models have been studied; typical examples include (sequences of) jobs with decreasing processing times, jobs with bounded processing times, sequences with known total processing times of jobs and so on. Most of these models can be viewed as online algorithms on a restricted set of input sequences. Restrictions of this type have been studied also for other online problems; the most prominent example is paging with locality of reference [1].

Our results

We give a semi-online algorithm for preemptive scheduling on uniformly related machines which is optimal for any chosen semi-online restriction, see Section 2. This means not only the cases listed above—the restriction can be given as an *arbitrary set* of sequences that are allowed as inputs. For any semi-online restriction, the algorithm achieves the best possible approximation ratio for any number of machines and any particular combination of machine speeds; it is deterministic, but its approximation ratio matches the best possible approximation ratio of any randomized algorithm. This generalizes and unifies previous results for various special cases of semi-online preemptive scheduling. We find such a general result providing a provably optimal algorithm for many problems quite exceptional not only in the area of scheduling but also in the whole area of online algorithms. Our result also provides a clear separation between the design of the algorithm and the analysis of the optimal approximation ratio. While the algorithm is always the same, analysis of the optimal ratio depends on the studied restrictions. Nevertheless, the general result also provides crucial new insights and methods and thus we can analyze the optimal ratio in cases that have been out of reach with previously known techniques.

For typical semi-online restrictions, we show that the optimal ratio can be computed by linear programs (with machine speeds as parameters). Studying these linear programs allows us to progress in two directions. First, we are able to completely analyze the optimal ratio for particular cases with a small number of machines. Second, we are able to study the

relations between the optimal approximation ratios for different semi-online restrictions and give some bounds for a large number of machines.

The exact analysis of special cases for a small number of machines was given in [6, 3, 11] for various restrictions, and in many more cases for non-preemptive scheduling. Typically, these results involve similar but ad hoc algorithms and an extensive case analysis which is tedious to verify, and can be done for two uniformly related machines or for more identical machines. Using our linear programs we can calculate the ratio as a formula in terms of speeds. This is a fairly routine task which can be simplified (but not completely automated) using standard mathematical software. Once the solution is known, verification amounts to checking the given primal and dual solutions for the linear program. We give a few examples of such results in Section 3; typically the verification is reasonably simple for $m = 3$ or $m = 4$.

Another research direction is to compute, for a given semi-online restriction, the optimal approximation ratio which works for any number of machines and combination of speeds. This task appears to be much harder, and even in the online case we only know that the ratio is between 2.054 and $e \approx 2.718$; the lower bound is shown by a computer-generated hard instance with no clear structure [4]. Only for identical machines, the exact ratio for any number of machines is known (i) for the online case, where it tends to $e/(e-1) \approx 1.58$ [2], and (ii) for non-increasing processing times, where it tends to $(1+\sqrt{3})/2 \approx 1.366$ [13].

We are able to prove certain relations between the approximation ratios for different restrictions. Some basic restrictions form an inclusion chain: The inputs where the first job has the maximal size (which is equivalent to known maximal size) include the inputs with non-increasing processing times, which in turn include the inputs with all jobs of equal size. Typically, the hard instances have non-decreasing processing times. Thus, one expected result is that the restriction to non-increasing processing times gives the same approximation ratio as when all jobs have equal size, even for any particular combination of speeds. The overall approximation ratio is at most 1.52, see Section 3.3. On the other hand, for known maximal size of a job we have a computer-generated hard instance with approximation ratio 1.88 with $m = 120$.[1] Thus restricting the jobs to be non-increasing helps the algorithm much more than just knowing the maximal size of a job. This is different from identical machines, where knowing the maximal

[1] See the Maple output at `http://www.math.cas.cz/~sgall/ps/semirel-pmax.mpl`

size is equally powerful as knowing that all the jobs are equal, see [13].

More interestingly, the overall approximation ratio with known sum of processing times is the same as in the online case—even though for a small fixed number of machines knowing the sum provides a significant advantage. This is shown by a padding argument, see Section 3.1.1. In fact this is true also in presence of any additional restriction that allows scaling input sequences, taking a prefix, and extending the input by small jobs at the end. Thus, for example, the overall approximation ratio with non-increasing jobs and known sum of processing times is at least 1.366, using the bound for identical machines from [13].

Preliminaries

Let M_i, $i = 1, 2, \ldots, m$ denote the m machines, and let s_i be the speed of M_i. W.l.o.g. we assume that the machines are sorted by decreasing speeds, i.e., $s_1 \geq s_2 \geq \ldots \geq s_m$. To avoid degenerate cases, we assume that $s_1 > 0$. The vector of speeds is denoted $\mathbf{s}$. The sum of speeds is denoted $S = \sum_{i=1}^{m} s_i$ and $S_k = \sum_{i=1}^{k} s_i$ is the sum of k largest speeds. To simplify the description of the algorithm, we assume that there are infinitely many machines of speed zero, i.e., $s_i = 0$ for any $i > m$. (Scheduling a job on one of these zero-speed machines means that we do not process the job at the given time at all.) Let $\mathcal{J} = (p_j)_{j=1}^{n}$ denote the input sequence of jobs, where n is the number of jobs and $p_j \geq 0$ is the size, or processing time, of jth job. The sum of processing times is denoted $P = P(\mathcal{J}) = \sum_{j=1}^{n} p_j$. Given $\mathcal{J}$ and $i \leq n$, let $\mathcal{J}_{[i]}$ be the prefix of $\mathcal{J}$ obtained by taking the first i jobs.

The time needed to process a job p_j on machine M_i is p_j/s_i; each machine can process at most one job at any time. Preemption is allowed, which means that each job may be divided into several pieces, which can be assigned to different machines, but any two time slots to which a single job is assigned must be disjoint (no parallel processing of a job); there is no additional cost for preemptions. Formally, if t_i denotes the total length of the time intervals when the job p_j is assigned to machine M_i, it is required that $t_1 s_1 + t_2 s_3 + \cdots + t_m s_m = p_j$. (A job may be scheduled in several time slots on the same machine, and there may be times when a partially processed job is not running at all.) In the (semi-)online version of this problem, jobs arrive one-by-one and at that time the algorithm has to give a complete assignment of this job at all times, without the knowledge of the jobs that arrive later. The objective is to find a schedule of all jobs in which the maximal completion time (the makespan) is minimized.

For an algorithm A, let $C^{\mathsf{A}}_{\max}[\mathcal{J}]$ be the makespan of the schedule of $\mathcal{J}$ produced by A. By $C^*_{\max}[\mathcal{J}]$ we denote the makespan of the optimal offline schedule of $\mathcal{J}$. An algorithm A is an R-approximation if for every input $\mathcal{J}$, the makespan is at most R times the optimal makespan, i.e., $C^{\mathsf{A}}_{\max}[\mathcal{J}] \leq R \cdot C^*_{\max}[\mathcal{J}]$. In case of a randomized algorithm, the same must hold for every input for the expected makespan of the online algorithm, $\mathbb{E}[C^{\mathsf{A}}_{\max}[\mathcal{J}]] \leq R \cdot C^*_{\max}[\mathcal{J}]$, where the expectation is taken over the random choices of the algorithm.

The optimal makespan can be computed as

$$C^*_{\max}[\mathcal{J}] \quad = \max\{P/S, \max_{k=1}^{m-1}\{P_k/S_k\}\}, \tag{1}$$

where P_k denotes the sum of k largest processing times in $\mathcal{J}$ and S_k is the sum of k largest speeds. It is easy to see that the right-hand side is a lower bound on the makespan, as the first term gives the minimal time when all the work can be completed using all machines fully, and similarly the term for k is the minimal time when the work of k largest jobs can be completed using k fastest machines fully. The tightness of this bound follows from [10, 8, 5].

Semi-online restrictions and previous work

We define a general semi-online input restriction to be simply a set Ψ of allowed inputs, also called *input sequences*. We call a sequence an *partial input* if it is a prefix of some input sequence; the set of all partial inputs is denoted $\mathrm{pref}(\Psi)$. Thus partial inputs are exactly the sequences that the algorithm can see at some point. A (randomized) semi-online algorithm A with restriction Ψ is an R-approximation algorithm if $\mathbb{E}[C^{\mathsf{A}}_{\max}[\mathcal{J}]] \leq R \cdot C^*_{\max}[\mathcal{J}]$ for any $\mathcal{J} \in \Psi$. Note that this implies the same condition even for any $\mathcal{J} \in \mathrm{pref}(\Psi)$.

Below we list some of the restrictions that are studied in literature, together with the notation that we are going to use, previous work, and our results.

