# Semi-Online Preemptive Scheduling: One Algorithm for All Variants

Tomáš Ebenlendr* Jiří Sgall*

**Abstract**

The main result is 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. We derive some global relations between various restrictions and tight bounds on the approximation ratios for a small number of machines.

## 1 Introduction

We study 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 $p/s$. 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.

This problem, also known as list scheduling, was first studied in Graham's seminal paper [8] for identical machines (i.e., all speeds equal), where it was shown that the greedy algorithm (which is online) is a 2-approximation. Soon after that, Graham [9] proved that if the jobs are presented in non-increasing order of their processing times, the approximation ratio goes down to 4/3. This is no longer an online algorithm, as some partial knowledge of the input is given to the scheduler. Since then, many problems with such a partial information given in advance have been studied, see Section 1.2. This class of algorithms is nowadays called *semi-online algorithms*.

We focus on the *preemptive* version of scheduling, which means that each job may be divided into several pieces, which can be assigned to different machines in disjoint time slots. (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.) Note that Graham [8, 9] studied the non-preemptive version, but the cited result from [8] holds for the preemptive version, too. A (semi-)online algorithm for preemptive scheduling, upon arrival of a job 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 in the order in 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 provide a semi-online algorithm for preemptive scheduling on uniformly related machines which is optimal for any chosen semi-online restriction. This means not only the cases listed in the abstract—the restriction can be given as an arbitrary set of allowed input sequences. Note that, similarly to the online algorithms, for semi-online algorithms we generally do not require the computation to be polynomial time. However, except for computing the optimal approximation ratio, our algorithm is efficient. And, for the restrictions considered in the literature, even the computation of the optimal ratio is efficient.

This is a direct sequel to our (i.e., the authors and Wojtek Jawor) previous work [3], where we have designed an optimal online algorithm. The algorithm and the proof of the main result are similar, however, some technical issues have to be handled carefully to achieve the full generality of our new result. The new algorithm has all the nice features from [3]: It 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 approximation ratio of any randomized algorithm.

The algorithm first computes the optimal approximation ratio for the given semi-online restriction and combination of speeds. Then, taking this as a parameter, it schedules each incoming job as slowly as possible, accounting for the jobs seen so far and all possible extensions of the input sequence. The precise specification of the schedule relies on the notion of a virtual machine developed in [4, 3].

One research direction is to analyze the exact approximation ratio for particular cases. This was done for a small number of machines for various restrictions in [5, 2, 16], and in many more cases for non-preemptive scheduling. Typically, the results involve similar but ad hoc algorithms and an extensive case analysis which is tedious to verify. Using our framework the algorithm is always the same and, for the basic restrictions, the approximation ratio can be computed by solving one or a few linear programs with speeds as parameters. To give the ratio, even as a formula in terms of speeds, is then 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. The details of analysis are omitted due to lack of space.

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. Only for identical machines, the exact ratio for any number of machines is known for the online case [1], where it tends to $e/(e-1) \approx 1.58$, and for non-increasing jobs [14], where it tends to $(1+\sqrt{3})/2 \approx 1.366$.

We are able to prove certain relations between the overall approximation ratios for different variants. Typically, the hard instances have non-decreasing processing times. One expected result is thus that the restriction to non-increasing processing times gives the same approximation as when all jobs have equal processing times, even for any particular combination of speeds, see Section 3.3. This phenomenon is already implicit in [14] for identical machines.

Some basic restriction form an inclusion chain, namely: known maximal processing time (or, more precisely, the equivalent restriction that the first job has maximal processing time), non-increasing processing times, equal processing times. The last two have the same approx-

imation ratio. We prove that this ratio is at most 1.52, see Section 3.3, while for known maximal processing time we have a computer-generated hard instance with approximation ratio 1.88. 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 than for identical machines, where knowing the maximal size is equally powerful as knowing that all the jobs are equal, see [14].

