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.

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.

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.

The online scheduling problem, also known as list scheduling, was first studied in Graham's seminal paper [14] 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 [15] studied a semi-online variant: He proved that if the jobs are presented in non-increasing order of their processing times, the approximation ratio goes down to 4/3. (Note that Graham [14, 15]

*Institute of Mathematics, AS CR, Žitná 25, CZ-11567 Praha 1, Czech Republic. Email: ebik@math.cas.cz.

†Dept. of Applied Mathematics, Faculty of Mathematics and Physics, Charles University, Malostranské nám. 25, CZ-11800 Praha 1, Czech Republic. Email: sgall@kam.mff.cuni.cz.

‡Partially supported by Inst. for Theor. Comp. Sci., Prague (project 1M0545 of MŠMT ČR) and grant IAA100190902 of GA AV ČR.

studied the non-preemptive version, but the cited result from [14] holds for the preemptive version, too.)

Since then, 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 time 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].

General results

We give 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 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, the 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.

Results for specific restrictions

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 [9, 4, 16, 17, 20] 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. Typically the verification is quite simple for $m = 3$ or $m = 4$. We present several examples of such results for $m = 3$. For a more detailed discussion of the automated techniques and examples of calculations for $m = 4$, see [5].

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 [6]. 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 $(1+\sqrt{3})/2 \approx 1.366$ [22].

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 processing time (which is equivalent to known maximal processing time) include the inputs with non-increasing processing times, which in turn include the inputs with all jobs of equal processing time. 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 processing times, even for any particular combination of speeds. The overall approximation ratio of these two equivalent problems (non-increasing and the largest job first) is at most 1.52, see Section 5. On the other hand, for known maximal processing time of a job we have a computer-generated hard instance with approximation ratio 1.88 with $m = 120$. Thus restricting the jobs to be non-increasing helps the algorithm much more than just knowing the maximal processing time of a job. This is very different from identical machines, where knowing the maximal processing time is equally powerful as knowing that all the jobs are equal, see [22].

More interestingly, the overall approximation ratio with known sum of processing times is the same as in the purely 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 6.1. 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 [22]. (Note that the ratio with equal jobs and known sum is 1; the restriction to equal jobs does not allow padding.)

Organization of the paper

In Section 2, in addition to standard preliminaries, we introduce various semi-online restrictions and survey the specific results, old and new. In Section 3 we prove our general results, namely we present the algorithm, prove its optimality, and also introduce notation for the values of the optimal ratio. In Section 4 we introduce a notion of a proper restriction, which encompasses the usual semi-online restrictions, and allows to simplify the calculations for the specific restrictions. Here we also survey the general principles that allow us to construct linear programs for specific restrictions. The rest of the paper, Sections 5 to 10 is devoted to the specific restrictions. One unifying purpose is to demonstrate various aspects and possibilities in calculating the optimal ratio, usually using linear programming. And, not the least, we present there the specific results, both the general ones mentioned above and specific bounds typically for three machines.

2 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 so that speeds are non-increasing, 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 the speeds is $S = \sum_{i=1}^{m} s_i$ and $S_k = \sum_{i=1}^{k} s_i$ is the sum of the k largest speeds. To simplify the description of the algorithm, we assume that there are infinitely many machines of speed zero, i.e., we put $s_i = 0$ 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. Also, note that the case of m identical machines now corresponds to the case with $s_i = 1$ for $i \leq m$ and $s_i = 0$ for $i > m$. The machines M_i, including the machines of speed zero, are called real machines (to be distinguished from virtual machines introduced later in the algorithm).

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 the processing time, or the size, of the jth job. (Note that we allow jobs of size zero, which has technical advantages for formulation of linear programs, as we can easily avoid sharp inequalities; this is inessential as in all the cases we can replace such jobs by jobs with size going to 0 in the limit.) 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 $C_{\max}^{\mathsf{A}}[\mathcal{J}]$ 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., $C_{\max}^{\mathsf{A}}[\mathcal{J}] \leq R \cdot C_{\max}^{*}[\mathcal{J}]$. In case of a randomized algorithm, the same must hold for every input for the expected makespan of the online algorithm, $\mathbb{E}[C_{\max}^{\mathsf{A}}[\mathcal{J}]] \leq R \cdot C_{\max}^{*}[\mathcal{J}]$, where the expectation is taken over the random choices of the algorithm.