Known sum of processing times, $\sum p_j = P$. For a given value $\bar{P}$, Ψ contains all sequences with $P = \bar{P}$. We prove that the overall ratio is surprisingly the same as in the general online case, on the other hand we note that for $m = 2$, 1-approximation is possible and we analyze the cases of $m = 3, 4$.

Non-increasing processing times, denoted *decr*. Ψ contains all sequences with $p_1 \geq p_2 \geq \cdots \geq p_n$. For $m = 2$, the optimal algorithm for all speeds was analyzed in [6] and for identical machines in [13]. We prove

that for any speeds this case is the same as the case with all jobs equal. We analyze the cases for $m = 2, 3$, and prove some bounds for larger m.

Known optimal makespan, $C^*_{\max} = T$. For a given value $\bar{T}$, Ψ contains all sequences with $C^*_{\max}[\mathcal{J}] = \bar{T}$. In this case 1-approximation semi-online algorithm is known for any speeds, see [5].

Known maximal job size, $p_{\max} = p$. For a given value $\bar{p}$, Ψ contains all sequences with $\max p_j = \bar{p}$. It is easy to see that this restriction is equivalent to the case when the first job is maximal, as any algorithm for that special case can be used also for the case when the maximal job arrives later. Thus this restriction also includes non-increasing jobs. This restriction was introduced in [9] for non-preemptive scheduling on 2 identical machines. In [13] it is shown that for identical machines, the approximation ratio is the same as when the jobs are non-increasing. We show that this is not the case for general speeds.

Tightly grouped processing times, $p_j \in [p, \alpha p]$. For given values $\bar{p}$ and α, Ψ contains all sequences with $p_j \in [p, \alpha p]$ for each j. This restriction was introduced in [9] for non-preemptive scheduling on 2 identical machines. Tight bounds for preemptive scheduling on 2 uniformly related machines were given in [3].

Inexact partial information. In this case, some of the previously considered values (optimal makespan, sum of job sizes, maximal job size) is not known exactly but only up to a certain factor. These variants were studied first in [15] without preemption and then in [11] for preemptive scheduling; both on identical machines.

Online scheduling. Here Ψ contains all sequences. In our (i.e., the authors and Wojtek Jawor) previous work [4], we have designed an optimal online algorithm for all speed vectors. The algorithm and the proof of the main result in this paper generalize that result, using the same techniques, however, some technical issues have to be handled carefully to achieve the full generality of our new result. Online preemptive scheduling was studied first in [2].

The paper [12] is probably the first paper which studied and compared several notions of semi-online algorithms, including known sum of processing times. Some combination of the previous restrictions were studied in [14] for non-preemptive scheduling on identical machines. We should note that there are also semi-online models that do not fit into our framework at all. For example, the algorithm may get a hint which job is the last one, or it is allowed to store some job(s) in a buffer.

2 The optimal algorithm

The new algorithm is based on the algorithm for online scheduling from [4]. Since the technical parts are similar, we have chosen to present the precise description and proofs only in Appendix A. In this section we present the key ideas with emphasis on the issues that need to be handled differently in the more general semi-online setting.

Suppose that we are given a parameter r and we try to develop an r-approximation algorithm. In the online case, we simply make sure that the current job completes by time r times the current optimal makespan. In the semi-online case, if the restriction is not closed under taking a prefix, this would be too pessimistic. It may happen that the current partial input is not in Ψ and we know that any extension in Ψ has much larger optimal makespan; then we can run the current job on a slow machine. For this purpose, we define the appropriate quantity to be used instead of the current optimal makespan.

Definition 2.1 *For an input restriction Ψ and a partial input $\mathcal{J} \in \mathrm{pref}(\Psi)$, we define the optimal makespan as the infimum over all possible end extensions of $\mathcal{J}$ that satisfy Ψ:*

$$C^{*,\Psi}_{\max}[\mathcal{J}] = \inf\{C^{*}_{\max}[\mathcal{J}'] \mid \mathcal{J}' \in \Psi \;\&\; \mathcal{J} \text{ is a prefix of } \mathcal{J}'\}$$

Note that for any input sequence $\mathcal{J} \in \Psi$ we have $C^{*}_{\max}[\mathcal{J}] = C^{*,\Psi}_{\max}[\mathcal{J}]$.

Algorithm RatioStretch

Our algorithm takes as a parameter a number r which is the desired approximation ratio. Later we show that for the right choice of this parameter, our algorithm is optimal. Given r, we want to schedule each incoming job so that it completes at time $r \cdot C^{*,\Psi}_{\max}[\mathcal{J}_{[j]}]$. If this is done for each job, the algorithm is obviously r-approximation.

Even when we decide the completion time of a job, there are many ways to schedule it given the flexibility of preemptions. We choose a particular one based on the notion of a *virtual machine* from [5, 4]. We define the ith *virtual machine*, denoted V_i, so that at each time τ it contains the ith fastest machine among those real machines M_1, M_2, ..., M_m that are idle at time τ. Due to preemptions, a virtual machine can be thought and used as a single machine with changing speed. When we schedule (a part of) a job on a virtual machine during some interval, we actually schedule it on the corresponding real machines that are uniquely defined at each time.

Upon arrival of a job j we compute a value T_j defined as $r \cdot C^{*,\Psi}_{\max}[\mathcal{J}_{[j]}]$. Then we find two adjacent virtual machines V_k and V_{k+1}, and time t_j, such that if we schedule j on V_{k+1} in the time interval $(0, t_j]$ and on V_k from t_j on, then j finishes exactly at time T_j.

We need to show that we can always find such machines V_k and V_{k+1}. Since we have added the machines of speed 0, it only remains to prove that each job can fit on V_1. This is true for the appropriate value of r.

Before we sketch the proof, we make a few remarks concerning efficiency and uniformity of the algorithm. The only parts of the algorithm that depend on the semi-online restriction are (i) the computation of the optimal approximation ratio and (ii) the computation of $C^{*,\Psi}_{\max}[\mathcal{J}]$. The rest of the algorithm is independent of the restriction and very efficient. Similarly to the online algorithms, for semi-online algorithms we generally do not require the computation to be polynomial time. For a general restriction the optimal algorithm cannot be efficient. (If the set of input sequences is, e.g., not recursive, then it may be algorithmically undecidable how much time we have even for scheduling the first job. Besides, there are more possible restrictions than algorithms.) Nevertheless, the algorithm is efficient for many natural restrictions. Computing $C^{*,\Psi}_{\max}[\mathcal{J}]$ is usually simple. If the restriction is closed under taking prefixes, then it is equal to $C^{*}_{\max}[\mathcal{J}]$. In other cases it is easy to see which extension has the smallest makespan. Computing the optimal approximation ratio is more difficult, but in Section 3 it is shown that in many natural cases it reduces to linear programming. Alternatively, we can use any upper bound on the approximation ratio and give to the algorithm as a parameter.

Optimality of Algorithm RatioStretch

Our goal is to show that Algorithm RatioStretch works whenever the parameter r is at least the optimal approximation ratio for the given Ψ and $\mathbf{s}$. We actually prove the converse: Whenever for some instance $\mathcal{J}$ Algorithm RatioStretch with the parameter r fails, we prove that there is no r-approximation algorithm.

This is based on a generalization of a lemma from [7] which provides the optimal lower bounds for online algorithms, as shown in [4]. The key observation in its proof is this: On an input $\mathcal{J}$, if the adversary stops the input sequence at the ith job from the end, any r-competitive online algorithm must complete by time r times the current optimal makespan, and after this time, in the schedule of $\mathcal{J}$, only $i-1$ machines can be used. This bounds the total work of all the jobs in terms of r and optimal makespans

of the prefixes, and thus gives a lower bound on r. To generalize to an arbitrary restriction Ψ, we need to deal with two issues.

First, the adversary cannot stop the input if the current partial input is not in Ψ. Instead, the sequence then must continue so that its optimal makespan is the current $C^{*,\Psi}_{\max}$ (or its approximation). Consequently, the bound obtained uses $C^{*,\Psi}_{\max}$ in place of previous $C^{*}_{\max}$, which possibly decreases the obtained bound.

Second, for a general semi-online restriction, using the last m prefixes of $\mathcal{J}$ may not give the best possible lower bound. E.g., the restriction may force that some job is tiny, and thus using the prefix ending at this job is useless; in general, we also cannot remove such a job from the input sequence. To get a stronger lower bound, we choose a subsequence of important jobs from $\mathcal{J}$ and bound their total work in terms of values $C^{*,\Psi}_{\max}$ of the prefixes of the original sequence $\mathcal{J}$.