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 the job sequence, taking a prefix, and extending the job sequence by small jobs at the end. Thus, for example, we can conclude that 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 [14]. (Note that ratio with equal jobs and known sum is 1; the restriction to equal jobs does not allow padding.)

### 1.1 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 length, or processing time, of $j$th 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 $t_1 s_1 + t_2 s_3 + \cdots + t_m s_m = p_j$. 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 $\mathsf{A}$, let $C^{\mathsf{A}}_{\max}[\mathcal{J}]$ be the makespan of the schedule of $\mathcal{J}$ produced by $\mathsf{A}$. By $C^{*}_{\max}[\mathcal{J}]$ we denote the makespan of the optimal offline schedule of $\mathcal{J}$. An algorithm $\mathsf{A}$ 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 [12, 7, 4].

### 1.2 Semi-online restrictions and previous work

Now we are interested in semi-online algorithms. The algorithm is given some information in advance in semi-online model. We define a general semi-online input restriction to be simply a set $\Psi$ of allowed inputs, also called *feasible sequences*. We call a sequence an *input sequence* if it is a prefix of some feasible sequence. Thus input sequences are exactly the partial inputs that the algorithm can see. Below we list some of the restrictions that are studied in literature.

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

**Non-increasing job sizes, denoted *decr*.** $\Psi$ 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 [5] and for identical machines in [14]. 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 optimtimal makespan, $C^*_{\max} = T$.** For a given value $\bar{T}$, $\Psi$ contains all sequences with $C^*_{\max}[\mathcal{J}] = \bar{T}$. In this case 1-approximation semi-online algorithm is known for any speeds, see [4].

**Known maximal job size, $p_{\max} = p$.** For a given value $\bar{p}$, $\Psi$ 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. In [14] 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 $\alpha$, $\Psi$ contains all sequences with $p_j \in [p, \alpha p]$ for each $j$. Tight bounds for preemptive scheduling for $m = 2, 3$ were given in [2, 16]. Non-preemptive version for 2 and 3 identical machines was studied in [11, 10].

**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 in [16] without preemption.

The paper [13] is probably the first paper which studied and compared several notions of semi-online algorithms. Some combination of the previous restrictions were studied in [15] 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.

The semi-online restrictions listed above are essentially of two kinds. The first type are those restriction that put conditions on individual jobs, their relations, or their order. These restrictions are closed under taking subsequences, i.e., any subsequence of a feasible sequence

is also feasible. 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. As we shall see later, for these restriction, computation of the optimal approximation ratio is simpler.

**Definition 1.1** *An input restriction $\Psi$ is* proper *if for any $\mathcal{J} \in \Psi$ and any subsequence $\mathcal{I}$ of $\mathcal{J}$ there exists an end extension $\mathcal{I}'$ of $\mathcal{I}$ such that $\mathcal{I}' \in \Psi$.*

## 2 The optimal algorithm

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 subsequence, this would be too pessimistic. It may happen that the current input is not feasible and we know that any feasible extension 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 current optimal makespan.

**Definition 2.1** *For an input restriction $\Psi$ and an input sequence $\mathcal{J}$, we define the optimal makespan as infimum over all possible end extensions of $\mathcal{J}$ that satisfy $\Psi$:*

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

This definition is meaningful for any input sequence (i.e., for any prefix of some $\mathcal{J} \in \Psi$). For $\mathcal{J} \in \Psi$ the values of $C^*_{\max}[\mathcal{J}]$ and $C^{*,\Psi}_{\max}[\mathcal{J}]$ are equal. Furthermore, for a proper restriction $\Psi$, $C^{*,\Psi}_{\max}[\mathcal{I}] \le C^{*,\Psi}_{\max}[\mathcal{J}]$ whenever $\mathcal{I}$ is a subsequence of $\mathcal{J}$.

**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 [4, 3]. We define the $i$th *virtual machine*, denoted $V_i$, so that at each time $\tau$ it contains the $i$th fastest machine among those real machines $M_1$, $M_2$, ..., $M_m$ that are idle at time $\tau$. 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 give a more formal description of the algorithm in Appendix A.