The optimal makespan can be computed as

$$C_{\max}^{*}[\mathcal{J}] = \max\left\{ \frac{P}{S}, \quad \max\left\{ \frac{P_k}{S_k} \,\middle|\, k = 1, \ldots, m-1 \right\} \right\}, \tag{1}$$

where P_k denotes the sum of the k largest processing times in $\mathcal{J}$ and S_k is the sum of the 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 the machines fully, and similarly the term for k is the minimal time when the work of the k largest jobs can be completed using the k fastest machines fully. The tightness of this bound is a classical result, see e.g. [19, 13, 7].

A useful observation is that the optimal set of jobs in (1) always contains all the jobs of the same size or none of them. More precisely, if the maximum in (1) is equal to P_k/S_k for

some k, then we may assume that the $(k+1)$st largest job is strictly smaller than kth largest job. This follows by a standard manipulation: If there are two or more jobs with equal size, and adding one of them to P_k increases the bound, then adding another job of the same size again increases the bound, using the fact that the speeds added to S_k are non-increasing.

In particular this implies that if there are m jobs of the same size, then either the optimum is P/S or the optimum is P_k/S_k for some k such that P_k does not contain any of these equal jobs. The possibility that P_k contains all of the jobs is now excluded, since it would imply that $k \geq m$ and such a bound is dominated by P/S.

2.1 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 a *partial input* if it is a prefix of some input sequence; the set of all partial inputs is denoted pref(Ψ). Thus the 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 that for any prefix $\mathcal{J}'$ of $\mathcal{J}$, $\mathbb{E}[C^{\mathsf{A}}_{\max}[\mathcal{J}']] \leq R \cdot C^*_{\max}[\mathcal{J}]$.

Below we list some of the restrictions that are studied in the literature, together with the notation that we are going to use, the previous work, and our results. We list previous work mainly in the preemptive case, the non-preemptive results are a narrow selection mainly of the papers that introduced the restriction. All our results are for the preemptive case.

Online scheduling. Here Ψ contains all sequences. In our (i.e., the authors and Wojtek Jawor) previous work [6], 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].

Known sum of processing times, denoted $\sum p_j = P$. For a given value $\bar{P}$, Ψ contains all sequences with $P = \bar{P}$. Note that we distinguish P, the general notation for the sum of the processing times in an input $\mathcal{J}$ (also used in the name of the restriction), and $\bar{P}$, the actual value given to the algorithm at the beginning; similarly for the other restrictions later. This restriction, for non-preemptive version on two machines, was studied in [21], which is probably also the first paper which studied and compared several notions of semi-online algorithms. We prove that the overall ratio is surprisingly the same as in the general online case, on the other hand we note that for two machines 1-approximation is possible, and we analyze the cases of three machines.

Non-increasing processing times, denoted *decr*. Here 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 [9] and for identical machines in [22]. 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, denoted $C^*_{\max} = T$. For a given value $\bar{T}$, Ψ contains all sequences with $C^*_{\max}[\mathcal{J}] = \bar{T}$. A 1-approximation semi-online algorithm is known for any $\mathbf{s}$, see [7].

Known maximal job processing time, denoted $p_{\max} = p$. 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 [18] for non-preemptive scheduling on 2 identical machines. The complete analysis of the preemptive version on two machines was given in [16]. In [22] 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. We also give a complete analysis for three machines.

Inexact partial information. In this case, some of the previously considered values (the optimal makespan, the sum of the processing times, the maximal processing time) is not known exactly but only up to a certain factor. These variants were studied first in [24] without preemption. The complete analysis of the preemptive version with approximately known optimum and maximal processing time on two machines and approximately known optimum on identical machines was given in [20]. We give a complete analysis for an approximately known optimum for three machines, denoted $T \leq C^*_{\max} \leq \alpha T$.

Tightly grouped processing times, denoted $p \leq p_j \leq \alpha p$. For given values $\bar{p}$ and α, Ψ contains all sequences with $p_j \in [\bar{p}, \alpha\bar{p}]$ for each j. This restriction was introduced in [18] for non-preemptive scheduling on 2 identical machines. The complete analysis of the preemptive version on two machines was given independently in [4, 16]. The case of three identical machines was analyzed in [17]. We sketch a complete analysis for two machines, reproving the results of [4, 16] in a simpler way.

Combined restrictions. In this case we have two different types of information. Technically, in our framework, we have $\Psi = \Psi' \cap \Psi''$ for some other restrictions Ψ' and Ψ''. We denote such a restriction Ψ', Ψ''. Some combinations of the previous restrictions were studied in [23] for non-preemptive scheduling on identical machines. We present two variants; the restriction of the known sum of processing time is combined either with the known maximal processing time or with non-increasing jobs.