Lemma 2.2 *Let* A *be any randomized R-approximation semi-online algorithm for preemptive scheduling on m machines with an input restriction Ψ. Then for any partial input $\mathcal{J} \in \mathrm{pref}(\Psi)$, for any k, and for any subsequence of jobs $1 \le j_1 < j_2 < \cdots < j_k \le n$ we have*

$$\sum_{i=1}^{k} p_{j_i} \quad \le \quad R \cdot \sum_{i=1}^{k} s_{k+1-i} C^{*,\Psi}_{\max}[\mathcal{J}_{[j_i]}].$$

We define r^{Ψ} to be the largest lower bound on the approximation ratio obtained by Lemma 2.2.

Definition 2.3 *For any vector of speeds* $\mathbf{s}$ *and any partial input* $\mathcal{J} \in \mathrm{pref}(\Psi)$,

$$r^{\Psi}(\mathbf{s}, \mathcal{J}) = \sup_{1 \le j_1 < j_2 < \cdots < j_k \le n} \frac{\sum_{i=1}^{k} p_{j_i}}{\sum_{i=1}^{k} s_{k+1-i} \cdot C^{*,\Psi}_{\max}[\mathcal{J}_{[j_i]}]}.$$

For any vector of speeds $\mathbf{s}$, *let* $r^{\Psi}(\mathbf{s}) = \sup_{\mathcal{J} \in \mathrm{pref}(\Psi)} r^{\Psi}(\mathbf{s}, \mathcal{J})$. *Finally, let* $r^{\Psi} = \sup_{\mathbf{s}} r^{\Psi}(\mathbf{s})$.

With these definitions and Lemma 2.2, we can prove the following main theorem. If Algorithm RatioStretch cannot schedule the incoming job, we choose a subsequence including the jobs scheduled so far on the first virtual machine and the incoming job. We use Lemma 2.2 with this subsequence to argue that that no (randomized) algorithm can have the same approximation ratio.

Theorem 2.4 *For any restriction Ψ and vector of speeds $\mathbf{s}$, Algorithm* RatioStretch *with a parameter $r \geq r^{\Psi}(\mathbf{s})$ is an r-approximation algorithm for semi-online preemptive scheduling on m uniformly related machines. In particular, $r^{\Psi}(\mathbf{s})$ (resp. r^{Ψ}) is the optimal approximation ratio for semi-online algorithms for Ψ with speeds $\mathbf{s}$ (resp. with arbitrary speeds).*

3 Reductions and linear programs

We have an abstract formula for $r^{\Psi}(\mathbf{s})$ which gives the desired approximation ratio for any speeds and Ψ as a supremum over a bound for all partial inputs and all their subsequences. It is not obvious how to turn this into an efficient algorithm. Now we develop a general methodology how to compute the ratio using linear programs and apply it to a few cases.

We observed that for a general restriction it may be necessary to use an arbitrary subsequence in Definition 2.3. However, for many restrictions it is sufficient to use the whole sequence, similarly as for online scheduling. Usual restrictions are essentially of two kinds. The first type are those restriction that put conditions on individual jobs or their order. These restrictions are closed under taking subsequences (not only prefixes), i.e., any subsequence of an input sequence is also in Ψ. The second type are those restriction where some global information is given in advance, like $\sum p_j = P$ or $C^*_{\max} = T$. These are not closed under taking subsequences, but are closed under permuting the input sequence. We define a large class of restrictions that includes both types of restrictions discussed above as well as their combinations; in particular it includes all the restrictions listed and studied here.

Definition 3.1 *An input restriction Ψ is* proper *if for any $\mathcal{J} \in \Psi$ and any subsequence $\mathcal{I}$ of $\mathcal{J}$, we have $\mathcal{I} \in \mathrm{pref}(\Psi)$ and furthermore $C^{*,\Psi}_{\max}[\mathcal{I}] \leq C^{*,\Psi}_{\max}[\mathcal{J}]$.*

Definition 3.2 *Let Ψ be a proper semi-online restriction and $\mathcal{J} \in \mathrm{pref}(\Psi)$ a partial input. We define*

$$\bar{r}^{\Psi}(\mathbf{s}, \mathcal{J}) = \frac{\sum_{j=1}^{n} p_j}{\sum_{j=1}^{n} s_{n+1-j} \cdot C^{*,\Psi}_{\max}[\mathcal{J}_{[j]}]}.$$

From now on, we focus on proper restrictions. It may happen that $r^{\Psi}(\mathbf{s}, \mathcal{J}) > \bar{r}^{\Psi}(\mathbf{s}, \mathcal{J})$. By Definitions 2.3 and 2.3 we may take a subsequence

of jobs $\mathcal{I} = (p_{j_i})_{i=1}^{k}$ that achieves the value of $\bar{r}^{\Psi}(\mathbf{s}, \mathcal{I}) \geq r^{\Psi}(\mathbf{s}, \mathcal{J}) - \varepsilon$ for any $\varepsilon > 0$. By the definition of a proper restriction, $\mathcal{I} \in \mathrm{pref}(\Psi)$. Taking the supremum over all partial inputs, we obtain the following simpler formula for the optimal approximation ratio.

Observation 3.3 *For any proper restriction Ψ,*

$$r^{\Psi}(\mathbf{s}) = \sup_{\mathcal{J} \in \mathrm{pref}(\Psi)} \bar{r}^{\Psi}(\mathbf{s}, \mathcal{J})$$

Our strategy is to reduce the number of sequences $\mathcal{J}$ that need to be taken into account. Typically, we show that the sequences must be sorted. Then we know which jobs are the biggest ones and we can express the optimal makespans for prefixes by linear constraints in job sizes. Maximizing the expression for $\bar{r}^{\Psi}(\mathbf{s})$, which gives the desired bound, is then reduced to solving one or several linear programs. The following observation helps us to limit the set of relevant sequences.

Observation 3.4 *Let Ψ be arbitrary proper restriction, let $\mathbf{s}$ be arbitrary speed vector, and let $\mathcal{J}, \mathcal{J}' \in \mathrm{pref}(\Psi)$,. be two partial inputs with n jobs. Suppose that for some $b > 0$:*

$$\begin{aligned} \sum_{j=1}^{n} p'_j &= b \cdot \sum_{j=1}^{n} p_j, \quad \textit{and} \\ (\forall i = 1, \ldots, n) \quad C^{*,\Psi}_{\max}[\mathcal{J}'_{[i]}] &\leq b \cdot C^{*,\Psi}_{\max}[\mathcal{J}_{[i]}]. \end{aligned}$$

Then $\bar{r}(\mathbf{s}, \mathcal{J}') \geq \bar{r}(\mathbf{s}, \mathcal{J})$.

The observation follows immediately from the definition of $\bar{r}^{\Psi}(\mathbf{s}, \mathcal{J})$.

Whenever (i) Ψ is closed under permutations of the sequence and (ii) increasing the size of the last job of a partial input cannot decrease $C^{*,\Psi}_{\max}$, the observation implies that it is sufficient to consider sequences of non-decreasing jobs: If $\mathcal{J}$ contains two jobs with $p_k < p_{k+1}$, swapping them can only increase $C^{*,\Psi}_{\max}[\mathcal{J}_{[k]}]$ and any other $C^{*,\Psi}_{\max}[\mathcal{J}_{[i]}]$ remains unchanged; thus the observation applies with $b = 1$.

3.1 Known sum of processing times, $\sum p_j = P$

Here we are given a value $\bar{P}$ and Ψ contains all $\mathcal{J}$ with $P = \bar{P}$. It can be easily verified that $C^{*,\Psi}_{\max}[\mathcal{J}] = \max\{C^{*}_{\max}[\mathcal{J}], \bar{P}/S\}$ for any $\mathcal{J}$ with $P \leq \bar{P}$.

Since we can permute the jobs and increasing the size of the last job does not decrease $C^{*,\Psi}_{\max}$, Observation 3.4 implies that we can restrict ourselves to non-decreasing sequences $\mathcal{J}$. Furthermore, we may assume that $P = \bar{P}$: We know that $P \leq \bar{P}$, as otherwise $\mathcal{J}$ is not a partial input. If $P < \bar{P}$, we scale up $\mathcal{J}$ to $\mathcal{J}'$ by multiplying all the sizes by $b = P'/P$. Observation 3.4 then applies, as each $C^{*,\Psi}_{\max}[\mathcal{J}'_{[i]}] = \max\{C^{*}_{\max}[\mathcal{J}'_{[i]}], \bar{P}/S\}$ increases by at most the scaling factor b. Finally, we observe that we can restrict ourselves to sequences $\mathcal{J}$ with less than m jobs. If $n \geq m$, we use the fact that $C^{*,\Psi}_{\max}[\mathcal{J}_{[i]}] \geq \bar{P}/S$ for any i and obtain $\bar{r}^{\Psi}(\mathbf{s}, \mathcal{J}) = P/(\sum_{i=1}^{n} s_{n+1-i} \cdot C^{*,\Psi}_{\max}[\mathcal{J}_{[i]}]) \leq P/(\sum_{i=1}^{n} s_{n+1-i} \cdot \bar{P}/S) = 1$, using $n \geq m$ in the last step.