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 $r$ and the proof is included in Appendix C.

**Optimality of Algorithm** RatioStretch

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

This is based on a lemma of Epstein and Sgall [6] which provides the optimal lower bounds for online algorithms. The key observation is that on an input $\mathcal{J}$, if the adversary stops the input sequence at the $i$th job from the end, any $r$-approximation algorithm must complete by time $r$ times the current 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 this to arbitrary input restrictions, we need to deal with two issues.

First, the adversary cannot stop the input if the current part is not in $\Psi$. 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, if $\Psi$ is not proper, this bound would not be optimal. Suppose, e.g., that $\Psi$ contains sequences that at each even position have jobs with size 0. Then we need to ignore these jobs and stop the sequence only after the non-trivial jobs. To deal with that, we choose from $\mathcal{J}$ a subsequence of important jobs, but bound their work in terms of values $C^{*,\Psi}_{\max}$ of the prefixes of the original sequence $\mathcal{J}$. This allows us to prove that our algorithm is optimal for any $\Psi$. However, we observe that if $\Psi$ is proper, then this is not needed, as $C^{*,\Psi}_{\max}[\mathcal{I}] \le C^{*,\Psi}_{\max}[\mathcal{J}]$ whenever $\mathcal{I}$ is a subsequence of $\mathcal{J}$, and thus we only need the case where an input sequence is considered, see Observation 2.4.

The proof of the lemma is along the lines of [6] and is given in Appendix B

**Lemma 2.2** *Given any randomized $R$-approximation semi-online algorithm* A *for preemptive scheduling on $m$ machines with an input restriction $\Psi$, then for any input sequence $\mathcal{J} \in \Psi$ 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 $r^{\Psi}$ to be the largest bound obtained by Lemma 2.2 for the given input, or for the largest bound for the given speeds and any instance, or the overall bound for the restriction $\Psi$.

**Definition 2.3** *for any vector of speeds $\mathbf{s}$ and any sequence of $\mathcal{J}$ which is an input sequence, we define*

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

*For any vector of speeds $\mathbf{s}$, let $r^{\Psi}(\mathbf{s}) = \sup_{\mathcal{J}} r^{\Psi}(\mathbf{s},\mathcal{J})$; the supremum is taken over all input sequences.*

*Finally, let $r^{\Psi} = \sup_{\mathbf{s}} r^{\Psi}(\mathbf{s})$.*

**Observation 2.4** *For any proper* $\Psi$*,*

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

*where the supremum is taken over all initial segments of sequences in* $\Psi$*.*

Thus, we will use another definition of $r^{\Psi}(\mathbf{s}, \mathcal{J})$ for proper restrictions:

**Definition 2.5** *Let* $\Psi$ *be proper semi-online restriction, then we redefine*

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

This definition is not consitent with previous of $r^{\Psi}(\mathbf{s}, \mathcal{J})$, in fact $\mathcal{J}$ here plays role of the sequence $(p_{j_i})_{i=1}^{k}$ in previous definition. The value of $r^{\Psi}(\mathbf{s})$ stays same for both definitions of $r^{\Psi}(\mathbf{s})$ for proper restrictions.

With these definitions and Lemma 2.2, it is easy to prove the following theorem, see Appendix C. Essentially, the chosen subsequence in Lemma 2.2 is taken to be the jobs on the first virtual machines and the incoming job, and we argue that if Algorithm RatioStretch cannot schedule the incoming jobs, Lemma 2.2 says that no (randomized) algoritm with same aproximation ratio is able to.

**Theorem 2.6** *For any restriction* $\Psi$ *and vector of speeds* $\mathbf{s}$*, Algorithm* RatioStretch *is* $r \geq r^{\Psi}(\mathbf{s})$ *approximation algorithm for semi-online preemptive scheduling on* $m$ *uniformly related machines.*

*In particular,* $r^{\Psi}(\mathbf{s})$ *(resp.* $r^{\Psi}$*) is the optimal approximation ratio for semi-online algorithms for* $\Psi$ *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 $\Psi$ as a supremum over a bound for all input sequences. Even if $\Psi$ is proper, it is not obvious how to turn this into an efficient algorithm. In this section, we develop a general methodology how to compute the ratio using linear programs and apply it to a few cases.