We should note that there are also semi-online models that do not fit into our framework. Some of them can still be solved by extending our methods, for others we do not know a good solution at this point. We discuss such variants briefly in the conclusions.

3 The optimal algorithm

The new algorithm is based on the algorithm for online scheduling from [6]. In this section we present the algorithm and the proof of its optimality 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 (e.g., if the restriction forces that some large jobs will arrive later). In this case we can schedule the current job so that it complete much later than at time r times the current optimal makespan. For this purpose, we define the appropriate quantity to be used instead of the current optimal makespan.

Definition 3.1 *For an input restriction Ψ and a partial input $\mathcal{I} \in \mathrm{pref}(\Psi)$, we define the optimal makespan as the infimum over all possible end extensions of $\mathcal{J}$ that satisfy Ψ:*

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

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

3.1 Description of the algorithm

Our algorithm takes as a parameter a number r which is the desired approximation ratio. Later we show that, for the right choice of r, our algorithm is optimal. Given r, we want to schedule each incoming job so that it completes at time $r \cdot C^{*,\Psi}_{\max}[\mathcal{J}_{[j]}]$. By the definition of $C^{*,\Psi}_{\max}[\mathcal{J}_{[j]}]$, any schedule for any possible extension of the current partial input will have makespan at least $C^{*,\Psi}_{\max}[\mathcal{J}_{[j]}]$, in particular $C^{*,\Psi}_{\max}[\mathcal{J}_{[j]}] \leq C^*_{\max}[\mathcal{J}]$. Thus, if each job j completes by time $r \cdot C^{*,\Psi}_{\max}[\mathcal{J}_{[j]}] \leq r \cdot C^*_{\max}[\mathcal{J}]$, we have an r-approximation algorithm.

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 [7, 6]. 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, ..., 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. Recall that the real machines include the machines of speed zero, thus V_i is always defined; scheduling a job on such a real machine means that the job is not running.

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 an appropriate value of r, as we show in Theorem 3.4.

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 assigned to this job are merged (during the time when j is scheduled to be running on them). 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. We leave out some implementation details. We only note that the functions w_i and W_i are piecewise

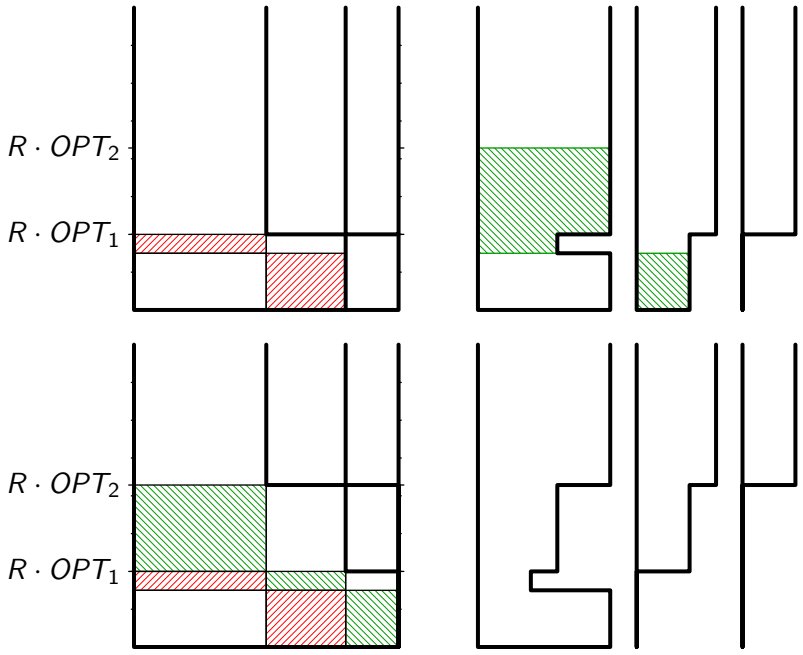

Figure 1: An illustration of a schedule of two jobs on three machines produced by RatioStretch. Vertical axis denotes the time, horizontal axis corresponds to the speed of the machines. The pictures on the left depict the schedule on the real machines, with bold lines separating the virtual machines.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

linear with at most $2n$ parts. Thus it is possible to represent and process them efficiently.

Algorithm RatioStretch**.** Let r be a parameter. 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 $i = k+1, \ldots, m$.