Summarizing, we can assume that $\mathcal{J}$ is a non-decreasing sequence of $n < m$ jobs with $P = \bar{P}$. (Note that this does not mean that the adversary uses fewer jobs than machines, as he may need to release some small jobs at the end of the prefix sequence, to extend it to a sequence in Ψ.) To obtain the worst case bound, we compute $m-1$ linear programs, one for each value of n, and take the maximum of their solutions. The linear program for a given P, $\mathbf{s}$, and n has variables q_i for job sizes and O_i for optimal makespans of the prefixes:

$$
\begin{array}{rcll}
\textbf{minimize} & r^{-1} = \dfrac{s_1 O_n + s_2 O_{n-1} + \cdots + s_n O_1}{\bar{P}} & & \\
\textbf{subject to} & & & \\
q_1 + \cdots + q_n & = & \bar{P} & \\
\bar{P} & \leq & (s_1 + \cdots + s_m) O_k & \text{for } k = 1, \ldots, n \\
q_j + q_{j+1} + \cdots + q_k & \leq & (s_1 + \cdots + s_{k-j+1}) O_k & \text{for } 1 \leq j \leq k \leq n \\
q_j & \leq & q_{j+1} & \text{for } j = 1, \cdots, n-1 \\
0 & \leq & q_1 &
\end{array}
$$

If we fix the input sequence, i.e., the values of q_i, then the smallest objective is achieved for O_k as small as possible which is exactly the value of the optimal makespan, by the constraints involving O_k. Thus the linear program computes correctly the (inverse of the) value $r^{\sum p_j = P}(\mathbf{s})$. We can also see that the linear program scales and the optimum does not depend on the value $\bar{P}$.

We now examine the special cases of $m = 2, 3$. The linear program is trivial for $n = 1$, and we conclude that for $m = 2$ the approximation ratio is equal to 1, i.e., RatioStretch always produces an optimal schedule. We can see this also intuitively: The algorithm starts scheduling the incoming jobs in the interval $[0, T_1)$ where $T_1 \geq \bar{P}/S$. Consider the first time when

a job is scheduled at the first real machine M_1. It is always possible to schedule this job at the empty machine M_1 so that it completes before the current optimal makespan. Furthermore, after M_1 is used the first time, the algorithm guarantees that in the interval $[0, T_1)$ there is only one real machine idle at any time. This in turn implies that the remaining jobs can be completed by time T_1, as the total size of all jobs is $\bar{P} \leq S \cdot T_1$.

For $m = 3$, it remains to solve the linear program for $n = 2$, which we do in Appendix B. The resulting ratio is:

$$r^{\sum p_j = P}(s_1, s_2, s_3) = \begin{cases} \dfrac{s_1(s_1+s_2)}{s_1^2+s_2^2} & \text{for } s_1^2 \leq s_2(s_2+s_3) \\ 1 + \dfrac{s_2 s_3}{s_1(s_1+s_2+s_3)+s_2(s_1+s_2)} & \text{for } s_1^2 \geq s_2(s_2+s_3) \end{cases}$$

The overall worst case ratio for three machines is $\frac{2+\sqrt{2}}{3} \approx 1.138$ for $s_1 = \sqrt{2}, s_2 = s_3 = 1$. This should be compared with the unrestricted online case where the optimal ratio for two machines is $4/3$ and for three machines 1.461.

3.1.1 Padding

We prove a theorem that shows that knowing the total size of jobs does not improve the overall approximation ratio. This may sound surprising, as for two machines, knowing the sum allows to generate an optimal schedule, and also for three machines the improvement is significant. The same result holds also in presence of an additional restriction with suitable properties. Among the restrictions that we consider, the requirements are satisfied for non-increasing jobs, known maximal job size, or the online case. By "$\Psi, \sum p_j = P$" we denote the intersection of the two restrictions, i.e., the set of all sequences $(p_j)_{j=1}^n \in \Psi$ such that $\sum_{i=1}^n p_j = \bar{P}$ for a given value of $\bar{P}$

We say that Ψ allows scaling if for any $\mathcal{J} \in \Psi$ and $b > 0$, the modified sequence $\mathcal{J}' = (bp_j)_{j=1}^n$ satisfies $\mathcal{J}' \in \Psi$. We say that Ψ allows padding if for any $\mathcal{J} \in \Psi$, there exists $\varepsilon_0 > 0$ such that any sequence $\mathcal{J}'$ created by extending $\mathcal{J}$ by an arbitrary number of equal jobs of size $\varepsilon < \varepsilon_0$ at the end satisfies $\mathcal{J}' \in \Psi$.

Theorem 3.5 *Suppose that Ψ is proper, allows scaling, padding, and is closed under taking prefixes. Let $\mathcal{J} \in \Psi$ and let $\mathbf{s}$ be arbitrary. Then for any $\delta > 0$ there exists $\mathcal{J}'$ and $\mathbf{s}'$ such that*

$$\bar{r}^{\Psi, \sum p_j = P}(\mathbf{s}', \mathcal{J}') \geq \bar{r}^{\Psi}(\mathbf{s}, \mathcal{J})/(1+\delta).$$

Consequently, $r^{\Psi,\sum p_j=P} = r^{\Psi}$.

Proof: We fix $\mathbf{s}$, $\mathcal{J}$, and $\bar{P}$ given to the algorithm with the restriction $\sum p_j = P$. We proceed towards constructing the appropriate $\mathbf{s}'$ and $\mathcal{J}'$.

Since Ψ allows scaling, the value $C^{*,\Psi}_{\max}[\mathcal{J}]$ is multiplied by b when $\mathcal{J}$ is scaled by b. Consequently, the value of $\bar{r}^{\Psi}(\mathbf{s}, \mathcal{J})$ does not change when $\mathcal{J}$ is scaled. Let $\mathcal{J}' = (p'_j)_{j=1}^{n}$ be the sequence $\mathcal{J}$ scaled so that $\sum_{j=1}^{n} p'_j = \bar{P}$. Then $\bar{r}^{\Psi}(\mathbf{s}, \mathcal{J}') = \bar{r}^{\Psi}(\mathbf{s}, \mathcal{J})$.

Choose a small $\sigma > 0$ so that $\sigma < s_m$ and $\sigma < \delta S/n$. Let $O_1 = p'_1/s_1$, i.e., the optimal makespan after the first job. Let $\mathbf{s}'$ be the sequence of speeds starting with $\mathbf{s}$ and continuing with $n + \bar{P}/(O_1\sigma)$ of values σ. The first condition on σ guarantees that $\mathbf{s}'$ is monotone and thus a valid sequence of speeds. The second condition guarantees that the added machines are sufficiently slow, so that for any sequence of at most n jobs, in particular for the prefixes of $\mathcal{J}'$, the makespan decreases by at most the factor of $(1+\delta)$. Since Ψ is closed under taking prefixes, $C^{*,\Psi}_{\max}$ equals $C^{*}_{\max}$ for any sequence. Thus we conclude that $\bar{r}^{\Psi}(\mathbf{s}', \mathcal{J}') \geq \bar{r}^{\Psi}(\mathbf{s}, \mathcal{J}')/(1+\delta)$.

Finally, we have added sufficiently many new machines so that for any sequence of at most n jobs, the empty new machines can accommodate total work of $\bar{P}$ without exceeding makespan O_1. This implies that for all prefixes of $\mathcal{J}'$, $C^{*,\Psi,\sum p_j=P}_{\max}[\mathcal{J}'_{[i]}] = C^{*,\Psi}_{\max}[\mathcal{J}'_{[i]}]$; thus $\bar{r}^{\Psi,\sum p_j=P}(\mathbf{s}', \mathcal{J}') = \bar{r}^{\Psi}(\mathbf{s}', \mathcal{J}')$.

Chaining the (in)equalities at the end of the last three paragraphs yields the theorem. □

3.2 Known maximal processing time, $p_{\max} = p$

Here we are given $\bar{p}$, the maximal size of a job. As noted before, any algorithm that works with the first job being the maximal one can be easily changed to a general algorithm for this restriction. First it virtually schedules the maximal job and then it compares the size of each job to $\bar{p}$. If it is equal for the first time, it schedules the job to the time slot(s) it reserved by virtual scheduling at the beginning. Other jobs are scheduled in the same way in both algorithms. Thus we can work with the equivalent restriction containing all the sequences where the first job is maximal. Then $C^{*,\Psi}_{\max}[\mathcal{J}] = C^{*}_{\max}[\mathcal{J}]$ for any partial input. Furthermore, by Observation 3.4, the other jobs can be swapped as in the previous case, and we can maximize only over sequences with non-decreasing job sizes from the second job on.