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 what are the biggest jobs and we can express the optimal makespans for prefixes by linear constraints in job sizes. Maximizing the expression for $r^{\Psi}(\mathbf{s})$ 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 3.1** *Let* $\Psi$ *be arbitrary proper restriction and* $\mathbf{s}$ *be arbitrary speed vector. Let* $\mathcal{J}$ *and* $\mathcal{J}'$ *be two input sequences with* $n$ *jobs. Suppose that for some* $b > 0$*:*

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

*then* $r(\mathbf{s}, \mathcal{J}') \geq r(\mathbf{s}, \mathcal{J})$*.*

It follows immediately from the definition of $r^{\Psi}(\mathbf{s}, \mathcal{J})$. If some choice of a subsequence provides a bound on $r(\mathbf{s}, \mathcal{J})$, we use the same subsequence to bound $r(\mathbf{s}, \mathcal{J}')$, and due to the assumptions the value cannot go down.

In particular, the observation implies that whenever $\Psi$ is closed under permutations of the sequence and increasing the size of the last job of a sequence 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 $p_k > p_{k+1}$, swaping them does not change any $C^{*,\Psi}_{\max}[\mathcal{J}_{[i]}]$ with the exception of $C^{*,\Psi}_{\max}[\mathcal{J}_{[k]}]$, and that one can only increase; 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 $\Psi$ 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, Observation 3.1 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 an input sequence. If $P < \bar{P}$, we scale up $\mathcal{J}$ to $\mathcal{J}'$. Observation 3.1 then applies with some $b < 1$, as each $C^{*,\Psi}_{\max}[\mathcal{J}'_{[i]}] = \max\{C^{*}_{\max}[\mathcal{J}'_{[i]}], \bar{P}/S\}$ increases by at most the scaling factor. Finally, we observe that we can restrict ourselves to sequences $\mathcal{J}$ with less than $m$ jobs. If $n \geq m$, we have $C^{*,\Psi}_{\max}[\mathcal{J}_{[i]}] \geq \bar{P}/S$ for any $i$ and thus $r^{\Psi}(\mathbf{s}, \mathcal{J}) = P/(\sum_{i=1}^{n} s_{n+1-i} \cdot C^{*,\Psi}_{\max}[\mathcal{J}_{[i]}]) \leq P/(\sum_{i=1}^{n} s_{n+1-i} \cdot \bar{P}/S) = 1$, using $m \leq n$ 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 $\Psi$.) To obtain the worst case bound, we compute $m-1$ linear programs, each for one value of $n$, and take the maximum of their solutions. The linear program for given $P$, $\mathbf{s}$, and $n$ has variables $q_i$ for job sizes and $O_i$ for optimal makespans of the initial segments:

$$
\begin{array}{rcll}
\textbf{minimize} & r^{-1} = \dfrac{s_1 O_n + s_2 O_{n-1} + \cdots + s_n O_1}{P} & & \\
\textbf{subject to} & & & \\
q_1 + \cdots + q_n & = & P & \\
P & \leq & (s_1 + s_2 + \cdots + s_m) O_k & \text{for } k = 1, \ldots, n \\
q_j + q_{j+1} + \cdots + q_k & \leq & (s_1 + s_2 + \cdots + s_{k-j+1}) O_k & \text{for } 1 \leq j \leq k \leq n \\
q_j & \leq & q_{j+1} & \text{for } j = 1, \cdots, n-1 \\
0 & \leq & q_1 &
\end{array}
\tag{2}
$$

We can see that $r^{-1}$ is independent of value $P$.