Before we analyze the algorithm, we make a few remarks concerning its efficiency and uniformity. The only parts of the algorithm that depend on the semi-online restriction are

(i) the computation of the parameter r, which should be equal to 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 Sections 5 to 10 it is shown that in many natural cases it reduces to linear programming or other more explicit expressions. Alternatively, we can use any upper bound on the approximation ratio and give it to the algorithm as the parameter r.

3.2 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 input $\mathcal{J}$ 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 [11] which provides the optimal lower bounds for online algorithms, as shown in [6]. 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 good 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 3.2 *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 $\mathcal{J} \in$* pref(Ψ)*, for any k, and for any subsequence of jobs $1 \le j_1 < j_2 < \cdots < j_k \le n$ we have*

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

Proof: Fix a sequence of random bits used by A and the corresponding schedule for $\mathcal{J}$. For $i \le k+1$, 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

$$\sum_{i=1}^{k} p_{j_i} \quad \le \quad \sum_{i=1}^{k} s_i T_i. \tag{2}$$

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 $(j_\ell)_{\ell=1}^{k}$ 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 from $(j_\ell)_{\ell=1}^{k}$ are completed, so (2) 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 $T_i \le C^{\mathsf{A}}_{\max}[\mathcal{J}_{[j_{k-i+1}]}]$ for any fixed random bits. Averaging over random bits of the algorithm and using (2), we have

$$\sum_{i=1}^{k} p_{j_i} \quad \le \quad \mathbb{E}\left[\sum_{i=1}^{k} s_i C^{\mathsf{A}}_{\max}[\mathcal{J}_{[j_{k-i+1}]}]\right] \quad = \quad \sum_{i=1}^{k} s_i \mathbb{E}\left[C^{\mathsf{A}}_{\max}[\mathcal{J}_{[j_{k-i+1}]}]\right]. \tag{3}$$

Since A is R-approximation algorithm, we claim that for any partial input $\mathcal{I} \in \mathrm{pref}(\Psi)$, we have $\mathbb{E}[C^{\mathsf{A}}_{\max}[\mathcal{I}]] \le R \cdot C^{*,\Psi}_{\max}[\mathcal{I}]$: 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 $\mathbb{E}[C^{\mathsf{A}}_{\max}[\mathcal{I}']] \ge \mathbb{E}[C^{\mathsf{A}}_{\max}[\mathcal{I}]]$ 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 (3), i.e., for each $\mathcal{I} = \mathcal{J}_{[j_{k-i+1}]}$ and reindexing the sum backwards. □

We define r^Ψ to be the largest lower bound on the approximation ratio obtained by Lemma 3.2.

Definition 3.3 *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, we can prove the following main theorem.

Theorem 3.4 *For any restriction* Ψ *and vector of speeds* $\mathbf{s}$*, Algorithm* RatioStretch *with a parameter* $r \ge r^\Psi(\mathbf{s})$ *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).*

Proof: If RatioStretch schedules a job, it is always completed at time $T_j \le r \cdot C^{*,\Psi}_{\max}[(p_i)_{i=1}^n]$. 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 $p_j \le W_1(T_j)$ in the iteration when j is to be scheduled. Since $W_{m+1} \equiv 0$, this is sufficient to guarantee that the 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 $W_k(T_j) \ge p_j$, and for $t_j = T_j$ it is equal to $W_{k+1}(T_j) \le p_j$.

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. (Possibly some of these machines have speed zero, if $k - i \ge m$.) 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

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

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

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

Rearranging, this is equivalent to

$$r \ge \frac{\sum_{i=1}^{k} p_{j_i}}{\sum_{i=1}^{k} s_{k+1-i} \cdot C^{*,\Psi}_{\max}[\mathcal{J}_{[j_i]}]}. \tag{4}$$

By the assumption of the theorem we have $r \ge r^{\Psi}(\mathbf{s}, \mathcal{J})$. This implies (4), as the right-hand side of (4) is one of the terms in the supremum of Definition 3.3 for $r^{\Psi}(\mathbf{s}, \mathcal{J})$. Thus we conclude that Algorithm RatioStretch does not stop and achieves approximation ratio r.

Lemma 3.2 implies that not better approximation ratio than $r^{\Psi}(\mathbf{s})$ (resp. r^{Ψ}) can be achieved by a any semi-online algorithm. □

4 Proper restrictions and linear programs

From Definition 3.3, 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. In this section, we develop a general methodology how to compute the ratio using linear programs and in the rest of the paper we apply it to a few cases.

In the first part of the section we identify a large set of restrictions for which we can simplify the computation of the optimal ratio from Definition 3.3. We also show some general observations that may often be used to restrict the relevant inputs to monotone sequences.