In this case we are able to use a single linear program to cover input sequences of an arbitrary length. The variables are: p for the length of the

first job, q_1 for the total length of jobs $p_2, \ldots, p_{n-m+1}$, and $q_2, \ldots, q_m$ for the jobs $p_{n-m+2}, \ldots, p_n$. For sequences with $n < m$, we set $q_1 = q_2 = \cdots q_{n-m} = 0$. The optimal approximation ratio is given by the following non-linear program:

$$\begin{array}{rcll} \textbf{maximize} & r = \dfrac{P}{s_1 O_m + s_2 O_{m-1} + \cdots + s_m O_1} & & \\ \textbf{subject to} & & & \\ p + q_1 + \cdots + q_m & = & P & \\ p + q_1 + \cdots + q_k & \leq & (s_1 + \cdots + s_m) O_k & \text{for } k = 1, \ldots, m \\ p + q_{j+1} + \cdots + q_k & \leq & (s_1 + \cdots + s_{k-j+1}) O_k & \text{for } 1 \leq j \leq k \leq m \\ q_j & \leq & q_{j+1} & \text{for } j = 2, \cdots, m-1 \\ 0 & \leq & q_1 & \\ q_m & \leq & p & \\ 0 & \leq & q_2 & \end{array} \tag{2}$$

If we fix the values of q_i, then the largest objective is achieved for O_k as small as possible. By the constraints involving O_k, this is exactly the value of the optimal makespan for a sequence where q_1 represents a prefix of a sequence of jobs smaller than q_2. (As a technicality, if $q_2 = 0$, O_k may only be an infimum of the optimal makespans of the corresponding sequences.) Thus the program computes correctly the value $r^{\sum p_j = P}(\mathbf{s})$.

The program scales, thus we can normalize any feasible solution so that the denominator of the objective function is a constant. More precisely, we get an equivalent linear program after adding the constraint

$$1 \quad = \quad s_1 O_m + s_2 O_{m-1} + \cdots + s_m O_1.$$

Now, after examining the linear program, we can further restrict its feasible domain. For any feasible solution, we can set both variables p and q_m to $(p+q_m)/2$. The solution remains feasible and the objective does not change. Thus the linear program with an added constraint $p = q_m$ is also equivalent. In other words, we can assume that the last job is equal to first (maximal) job.

Small number of machines. For two machines we get the approximation ratio

$$r^{p_{\max} = p}(s_1, s_2) = 1 + \frac{s_1 s_2}{(s_1 + s_2)^2 + s_1^2}$$

The maximum is 1.2 for $s_1 = s_2$. For three machines we get

$$r^{p_{\max}=p}(s_1, s_2, s_3) = \begin{cases} 1 + \dfrac{s_1(s_2+s_3)}{S^2+s_1^2} & \text{for } s_1 s_2 \geq s_3 S \\ 1 + \dfrac{s_1 s_2 + 2 s_1 s_3}{S^2 + 2s_1^2 + s_1 s_2} & \text{for } s_1 s_2 \leq s_3 S \end{cases}$$

This is maximized at $s_1 = 2, s_2 = s_3 = \sqrt{3}$, which falls into the second case and gives the ratio $(8 + 12\sqrt{3})/23 \approx 1.252$.

3.3 Non-increasing processing times, *decr*

We are also interested in sequences of non-increasing jobs, as this is one of the most studied restrictions. Now Ψ contains sequences which have $p_j \geq p_{j+1}$ for all j. We cannot swap jobs, however, we can take two adjacent jobs j and $j+1$ and replace both of them by jobs of the average size $(p_j + p_{j+1})/2$. By Observation 3.4, the approximation ratio does not decrease. Similarly, we can replace longer segment of jobs with only two distinct sizes by the same number of jobs of the average size. Repeating this process, we can conclude that for the worst case for a given set of speeds it is sufficient to consider sequences where all jobs have equal size. By scaling, the actual size of jobs does not matter, we only need to determine the length of the sequence which gives the highest ratio.

Let us denote $\hat{r}_n(\mathbf{s}) = \bar{r}^{decr}(\mathbf{s}, \mathcal{J})$ for a sequence $\mathcal{J}$ with n jobs with $p_j = 1$. For this sequence, $C^{*,\Psi}_{\max}[\mathcal{J}] = C^{*}_{\max}[\mathcal{J}] = n/S_n$. (Recall that $s_i = 0$ for $i > m$ and $S_k = \sum_{i=1}^{k} s_i$.) Using this for the prefixes, we obtain from Observation 3.3 that

$$\hat{r}_n(\mathbf{s}) = n \cdot \left(\sum_{k=1}^{n} \frac{k s_{n-k+1}}{S_k} \right)^{-1} . \tag{3}$$

It can be seen that for any speed vector, the sequence $\hat{r}_n(\mathbf{s})$ decreases with n for $n \geq 2m$. Thus computing the approximation ratio for any given speeds is efficient.

A natural approach to estimate the overall ratio is to find for each n the worst speed vector and the corresponding ratio $\hat{r}_n = \sup_{\mathbf{s}} \hat{r}_n(\mathbf{s})$. Based on numerical experiments, we conjecture that for each n, $\hat{r}_n$ is attained for some $\mathbf{s}$ with $s_1 = s_2 = \cdots = s_{m-1}$. I.e., almost all the speeds are equal. This conjecture would imply that with non-increasing jobs, the optimal overall

approximation ratio is the same for the uniformly related machines and for the identical machines, and this is equal to $(1+\sqrt{3})/2 \approx 1.366$ by [13].

This is almost equivalent to an intriguing geometric question. Suppose we have numbers x_i, y_i, $i = 1, \ldots, n$ such that $x_i y_i = i$ for all i and both sequences $(x_i)_{i=1}^n$ and $(y_i)_{i=1}^n$ are non-decreasing. Consider the union of rectangles $[0, x_i] \times [0, y_{n+1-i}]$ over all i; this is a staircase-like part of the positive quadrant of the plane. What is the smallest possible area of this union of rectangles? We conjecture that the minimum is attained for an instance with $y_1 = y_2 = \ldots = y_k$ and $x_{k+1} = x_{k+2} = \ldots = x_n$ for some k. This would imply the previous conjecture.

We are not able to determine exactly the values of $\hat{r}_n$, but we can prove certain relations between these values. In particular, for any integers a, n, and n', $r_{an} \geq r_n$ and $r_{n'} \leq \frac{n+1}{n} r_n$. For the first proof, we replace a sequence of speeds from the bound for r_n by a sequence where each speed is repeated a times, and the bound follows by manipulating the formula for r_n. The second inequality is shown by replacing the speeds for $r_{n'}$ by a shorter sequence where each new speed is a sum of a segment of a speeds in the original sequence, for a suitable a. Details are postponed to Appendix C. These relations show that whenever we are able to evaluate some r_n for a fixed n, the optimal overall ratio is at most $\frac{n+1}{n} r_n$.

For $n = 3$, maximizing the function $\hat{r}_n(\mathbf{s})$ can be done by hand and the maximum is $r_3 = 1.2$ for $s_1 = s_2 = 1$, $s_3 = 0$. This yields an overall upper bound of $\hat{r}_n \leq \frac{4}{3} \cdot \frac{6}{5} = 1.6$. By a computer-assisted proof we have shown that $\hat{r}_4 = (\sqrt{7}+1)/3 \approx 1.215$, yielding an overall upper bound of $\hat{r}_n \leq \frac{5}{4}\hat{r}_4 = \frac{5}{12}(\sqrt{7}+1) \approx 1.52$.

3.4 Other variants

If Ψ is given as an intersection of two standard restrictions, the same methods for reducing the number of candidates for the worst case instances apply. Thus typically we get again a linear program or an expression as for the individual restrictions, with additional constraints.

If some value from previous restrictions is given not exactly but it is only known to belong to some interval, typically it means that the linear program is weakened by relaxing some equation to a pair of inequalities, or by relaxing some inequality. Then the optimal ratio is again computed using a linear program.

Conclusions.

Similar methods can be used to analyze other semi-online restrictions, their combinations and inexact versions, or give formulas for the approximation ratios for more machines. This becomes a somewhat mechanical exercise; we have not found any surprising phenomenon in the cases we have examined so far.