We now examine the special cases of $m = 2, 3$. The linear program is trivial for $n = 1$, as we can always schedule one job optimally. So for two machines the approximation ratio is equal to 1. We can see this also intuitively: if optimal makespan is determined by $\sum p/(s_1+s_2)$ then the 1-approximation algorithm for known optimum works. We follow this algorithm until it fails. In that case the optimal makespan is $p_{max}/s_1$, then all other jobs are scheduled on the slow machine and when job with size $\sum p s_1/(s_1 + s_2)$ is encountered, it is scheduled on the fast machine. Then the rest of jobs is scheduled on second machine.

For $m = 3$, it remains to solve the case when $n = 2$. The ratio is:

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

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 the padding lemma 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 machine the improvement is significant.

We say that $\Psi$ 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 $\Psi$ 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 some number of equal jobs of size $\varepsilon < \varepsilon_0$ at the end satisfies $\mathcal{J}' \in \Psi$.

**Theorem 3.2** *Suppose that $\Psi$ 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*

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

*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 $\Psi$ allows scaling, the value $C^{*,\Psi}_{\max}[\mathcal{J}]$ is multiplied by $b$ when $\mathcal{J}$ is scaled by $b$. Consequently, the value of $r^{\Psi}(\mathbf{s}, \mathcal{J})$ 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 $r^{\Psi}(\mathbf{s}, \mathcal{J}') = r^{\Psi}(\mathbf{s}, \mathcal{J})$.

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 $\sigma$. The first condition on $\sigma$ 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 $\Psi$ is closed under taking prefixes, $C^{*,\Psi}_{\max}$ equals $C^{*}_{\max}$ for any sequence. Thus we conclude that $r^{\Psi}(\mathbf{s}', \mathcal{J}') \ge r^{\Psi}(\mathbf{s}, \mathcal{J}')/(1+\delta)$.

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 $r^{\Psi,\sum p_j=P}(\mathbf{s}', \mathcal{J}') = r^{\Psi}(\mathbf{s}', \mathcal{J}')$.

Chaining the inequalities at the end of the previous three paragraphs yields the statement of the theorem. □

### 3.2 Known maximal job size, $p_{\max} = p$

Here we are given size of maximal job $\bar{p}$. As noted before, any algorithm that works with 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 processing time 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 beginning. Other jobs are scheduled in the same way in both algorithms. Thus we can work with the equivalent restriction where the first job is maximal. Then $C^{*,\Psi}_{\max}[\mathcal{J}] = C^{*}_{\max}[\mathcal{J}]$ for any input sequence. Furthermore, by Observation 3.1, the other jobs can be swapped as in previous case, and we can maximize only over non-decreasing sequences (with the exception of the first job).

Again we want to solve $m$ linear programs, one for sequences of $n$ jobs for each $n < m$ and one for $n \geq m$ jobs. We will show the program for $n \geq m$ jobs. Program for $n = m$ can be derived by setting $q_1 = 0$, and programs for smaller $n$ are in fact programs for $n = m$, as we do not use last $m - n$ machines.

We want to solve following non-linear programs that can be linearized by theirs homogeneity. The variable $q_1$ represents the total processing time of jobs between $p_1$ and $p_{n-m+2}$.

$$\begin{aligned}
\textbf{maximize} \quad & r = \frac{P}{s_1 O_m + s_2 O_{m-1} + \cdots + s_m O_1} \\
\textbf{subject to} \quad & \\
p + q_1 + \cdots + q_m &= P \\
p + q_1 + \cdots + q_k &\leq (s_1 + s_2 + \cdots + s_m) O_k && \text{for } k = 1, \ldots, m \\
p + q_{j+1} + q_{j+2} + \cdots + q_k &\leq (s_1 + s_2 + \cdots + s_{k-j+1}) O_k && \text{for } 1 \leq j \leq k \leq m \\
q_j &\leq q_{j+1} && \text{for } j = 2, \cdots, m-1 \\
0 &\leq q_1 \\
q_m &\leq p \\
0 &\leq q_2
\end{aligned} \tag{3}$$