The second part contains informal recipes that appear to be useful for formulating the linear programs. The tricks described here are used in several sections on specific restrictions, thus this saves repetition of the similar points later. Moreover, these methods appear to be useful in more general circumstances, and thus this part can be used as a guide when applying our methods to further restrictions. Some of the points may become more clear after studying the specific applications in the later sections.

4.1 Proper restrictions and ordering of input sequences

We observed that for a general restriction it may be necessary to use an arbitrary subsequence in Definition 3.3. 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 the restrictions 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 the restrictions 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 they are closed under permuting the input sequence.

We define a large class of restrictions, called *proper* restrictions that includes both types of restrictions discussed in the previous paragraph as well as their combinations; in particular it includes all the restrictions listed in Section 2.1 and studied in this paper. From now on we focus on proper restrictions.

Definition 4.1 *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 4.1 implies that any subsequence of any input sequence is a prefix of another input. Thus, the sets of all the subsequences and all the prefixes of Ψ coincide, and using the monotonicity condition in the definition allows us to simplify the computation of $r^\Psi(\mathbf{s})$. Compared to Definition 3.3, it is not necessary to take a supremum over all subsequences; instead we simply use the last prefixes.

Definition 4.2 *Let Ψ be a proper semi-online restriction and $\mathcal{J} \in \mathrm{pref}(\Psi)$ a partial input. We define*

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

Observation 4.3 *For any proper restriction Ψ,*

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

Proof: Fix a small $\varepsilon > 0$. By Definition 3.3 there exists $\mathcal{J} \in \mathrm{pref}(\Psi)$ such that $r^\Psi(\mathbf{s}) = r^\Psi(\mathbf{s}, \mathcal{J}) - \varepsilon$. Let $\mathcal{I}$ be a subsequence $j_1, \ldots, j_k$ of $\mathcal{J}$ from Definition 3.3 for $r^\Psi(\mathbf{s}, \mathcal{J})$ which is ε-close to the supremum. By Definition 4.1, $\mathcal{I}_{[i]} \in \mathrm{pref}(\Psi)$ for all $i \leq k$. Furthermore, by the

monotonicity condition in Definition 4.1, $C^{*,\Psi}_{\max}[\mathcal{I}_{[i]}] \leq C^{*,\Psi}_{\max}[\mathcal{J}_{[j_i]}]$ for all $i \leq k$. Summarizing and using the notation of Definition 4.2, we obtain $\bar{r}^{\Psi}(\mathbf{s}, \mathcal{I}) \geq r^{\Psi}(\mathbf{s}, \mathcal{J}) - 2\varepsilon$. Taking the supremum over all $\mathcal{I}$ obtained for $\varepsilon \to 0$, we obtain the observation. □

The following observations help us to further limit the set of relevant sequences.

Observation 4.4 *Let Ψ be an arbitrary proper restriction, let $\mathbf{s}$ be an 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 $\bar{r}(\mathbf{s}, \mathcal{J}') \geq \bar{r}(\mathbf{s}, \mathcal{J})$.

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

Observation 4.5 *Assume that (i) Ψ is closed under permutations of the sequence and (ii) increasing the processing time of the last job of a partial input cannot decrease $C^{*,\Psi}_{\max}$, i.e., for any partial inputs $\mathcal{J}, \mathcal{J}' \in \mathrm{pref}(\Psi)$ with n jobs such that $p_1 = p'_1$, …, $p_{n-1} = p'_{n-1}$, and $p_n \leq p'_n$ we have $C^{*,\Psi}_{\max}[\mathcal{J}] \leq C^{*,\Psi}_{\max}[\mathcal{J}']$. Then it is sufficient to consider sequences of non-decreasing jobs, i.e.,*

$$r^{\Psi}(\mathbf{s}) = \sup_{\mathcal{J} \in \mathrm{pref}(\Psi) \,:\, p_1 \leq p_2 \leq \cdots \leq p_n} \bar{r}^{\Psi}(\mathbf{s}, \mathcal{J})$$

Proof: First note that if Ψ is closed under permutations, then Ψ is a proper restriction. Whenever $\mathcal{J}$ contains two jobs with $p_k > p_{k+1}$, swapping them can only decrease $C^{*,\Psi}_{\max}[\mathcal{J}_{[k]}]$ and any other $C^{*,\Psi}_{\max}[\mathcal{J}_{[i]}]$ remains unchanged. Using Observation 4.4 with $b = 1$, it follows that swapping the two jobs can only increase the lower bound. Since any sequence can be sorted by swapping adjacent elements, it follows the supremum over non-decreasing inputs in Ψ is equal to the supremum over all Ψ. □

If the restriction is not closed under permutations, in some cases we can still use Observation 4.4 in a similar way as in Observation 4.5 to replace two or more jobs by identical jobs (with processing times equal to the arithmetic mean of the previous processing times). Also, using b different from 1 we can scale the instance without changing the restriction; this is used in a subtle way, for example, for the known sum of processing times. Details are given in appropriate sections.