It would be interesting, and it seems hard to us but not impossible, to determine the exact overall approximation ratios for the basic restrictions.

References

[1] A. Borodin, S. Irani, P. Raghavan, and B. Schieber. Competitive paging with locality of reference. *J. Comput. Systems Sci.*, 50:244–258, 1995.

[2] B. Chen, A. van Vliet, and G. J. Woeginger. An optimal algorithm for preemptive on-line scheduling. *Oper. Res. Lett.*, 18:127–131, 1995.

[3] D. Du. Optimal preemptive semi-online scheduling on two uniform processors. *Inform. Process. Lett.*, 92(5):219–223, 2004.

[4] T. Ebenlendr, W. Jawor, and J. Sgall. Preemptive online scheduling: Optimal algorithms for all speeds. In *Proc. 13th European Symp. on Algorithms (ESA)*, volume 4168 of *Lecture Notes in Comput. Sci.*, pages 327–339. Springer, 2006.

[5] T. Ebenlendr and J. Sgall. Optimal and online preemptive scheduling on uniformly related machines. In *Proc. 21st Symp. on Theoretical Aspects of Computer Science (STACS)*, volume 2996 of *Lecture Notes in Comput. Sci.*, pages 199–210. Springer, 2004.

[6] L. Epstein and L. M. Favrholdt. Optimal preemptive semi-online scheduling to minimize makespan on two related machines. *Oper. Res. Lett.*, 30:269–275, 2002.

[7] L. Epstein and J. Sgall. A lower bound for on-line scheduling on uniformly related machines. *Oper. Res. Lett.*, 26(1):17–22, 2000.

[8] T. F. Gonzales and S. Sahni. Preemptive scheduling of uniform processor systems. *J. ACM*, 25:92–101, 1978.

[9] Y. He and G. Zhang. Semi on-line scheduling on two identical machines. *Computing*, 62(3):179–187, 1999.

[10] E. Horwath, E. C. Lam, and R. Sethi. A level algorithm for preemptive scheduling. *J. ACM*, 24:32–43, 1977.

[11] Y. Jiang and Y. He. Optimal semi-online algorithms for preemptive scheduling problems with inexact partial information. *Theoret. Comput. Sci.*, 44(7-8):571–590, 2007.

[12] H. Kellerer, V. Kotov, M. G. Speranza, and Z. Tuza. Semi on-line algorithms for the partition problem. *Oper. Res. Lett.*, 21:235–242, 1997.

[13] S. Seiden, J. Sgall, and G. J. Woeginger. Semi-online scheduling with decreasing job sizes. *Oper. Res. Lett.*, 27:215–221, 2000.

[14] Z. Tan and Y. He. Semi-on-line problems on two identical machines with combined partial information. *Oper. Res. Lett.*, 30:408–414, 2002.

[15] Z. Tan and Y. He. Semi-online scheduling problems on two identical machines with inexact partial information. *Theoret. Comput. Sci.*, 377(1-3):110–125, 2007.

Appendix

A The optimal algorithm: Description and proofs

Description of the algorithm

To facilitate the proof, we maintain an assignment of scheduled jobs (and consequently busy machines at each time) to the set of virtual machines, i.e., for each virtual machine V_i we compute a set $\mathcal{S}_i$ of jobs assigned to V_i. Although the incoming job j is split between two different virtual machines, at the end of each iteration each scheduled job belongs to exactly one set $\mathcal{S}_i$, since right after j is scheduled the virtual machines executing this job are merged (during the execution of j). The sets $\mathcal{S}_i$ serve only as means of bookkeeping for the purpose of the proof, and their computation is not an integral part of the algorithm.

At each time τ, machine $M_{i'}$ belongs to V_i if it is the ith fastest idle machine at time τ, or if it is running a job $j \in \mathcal{S}_i$ at time τ. At each time τ the real machines belonging to V_i form a set of adjacent real machines, i.e.,

all machines $M_{i'}, M_{i'+1}, \ldots, M_{i''}$ for some $i' \leq i''$. This relies on the fact that we always schedule a job on two adjacent virtual machines which are then merged into a single virtual machine during the times when the job is running, and on the fact that these time intervals $(0, T_j]$ increase with j, as adding new jobs cannot decrease $C^{*,\Psi}_{\max}[(p_i)_{i=1}^{j}]$. See Figure 1 for an example of a step of the algorithm.

Let $v_i(t)$ denote the speed of the virtual machine V_i at time t, which is the speed of the unique idle real machine that belongs to V_i. Let $W_i(t) = \int_0^t v_i(\tau)d\tau$ be the total work which can be done on machine V_i in the time interval $(0, t]$. By definition we have $v_i(t) \geq v_{i+1}(t)$ and thus also $W_i(t) \geq W_{i+1}(t)$ for all i and t. Also $W_{m+1}(t) = v_{m+1}(t) = 0$ for all t.

Algorithm RatioStretch**.** Let r be a parameter such that $r \geq r^{\Psi}(s_1, \ldots, s_m)$. Initialize $T_0 := 0$, $\mathcal{S}_i := \emptyset$, $v_i(\tau) := s_i$, for all $i = 1, 2, \ldots, m+1$ and $\tau \geq 0$. This also sets $v_{m+1}(\tau) \equiv 0$.
For each arriving job j, compute the output schedule as follows:

(1) Let $T_j := r \cdot C^{*,\Psi}_{\max}[(p_i)_{i=1}^{j}]$.

(2) Find the smallest k such that $W_k(T_j) \geq p_j \geq W_{k+1}(T_j)$. If such k does not exist, then output "failed" and stop. Otherwise find time $t_j \in [0, T_j]$ such that $W_{k+1}(t_j) + W_k(T_j) - W_k(t_j) = p_j$.

(3) Schedule job j on V_{k+1} in time interval $(0, t_j]$ and on V_k in time interval $(t_j, T_j]$.

(4) Set $v_k(\tau) := v_{k+1}(\tau)$ for $\tau \in (t_j, T_j]$, and $v_i(\tau) := v_{i+1}(\tau)$ for $i = k+1, \ldots, m$ and $\tau \in (0, T_j]$. Also set $\mathcal{S}_k := \mathcal{S}_k \cup \mathcal{S}_{k+1} \cup \{j\}$, and $\mathcal{S}_i := \mathcal{S}_{i+1}$ for $i = k+1, \ldots, m$.

We leave out implementation details. We only note that the functions w_i and W_i are piecewise linear with at most $2n$ parts. Thus it is possible to represent and process them efficiently. The computation of T_j is efficient as well. To compute the parameter r, we need to design an algorithm for every particular restriction Ψ. In our applications we can use $r = r^{\Psi}(s_1, \ldots, s_m)$ as the optimal approximation can be computed for fixed speeds using linear programs.

Proof of Lemma 2.2

Fix a sequence of random bits used by A. For $i \leq k+1$, let T_i denote the last time when at least i machines are running the jobs from subsequence

$j_1, j_2, \ldots, j_k$; note that $T_{k+1} = 0$. First observe that

$$\sum_{i=1}^{k} p_{j_i} \quad \leq \quad \sum_{i=1}^{k} s_i T_i. \tag{4}$$

During the time interval $(T_{i+1}, T_i]$ at most i machines are busy with jobs from $(j_\ell)_{\ell=1}^{k}$, and their total speed is at most $s_1 + s_2 + \ldots + s_i$. Thus the maximum possible work done on $\mathcal{J}$ in this interval is $(T_i - T_{i+1})(s_1 + s_2 + \ldots + s_i)$. Summing over all $i = 1, \ldots, k$, we obtain $\sum_{i=1}^{m} s_i T_i$. In any valid schedule of $\mathcal{J}$ all the jobs are completed, so (4) follows.

Since the algorithm is semi-online, the schedule for $\mathcal{J}_{[j_i]}$ is obtained from the schedule for $\mathcal{J}$ by removing the jobs $j > j_i$. At time T_i there are at least i jobs from $(j_\ell)_{\ell=1}^{k}$ running, thus at least one job from $(j_\ell)_{\ell=1}^{k-i+1}$ is running. So we have $T_i \leq C^{\mathsf{A}}_{\max}[\mathcal{J}_{[j_{k-i+1}]}]$ for any fixed random bits. Averaging over random bits of the algorithm and using (4), we have

$$\sum_{i=1}^{k} p_{j_i} \quad \leq \quad \mathbb{E}\left[\sum_{i=1}^{k} s_i C^{\mathsf{A}}_{\max}[\mathcal{J}_{[j_{k-i+1}]}]\right] \;=\; \sum_{i=1}^{k} s_i \mathbb{E}\left[C^{\mathsf{A}}_{\max}[\mathcal{J}_{[j_{k-i+1}]}]\right]. \tag{5}$$

Since A is R-approximation algorithm, we claim that for any partial input $\mathcal{I} \in \mathrm{pref}(\Psi)$, we have $\mathbb{E}[C^{\mathsf{A}}_{\max}[\mathcal{I}]] \leq R \cdot C^{*,\Psi}_{\max}[\mathcal{I}]$: For $\mathcal{I} \in \Psi$ this follows from the definition of an approximation ratio of the semi-online algorithm. Otherwise this follows since the semi-online algorithm has $\mathbb{E}[C^{\mathsf{A}}_{\max}[\mathcal{I}']] \geq \mathbb{E}[C^{\mathsf{A}}_{\max}[\mathcal{I}]]$ for any end extension $\mathcal{I}'$ of $\mathcal{I}$ and $C^{*,\Psi}_{\max}[\mathcal{I}]$ is defined the an infimum for all such extensions in Ψ.