Because of homogenity $r$ is independent of $p$, so we can choose $p$ to be a variable and the denominator of the objective function to be a constant. Thus we add following constraint

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

**Further simplifications**

Now, after examining the linear programs, we can further restrict the set of sequences. We can assume that the last job is equal to first (maximal) job: The sequence is sorted so that the first job is the maximal job and last job is second largest job in the sequence. So we can replace both first and last jobs with jobs of size $(p_{max} + p_n)/2$. As $p_n$ is used only to compute optimum of whole sequence, and it occurs in all inequalities only with $p_{max}$, the optima may not increase and Observation 3.1 applies again.

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

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

The is maximum is 1.2 for $s_1 = s_2$.

For three machines we get

$$r^{p_{\max}=p}(s_1,s_2,s_3) = \begin{cases} \dfrac{S^2+s_1S}{S^2+s_1^2} = 1 + \dfrac{s_1(s_2+s_3)}{S^2+s_1^2} & \text{for } s_1s_2 \geq s_3S \\ \dfrac{S^2+2s_1S}{S^2+2s_1^2+s_1s_2} = 1 + \dfrac{s_1s_2+2s_1s_3}{S^2+2s_1^2+s_1s_2} & \text{for } s_1s_2 \leq s_3S \end{cases}$$

The first case is optimized by $s_1 = s_2 = 1, s_3 = \sqrt{2}-1$. This gives ratio of $(1+\sqrt{2})/2 \approx 1.207$. The second case is optimized by $s_1 = 2, s_2 = s_3 = \sqrt{3}$, giving the ratio $(8+12\sqrt{3})/23 \approx 1.252$.

### 3.3 Non-increasing jobs, *decr*

We are also interested in sequences of non-increasing jobs, as this is one of the most studied restrictions. Now $\Psi$ 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 a job of the average size $(p_j + p_{j+1})/2$. By Observation 3.1, the approximation ratio does not decrease. Similarly, we can replace longer segement of jobs with only two distinct sizes by all jobs of the same 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. We only need to determine the length which gives the highest ratio.

W.l.o.g. all jobs have $p_j = 1$. Then $C^{*,\Psi}_{\max}[\mathcal{J}] = C^{*}_{\max}[\mathcal{J}] = n/S_n$. (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 2.4

$$r^{decr}(\mathbf{s},\mathcal{J}) = \frac{n}{\sum_{k=1}^{n} \frac{ks_{n-k+1}}{S_k}} \ .$$

Having this formula, it would appear to be easy to find the worst case speeds for each $n$ and thus to obtain the overall ratio. This amounts to an intriguing geometric question which we are unable to solve. Suppose we have numbers $x_i, y_i, i = 1, \ldots, n$ such that $x_iy_i = i$ for all $i$ and each sequence $x_i$ and $y_i$ is non-decreasing. Consider the union of rectangles $[0, x_i] \times [0, y_{n+1-i}]$ over all $i$; this is a sort of staircase part of the positive quadrant of the plane. What is the smallest possible area of this union of rectangles? After some numerical experiments, we conjecture that the optimum has $y_1 = y_2 = \ldots = y_k$ and $x_{k+1} = x_{k+2} = \ldots = x_n$ for some $k$. This would imply that the overall competitive ratio is the same as for identical machines, which is about 1.366 by [14].

Let us denote $r_n = \sup_{\mathbf{s}} r(\mathbf{s},\mathcal{J})$. We can prove certain relations between these values. In particular, for any integers $a$ and $n$, $r_{an} \geq r_n$, and $\sup_{n'} r_{n'} \leq \frac{n+1}{n} r_n$. 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 converse 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 omitted due to the space limit. Together these relations show that whenever we are able to evaluate some $r_n$ for a fixed $n$, the overall ratio is between $r_n$ and $\frac{n+1}{n} r_n$.