4.2 General methods for obtaining linear programs

Before we move to the specific restrictions, let us give a few general remarks about constructing the linear programs for computing the optimal ratio.

Once we restrict the set of instances sufficiently, it is usually possible to describe instances with a given number of jobs together with the optimal makespans by a set of linear conditions. In particular, if the sequences are sorted, we know which jobs are the biggest ones and we can express bounds on the optimal makespans for prefixes easily.

To obtain a bounded number of linear programs, we observe that only last m prefixes are relevant (for a proper restriction). To compute the optimal makespans, we only need to know the total processing time and the $m-1$ largest jobs of each used prefix, which are typically again the last jobs. Thus it is usually sufficient to represent the last (and largest) $2m-1$ jobs explicitly each by one variable and to represent the remaining small jobs by a single variable giving their total processing time. This gives us one linear program for each $n < 2m$ and one additional linear program for all $n \geq m$. We can then solve all the linear programs. Often it is even sufficient to represent only the last m jobs, as we know that for the hardest instances the previous jobs do not influence the optima.

In the online case, we may represent shorter instances by having additional jobs with processing time 0, and then we may cover all instances by a single linear program, see [6]. However, in the semi-online case, generally, this does not work. The difference is in computing $C^{*,\Psi}_{\max}[\mathcal{I}]$ for $\mathcal{I}$ containing only jobs with processing time 0. In the online case, we have $C^{*}_{\max}[\mathcal{I}] = 0$, thus these artificial initial segments do not influence the bound. However, if for example the total or maximal processing time is known, then $C^{*,\Psi}_{\max}[\mathcal{I}] > 0$. In this case we need to handle sequences with a small number of jobs by separate linear programs. On the other hand, for sequences with $n < m$ jobs, the linear program is slightly simpler: In the formula (1) for computing the optimum, the maximum is always achieved in the second term for some $k = 1, \ldots, n$. Also the programs for small n stay the same for all $m \geq n$: the additional machines simply can have no influence on inputs with fewer jobs. As a further simplification, the case of $n = 1$ is necessarily trivial and yields the optimal objective equal to 1, since a partial input with a single job cannot give any non-trivial lower bound.

Finally, we actually do not optimize a linear function, but a rational function $\bar{r}^{\Psi}(\mathbf{s})$. However, in all cases that we study the semi-online restriction scales, thus we further restrict the instances by a convenient normalization, without changing the optimal ratio. Typically we normalize the denominator of the expression for the approximation ratio to 1, i.e., $1 = s_1O_n + s_2O_{n-1} + \cdots + s_nO_1$ or $1 = s_1O_m + s_2O_{m-1} + \cdots + s_mO_1$. Then the objective becomes linear.

5 Non-increasing processing times, *decr*

Among other restrictions, we are also interested in sequences of non-increasing jobs, as this is one of the most studied restrictions. Now the restriction Ψ contains sequences which have $p_j \geq p_{j+1}$ for all j. Note that Ψ is closed under taking subsequences and, in particular, $\mathrm{pref}(\Psi) = \Psi$ and $C^{*,\Psi}_{\max}[\mathcal{J}] = C^{*}_{\max}[\mathcal{J}]$ for all $\mathcal{J} \in \Psi$.

We cannot swap jobs, however, we can replace all the jobs by jobs with all processing times equal to the arithmetic mean of the original processing times. Since the jobs are non-increasing, this cannot increase the sum of processing times in any initial segment. Consequently, as the k largest jobs of any prefix are a (possibly shorter) prefix of $\mathcal{J}$, this transformation cannot increase any $C^{*,\Psi}_{\max}[\mathcal{J}_{[i]}]$. Thus, by Observation 4.4, to compute $r^{\Psi}[\mathcal{J}]$, we may restrict ourselves to instances $\mathcal{J}$ with equal processing times, i.e., $p_1 = p_2 = \cdots = p_n$. 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

$$(\hat{r}_n(\mathbf{s}))^{-1} = \frac{1}{n}\sum_{k=1}^{n} \frac{k s_{n-k+1}}{S_k} \ . \tag{5}$$