The bound in the lemma now follows by using the previous claim for each term of the right-hand side of (5), i.e., for each $\mathcal{I} = \mathcal{J}_{[j_{k-i+1}]}$ and reindexing the sum backwards. □

Proof of Theorem 2.4

If RatioStretch schedules a job, it is always completed at time $T_j \leq r \cdot C^{*,\Psi}_{\max}[(p_i)_{i=1}^{n}]$. Thus to prove the theorem, it is sufficient to guarantee that the algorithm does not fail to find machines V_k and V_{k+1} for the incoming job j. This is equivalent to the statement that there is always enough space on V_1, i.e., that $p_j \leq W_1(T_j)$ in the iteration when j is to be scheduled. Since $W_{m+1} \equiv 0$, this is sufficient to guarantee that required k exists. Given the choice of k, it is always possible to find time t_j as the expression $W_{k+1}(t_j) + W_k(T_j) - W_k(t_j)$ is continuous in t_j, for $t_j = 0$ it is equal to $W_k(T_j) \geq p_j$, and for $t_j = T_j$ it is equal to $W_{k+1}(T_j) \leq p_j$.

Consider now all the jobs scheduled on the first virtual machine, i.e., the set $\mathcal{S}_1$. Let $j_1 < j_2 < \cdots < j_{k-1}$ denote the jobs in $\mathcal{S}_1$, ordered as they appear on input. Finally, let $j_k = j$ be the incoming job.

Consider any $i = 1, \ldots, k$ and any time $\tau \in (0, T_{j_i}]$. Using the fact that the times T_j are non-decreasing in j and that the algorithm stretches each job j over the whole interval $(0, T_j]$, there are at least $k - i$ jobs from $\mathcal{S}_1$ running at τ, namely jobs $j_i, j_{i+1}, \ldots, j_{k-1}$. Including the idle machine, there are at least $k+1-i$ real machines belonging to V_1. Since V_1 is the first virtual machine and the real machines are adjacent, they must include the fastest real machines M_1, ..., M_{k+1-i}. It follows that the total work that can be processed on the real machines belonging to V_1 during the interval $(0, T_{j_m}]$ is at least $s_1 T_{j_m} + s_2 T_{j_{m-1}} + \cdots + s_m T_{j_1}$. The total processing time of jobs in $\mathcal{S}_1$ is $p_{j_1} + p_{j_2} + \cdots + p_{j_{k-1}}$. Thus to prove that j_k can be scheduled on V_1 we need to verify that

$$p_{j_k} \leq s_1 T_{j_k} + s_2 T_{j_{k-1}} + \cdots + s_k T_{j_1} - (p_{j_1} + p_{j_2} + \cdots + p_{j_{k-1}}).$$

Using $T_{j_i} = r \cdot C^{*,\Psi}_{\max}[\mathcal{J}_{[j_i]}]$, this is equivalent to the conclusion of Lemma 2.2

$$\sum_{i=1}^{k} p_{j_i} \quad \leq \quad r \cdot \sum_{i=1}^{k} s_{k+1-i} \cdot C^{*,\Psi}_{\max}[\mathcal{J}_{[j_i]}].$$

By the choice of r in the algorithm we know that there exist an semi-online r-approximation algorithm, thus Lemma 2.2 guarantees that the inequality indeed holds. □

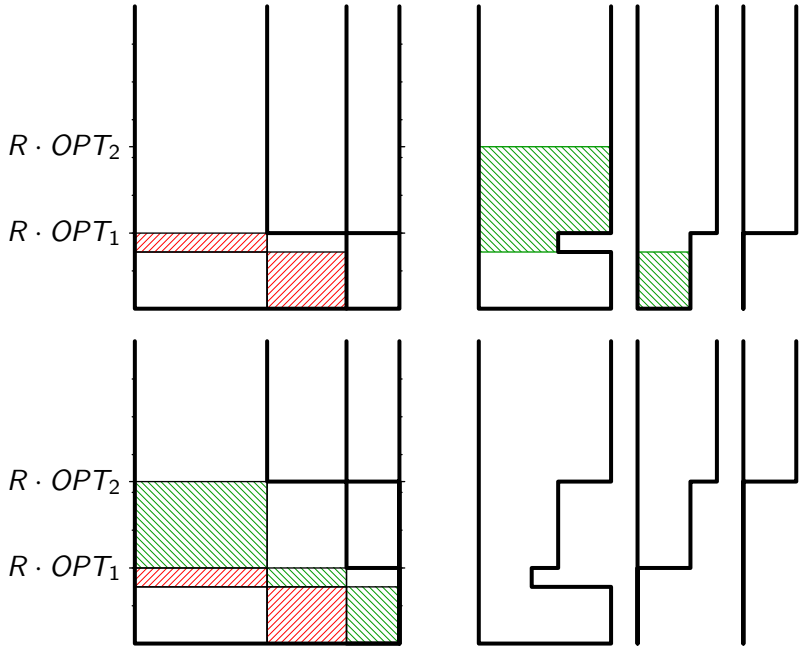

Figure 1: An illustration of a schedule of two jobs on three machines produced by RatioStretch. Vertical axis denotes the time, horizontal axis corresponds to the speed of the machines. The pictures on the left depict the schedule on the real machines, with bold lines separating the virtual machines. The pictures on the right show only the idle time on the virtual machines. The top pictures show the situation after the first job, with the second job being scheduled on the first two virtual machines. The bottom pictures show the situation after the second job is scheduled and virtual machines updated.

B Known sum of processing times, $m = 3$

Here we solve explicitly the linear program from Section 3.1 for $m = 3$ to illustrate our techniques for obtaining closed formulas for a fixed number of machines. Here the resulting formula is

$$r^{\sum p_j = P}(s_1, s_2, s_3) = \begin{cases} \dfrac{s_1(s_1+s_2)}{s_1^2+s_2^2} & \text{for } s_1^2 \le s_2(s_2+s_3) \\ 1 + \dfrac{s_2 s_3}{s_1(s_1+s_2+s_3)+s_2(s_1+s_2)} & \text{for } s_1^2 \ge s_2(s_2+s_3) \end{cases}$$

We know that the optimal approximation ratio is given by the following linear program for $n = 2$ jobs (as the case $n = 1$ is trivial). We write it explicitly:

$$\begin{array}{rcll} \textbf{maximize} & r = q_1 + q_2 & & \\ \textbf{subject to} & & & \\ 1 & = & s_1 O_2 + s_2 O_1 & (z_{norm}) \\ q_1 + q_2 & \le & (s_1+s_2+s_3) O_1 & (z_1) \\ q_1 + q_2 & \le & (s_1+s_2+s_3) O_2 & (z_2) \\ q_1 & \le & s_1 O_1 & (z_{1,1}) \\ q_1 + q_2 & \le & (s_1+s_2) O_2 & (z_{1,2}) \\ q_2 & \le & s_1 O_2 & (z_{2,2}) \\ q_1 & \le & q_2 & (z_{\le}) \\ 0 & \le & q_1 & (z_0) \end{array} \tag{6}$$

We can see that condition (z_2) is implied by condition $(z_{1,2})$. So we omit (z_2) in the following computations.

We distinguish two cases. In each case we simply give explicit primal and dual optimal solutions. The primal solution is essentially the hardest sequence of jobs, together with the values of O_i corresponding to the values of $C_{\max}^{*,\Psi}$ on the prefixes of the sequence. The dual solution gives a linear combination of the constraints such that if we add up these multiples of the constraint, we derive a tight upper bound on r.

Case I: $s_1^2 \le s_2(s_2+s_3)$**.**

Let $D = s_1^2 + s_2^2$. This will be the common denominator for all values in the feasible solution of this case.