For $n = 2$ we get formula:

$$r^{decr}(s_1,s_2,s_3) = \left(\frac{s_2}{2s_1} + \frac{s_1}{s_1+s_2}\right)^{-1}$$

Theis maximum is $r_2 = \frac{4}{7}\sqrt{2} + \frac{2}{7} \approx 1.094$ for $s_2 = (\sqrt{2}-1)s_1$.

For $n = 3$ the formula is

$$r^{decr}(s_1, s_2, s_3) = \left( \frac{s_3}{3s_1} + \frac{2s_2}{3(s_1 + s_2)} + \frac{s_1}{s_1 + s_2 + s_3} \right)^{-1}$$

The maximum is $r_3 = 1.2$ for $s_1 = s_2 = 1$, $s_3 = 0$. This yields global upper bound of $r_n \leq \frac{4}{3} \cdot \frac{6}{5} = 1.6$. By a computer-assisted proof we have shown that $r_4 = (\sqrt{7} + 1)/3 \approx 1.215$ yielding global upper bound of $r_n \leq \frac{5}{12}(\sqrt{7} + 1) \approx 1.52$.

### 3.4 Other variants

In the full version of the paper we will demonstrate a few other semi-online restrictions. At this point, this becomes a somewhat mechanical exercise; we have not found any surprising phenomenon in the cases we have examined so far.

If $\Psi$ 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.

The case of tightly grouped jobs is more interesting. We set up the usual linear program with additional constraints on variables corresponding to individual jobs. The linear program also has a variable for the sum of processing times of the sequence of small jobs, before the last large jobs. If the processing times are in $[p, \alpha p]$, this variable needs to be constrained to a union of intervals $[p, \alpha p]$, $[2p, 2\alpha p]$, etc. Eventually the intervals start to overlap, thus there is a finite number of them and we can solve one linear program for each interval. However, the number of intervals goes to infinity as $\alpha$ goes to 1. This exactly corresponds to analytical expressions from [11, 10], which have infinite number of cases for $\alpha$ close to 1.

**Conclusions.** We have provided a general algorithm for semi-online preemptive scheduling. Now, instead of re-inventing the whole machinery for new cases of interest and going through the ad hoc case analysis, we can just analyze the appropriate linear programs as we have demonstrated on several special cases.

The most interesting question in this area is if some similar results can be proven for non-preemptive algorithms. There we do not have the same understanding of the optimal makespan, so perhaps one can only hope for some reasonably good bound on the competitive ratios. The current best overall upper bounds are 5.828 for deterministic and 4.311 for randomized algorithms, while the lower bounds are only 2.438 and 2. Also for a fixed number of machines very little is known.

## References


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

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

[3] 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.

[4] 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.

[5] 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.

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

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

[8] R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical J.*, 45:1563–1581, 1966.

[9] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math.*, 17:263–269, 1969.

[10] Y. He and G. Dósa. Semi-online scheduling jobs with tightly-grouped processing times on three identical machines. *Discrete Appl. Math.*, 150(1):140–159, 2005.

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

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

[13] 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.

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

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

[16] Z. Tan and Y. He. Semi-online problems on identical machines with inexact partial information. In *Proc. 11th Annual International Computing and Combinatorics Conf. (COCOON)*, volume 3595 of *Lecture Notes in Comput. Sci.*, pages 297–307. Springer, 2005.

# A The Optimal Algorithm: Description

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.

See Figure 1 for an example of a step of the algorithm.