Using Observation 4.3 and the previous transformations, we obtain that $r^{\Psi}(\mathbf{s}) = \sup_n \hat{r}_n(\mathbf{s})$. We show that for any speed vector, the sequence $\hat{r}_n(\mathbf{s})$ decreases with n for $n \geq 2m$: In this case $s_{n-k+1} = 0$ for $k < n - m + 1$ and $S_k = S$ for $k > m$ and the right-hand side of (5) after reindexing $j = n - k + 1$ is

$$\frac{1}{nS}\sum_{j=1}^{m}(n - j + 1)s_i.$$

This increases with n and thus $\hat{r}_n(\mathbf{s})$ decreases. Consequently the approximation ratio for any given speeds reduces to finding a maximum of $2m$ closed formulas, and this is efficient.

5.1 The overall ratio

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 [22]. A subtle detail is that our conjecture allows one machine with a smaller speed (and from the calculations we know that this is necessary). However, with increasing number of machines, the difference from the value of the formula (5) with the last machine removed decreases to 0. Since the competitive ratio for identical machines increases with the number of machines, as shown in [22], the limits would be equal.

This is related 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? Any speed vector can be turned into an instance of the geometric sequence with $x_i = S_i$ and $y_i = i/S_i$. Then, setting $x_0 = 0$, the area is equal to

$$\sum_{i=1}(x_i - x_{i-1})y_{n-i+1} = \sum_{k=1}^{n} \frac{k s_{n-k+1}}{S_k},$$

which is inversely proportional to $\hat{r}_n(\mathbf{s})$. We conjecture that the minimum of the geometric problem 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', $\hat{r}_{an} \geq \hat{r}_n$ and $\hat{r}_{n'} \leq \frac{n+1}{n}\hat{r}_n$. For the first proof, we replace a sequence of speeds from the bound for $\hat{r}_n$ by a sequence where each speed is repeated a times, and the bound follows by manipulating the formula for $\hat{r}_n$. The second inequality is shown by replacing the speeds for $\hat{r}_{n'}$ 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. These relations show that whenever we are able to evaluate some $\hat{r}_n$ for a fixed n, the optimal overall ratio is at most $\frac{n+1}{n}\hat{r}_n$.

Lemma 5.1 *For any positive integers n and a and any speed vector $\mathbf{s}$ there exists a speed vector $\mathbf{s}'$ such that $\hat{r}_n(\mathbf{s}) \leq \hat{r}_{an}(\mathbf{s}')$. Consequently $\hat{r}_n \leq \hat{r}_{an}$.*

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 $v = 0, \ldots, a-1$. 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 $v \in \{0, \ldots, a-1\}$. 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

$$\frac{ks'_{an-k+1}}{S'_k} \leq \frac{us_{n-u+1}}{S_u}. \tag{6}$$

In the middle step of the following derivation, we use (6) for each term in the sum (obtaining a equal terms for each u). The first and the last steps follow from (5).

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

□

Lemma 5.2 *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 5.1 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, $s_u = \sum_{v=0}^{a-1} s'_{ua-v}$.

By (5) 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 $k = N - na + ua - v$ for some $u \geq 1$ and $v \in \{0, \ldots, a-1\}$.

Now we proceed towards bounding the inner sums. We have $S'_{N-na+ua-v}/(N-na+ua-v) \leq S'_{ua}/(ua)$, 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

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

□

Now we consider small values of n. For $n = 3$ the formula is

$$\hat{r}_3(s_1, s_2, s_3) = \left(\frac{s_3}{3s_1} + \frac{2s_2}{3(s_1+s_2)} + \frac{s_1}{s_1+s_2+s_3}\right)^{-1}$$

Maximizing the function $\hat{r}_3(\mathbf{s})$ can be done by hand and the maximum is $\hat{r}_3 = 1.2$ 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 $\hat{r}_n \leq \frac{5}{4}\hat{r}_4 = \frac{5}{12}(\sqrt{7}+1) \approx 1.52$.

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

Here we are given a value $\bar{P}$ and Ψ contains all $\mathcal{J}$ with $P = \bar{P}$. This restriction is not closed under taking prefixes, thus one has to carefully distinguish the possible input sequences Ψ and the larger set of possible partial inputs $\mathrm{pref}(\Psi)$. In particular, in the computation of the optimal ratio both in Definition 3.3 and Observation 4.3 one needs to consider the whole set of partial inputs $\mathrm{pref}(\Psi)$.