The hardest sequence has two jobs: $p_1 = s_1 s_2/D$ and $p_2 = s_1^2/D$. These jobs induce a feasible solution, where $q_1 = p_1$, $q_2 = p_2$, $O_1 = s_2/D$, $O_2 = s_1/D$. We can see that $(z_{\le})$ and (z_0) are satisfied. Moreover $(z_{1,1})$, $(z_{2,2})$ and $(z_{1,2})$ are satisfied and an equality is attained. Thus it remains to

prove (z_1). After substitution we get $(s_1+s_2)s_1/D \le (s_1+s_2+s_3)s_2/D$. If we multiply both sides by D and subtract s_1s_2, we can see that this is equivalent to case condition. Finally, we check that the objective value is $s_1s_2/D + s_1^2/D = s_1(s_1+s_2)/(s_1^2+s_2^2)$ which is equal to the claimed bound.

To demonstrate the dual solution, we add up inequalities in (6) with the following coefficients (each coefficient corresponds to the inequality with the same label): $z_{1,1} = z_{2,2} = s_2(s_1+s_2)$, $z_{1,2} = s_1(s_1-s_2)$, $z_{norm} = -s_1(s_1+s_2)$. We obtain the inequality

$$(q_1+q_2)(s_1^2+s_2^2) - s_1(s_1+s_2) \le 0.$$

Equivalently, we get $r = q_1+q_2 \le s_1(s_1+s_2)/(s_1^2+s_2^2)$. Thus for every feasible primal solution, r satisfies this bound.

This completes the proof that $s_1(s_1+s_2)/(s_1^2+s_2^2)$ is the optimal approximation ratio for three machines in Case I.

Case II: $s_1^2 \ge s_2(s_2+s_3)$**.**

Let $D = s_1(s_1+s_2+s_3) + s_2(s_1+s_2)$.

The worst sequence has two jobs: $p_1 = s_2(s_1+s_2+s_3)/D$ and $p_2 = s_1(s_1+s_2+s_3)/D$. These jobs induce a feasible solution, where $q_1 = p_1$, $q_2 = p_2$, $O_1 = (s_1+s_2)/D$ and $O_2 = (s_1+s_2+s_3)/D$. We can see that $(z_{\le})$ and (z_0) are satisfied. Moreover $(z_{1,2})$, $(z_{2,2})$ and (z_1) are satisfied and equality holds. We need to prove $(z_{1,1})$. After substitution we get $s_2(s_1+s_2+s_3)/D \le s_1(s_1+s_2)/D$, which is equivalent to the case condition. Finally, the objective value is $q_1+q_2 = s_2(s_1+s_2+s_3)/D + s_1(s_1+s_2+s_3)/D$, which is equal to the claimed bound.

Again we need to prove a matching upper bound. We add up inequalities in (6) with the following coefficients: $z_1 = s_2(s_1+s_2)$, $z_{1,2} = s_1(s_1+s_2+s_3)$, $z_{norm} = -(s_1+s_2)(s_1+s_2+s_3)$. We obtain $(q_1+q_2)D - (s_1+s_2)(s_1+s_2+s_3) \le 0$. This gives the upper bound $r = q_1+q_2 \le (s_1+s_2)(s_1+s_2+s_3)/D$ which is equal to the claimed bound. This completes the proof.

C Sequences with non-increasing processing times

Here we can prove two bounds on $\hat{r}_n$, the approximation ratio for jobs with non-increasing sizes.

Lemma C.1 *For any positive integers n and a and any speed vector $\mathbf{s}$ there exists a speed vector $\mathbf{s}'$ such that $\hat{r}_n(\mathbf{s}) \le \hat{r}_{an}(\mathbf{s}')$. Consequently $\hat{r}_n \le \hat{r}_{an}$.*

Proof: We choose $\mathbf{s}'$ so that it has a machines with each speed s_i. Formally, we set $s'_{ua-v} = s_u$ for any positive integer u and $v = 0, \ldots, n-1$. Let $S'_k = \sum_{i=1}^{k} s'_k$, and recall that $S_u = \sum_{i=1}^{u} s_u$.

Let $k = ua - v$ for some positive integer u and $v \in \{0, \ldots, n-1\}$. We claim that $S'_k/k \geq S_u/u$: We are comparing two averages of some sets of speeds. In S'_k we sum a copies of each speed in the sum S_u, except that v copies of the smallest speed are omitted; thus the average can only increase. Observe also that $s'_{an-k+1} = s_{n-u+1}$. Thus

$$\frac{ks'_{an-k+1}}{S'_k} \leq \frac{us_{n-u+1}}{S_u}. \tag{7}$$

In the middle step of the following derivation, we use (7) for each term in the sum (obtaining a equal terms for each u). The first and the last steps follow from (3).

$$(\hat{r}_{an}(\mathbf{s}'))^{-1} = \frac{1}{an}\sum_{k=1}^{an}\frac{ks'_{an-k+1}}{S'_k} \leq \frac{1}{an}\sum_{u=1}^{n} a\frac{us_{n-u+1}}{S_u} =$$

$$= \frac{1}{n}\sum_{u=1}^{n}\frac{us_{n-u+1}}{S_u} = (\hat{r}_n(\mathbf{s}))^{-1}.$$

This completes the proof of the lemma. □

Lemma C.2 *For any positive integers n and n', $\hat{r}_{n'} \leq \frac{n+1}{n} \cdot \hat{r}_n$.*

Proof: Let $N \geq n$ and let $a = \left\lfloor \frac{N+1}{n+1} \right\rfloor$. We prove that $\hat{r}_N \leq \frac{N}{an} \cdot \hat{r}_n$.

First we show that this implies the lemma. For $N \to \infty$, $\frac{N}{an}$ converges to $\frac{n+1}{n}$. Thus $\limsup_{N\to\infty} \hat{r}_N \leq \frac{n+1}{n} \cdot \hat{r}_n$. If N is a multiple of n', Lemma C.1 implies that $\hat{r}_{n'} \leq \hat{r}_N$. Since the multiples can be taken arbitrarily large, together with the limit property above this implies $\hat{r}_{n'} \leq \limsup_{N\to\infty} \hat{r}_N \leq \frac{n+1}{n} \cdot \hat{r}_n$.

Now consider an arbitrary speed vector $\mathbf{s}'$. We construct a speed vector $\mathbf{s}$ such that $\hat{r}_N(\mathbf{s}') \leq \frac{N}{an}\hat{r}_n(\mathbf{s})$. This implies that $\hat{r}_N \leq \frac{N}{an} \cdot \hat{r}_n$.

Denote again $S'_k = \sum_{i=1}^{k} s'_k$. We choose the speeds $\mathbf{s}$ so that we divide $\mathbf{s}'$ into groups of a speeds, and s_u is the sum of the speeds in the uth group. Formally, $s_i = \sum_{v=0}^{a-1} s'_{ua-v}$.

By (3) we have

$$(\hat{r}_N(\mathbf{s}'))^{-1} = \frac{1}{N}\sum_{k=1}^{an}\frac{ks'_{an-k+1}}{S'_k} \geq \frac{1}{N}\sum_{u=1}^{n}\sum_{v=0}^{a-1}\frac{(N-na+ua-v)s'_{na-ua+v+1}}{S'_{N-na+ua-v}}\ ,$$

where the inequality follows by dropping the first $N-an$ terms in the sum and grouping and reindexing the remaining ones, using the substitution $k = N-na+ua-1$ for some $u \geq 1$ and $v \in \{0,\ldots,a-1\}$.

Now we proceed towards bounding the inner sums. We have

$$\frac{S'_{N-na+ua-v}}{N-na+ua-v} \leq \frac{S'_{ua}}{ua},$$

as by the choice of a we have $N \geq an+a-1 \geq na+v$ and thus on the left-hand side we take the average of the same speeds as on the right-hand side, plus possibly some smaller ones. Thus we have

$$\sum_{v=0}^{a-1}\frac{(N-na+ua-v)s'_{na-ua+v+1}}{S'_{N-na+ua-v}} \geq \frac{ua}{S'_{ua}}\cdot\sum_{v=0}^{a-1}s'_{na-ua+v+1} = \frac{ua}{S_u}\cdot s_{n-u+1}\ .$$

Using this bound for the inner sums we have

$$(\hat{r}_N(\mathbf{s}'))^{-1} \geq \frac{1}{N}\sum_{u=1}^{n}\left(\frac{ua}{S_u}\cdot s_{n-u+1}\right) = \frac{na}{N}\cdot\frac{1}{n}\cdot\sum_{u=1}^{n}\frac{us_{n-u+1}}{S_u} = \frac{na}{N}(\hat{r}_n(\mathbf{s}))^{-1}\ .$$

This completes the proof of the lemma. □