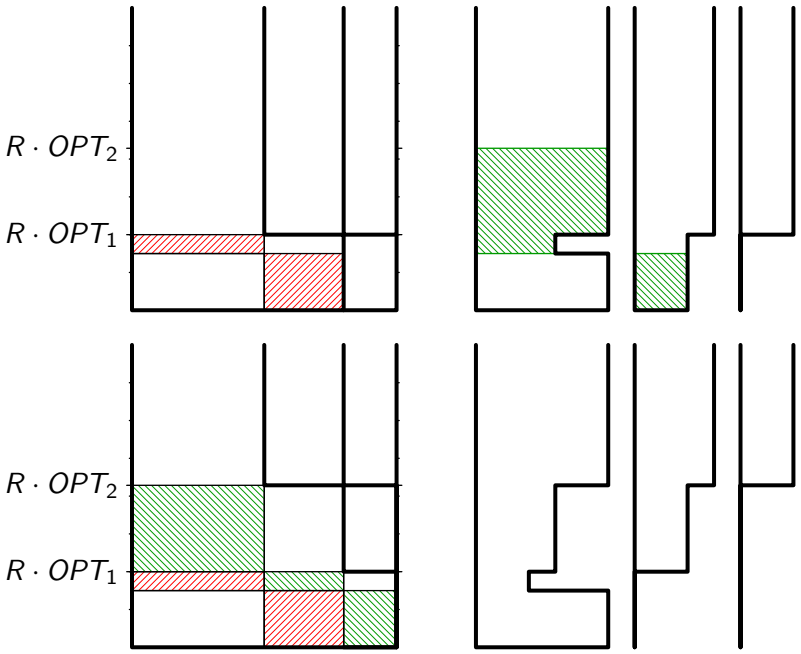


Figure 1: An illustration of a schedule of two jobs on three machines produced by RatioStretch. Vertical axis denotes the time, horizontal axis corresponds to the speed of the machines. The pictures on the left depict the schedule on the real machines, with bold lines separating the virtual machines.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

At each time $\tau$, machine $M_{i'}$ belongs to $V_i$ if it is the $i$th fastest idle machine at time $\tau$, or if it is running a job $j \in \mathcal{S}_i$ at time $\tau$. At each time $\tau$ 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 the optimal makespan.

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 $t$. Note also that $W_{m+1}(t) = v_{m+1}(t) = 0$ for all $t$.

**Algorithm** RatioStretch**.** Let a parameter $r \geq r^{\Psi}(s_1, \ldots, s_m)$ be the optimal objective value

or its approximation. Initialize $T_0 := 0$, $\mathcal{S}_i := \emptyset$, $v_i(\tau) := s_i$, and $v_{m+1}(\tau) := 0$ for all $i = 1, 2, \ldots, m$ and $\tau \geq 0$.
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 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 comput the parameter $r$, we need to design effective computation of $r$ for every particular restriction $\Psi$. 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.

# B Proof of Lemma 2.2

Having a witness sequence $\mathcal{J}$ that forbids approximation ratio $R$ according to Lemma 2.2, adversary will submit jobs from this sequence one by one until it finds that makespan of algorithm's output is larger than $RC^{*,\Psi}_{\max}[\mathcal{J}_i]$, where $\mathcal{J}_i$ is sequence of jobs submitted so far. Then it has to find $\mathcal{J}' = \operatorname{argmin}_{\mathcal{J}' \supseteq \mathcal{J}_i}\{C^{*,\Psi}_{\max}[\mathcal{J}'] \mid \mathcal{J}' \in \Psi\}$ (or some good approximation if minimum does not exist) and finishes the input sequence with jobs from $\mathcal{J}' \setminus \mathcal{J}_i$. Algorithm's makespan will not decrease, and $\mathcal{J}'$ sequence ensures that $C^{*,\Psi}_{\max}$ will not increase. This way adversary wins the game and proves that algorithm is not $R$-competitive.

Fix a sequence of random bits used by $\mathsf{A}$. Let $T_i$ denote the last time when at most $i$ machines are running the jobs from subsequence $j_1, j_2, \ldots, j_k$. 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 $i$, 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 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

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

Since $\mathsf{A}$ is $R$-competitive, i.e., $\mathbb{E}[C^{\mathsf{A}}_{\max}[\mathcal{J}_{[j_{k-i+1}]}]] \leq R \cdot C^{*,\Psi}_{\max}[\mathcal{J}_{[j_{k-i+1}]}]$, the lemma follows, reindexing the sum on the right-hand side backwards. □

## C Proof of Theorem 2.6

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

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

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. □