It can be easily verified that $\mathrm{pref}(\Psi)$ contains exactly all sequences $\mathcal{J}$ with $P \leq \bar{P}$ and $C^{*,\Psi}_{\max}[\mathcal{J}] = \max\{C^{*}_{\max}[\mathcal{J}], \bar{P}/S\}$ for any $\mathcal{J} \in \mathrm{pref}(\Psi)$. Since we can permute the jobs and increasing the size of the last job does not decrease $C^{*,\Psi}_{\max}$, Observation 4.5 implies that to compute $r^{\Psi}(\mathbf{s}) = r^{\sum p_j = P}(\mathbf{s})$, we can restrict ourselves to non-decreasing partial inputs $\mathcal{J}$.

Furthermore, we observe that we can restrict ourselves to partial inputs $\mathcal{J}$ 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

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

using $n \geq m$ and $P \leq \bar{P}$ in the last step. Thus the partial inputs with $n \geq m$ cannot lead to any non-trivial bound.

Finally, we may restrict ourselves to partial inputs $\mathcal{J}$ with $P = \bar{P}$: If $P < \bar{P}$, we scale up $\mathcal{J}$ to $\mathcal{J}'$ by multiplying all the processing times by $b = \bar{P}/P$. Observation 4.4 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.

Summarizing, we can assume that partial input $\mathcal{J}$ is a non-decreasing sequence of $n < m$ jobs with $P = \bar{P}$. This reduction seems paradoxical at first, as fewer than m jobs apparently cannot be the hardest case for m machines. However note that all the machines up to M_m actually may be used in the lower bound. Even if the sequence $\mathcal{J}$ is now in Ψ, the prefixes are not, and thus the computation of their optima $C^{*,\Psi}_{\max}[\mathcal{J}_{[i]}]$ may involve input sequences with more jobs than n.

To calculate $r(\mathbf{s})$, we solve $m-1$ mathematical programs, one for each value of n, and take the maximum of the optimal values of their objective functions. The program for given

P, $\mathbf{s}$, and n has variables q_i for job sizes and O_i for optimal makespans of the prefixes:

$$\begin{aligned}
\textbf{maximize} \quad & r = \frac{\bar{P}}{s_1 O_n + s_2 O_{n-1} + \cdots + s_n O_1} \\
\textbf{subject to} & \\
q_1 + \cdots + q_n &= \bar{P} \\
\bar{P} &\leq SO_k && \text{for } k = 1, \ldots, n \\
q_j + q_{j+1} + \cdots + q_k &\leq (s_1 + s_2 + \cdots + s_{k-j+1})O_k && \text{for } 1 \leq j \leq k \leq n \\
q_j &\leq q_{j+1} && \text{for } j = 1, \cdots, n-1 \\
0 &\leq q_1
\end{aligned}$$

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 mathematical program computes correctly the value $\bar{r}^{\Psi}(\mathbf{s}, \mathcal{J})$.

We can also see that the linear program scales and the optimum does not depend on the value $\bar{P}$. Thus we can normalize using $1 = s_1 O_n + s_2 O_{n-1} + \cdots + s_n O_1$ and also eliminate $\bar{P}$ by combining the first two constraints. The resulting linear program is:

$$\begin{aligned}
\textbf{maximize} \quad & r = q_1 + \cdots + q_n \\
\textbf{subject to} & \\
1 &= s_1 O_n + s_2 O_{n-1} + \cdots + s_n O_1 \\
q_1 + \cdots + q_n &\leq SO_k && \text{for } k = 1, \ldots, n \\
q_j + q_{j+1} + \cdots + q_k &\leq (s_1 + s_2 + \cdots + s_{k-j+1})O_k && \text{for } 1 \leq j \leq k \leq n \\
q_j &\leq q_{j+1} && \text{for } j = 1, \cdots, n-1 \\
0 &\leq q_1
\end{aligned}$$

Observe that the linear program for $n = 1$ is trivial; this is always true as a partial input with a single job cannot give any non-trivial lower bound. From the analysis above it follows that for any $\mathbf{s}$, the maximum of the $m - 2$ optima of the remaining linear programs for $n = 2, \ldots, m-1$ is the correct value of $r^{\sum p_j = P}(\mathbf{s})$.

6.1 Padding

We prove a theorem that shows that knowing the total processing time 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, see Section 6.2. The same result holds also in presence of an additional restriction with suitable properties. Among others, the requirements are satisfied for non-increasing jobs and the online case. Recall that 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 6.1 *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 $\mathcal{J}'$.

The intuitive explanation is this. First, we scale $\mathcal{J}$ to $\mathcal{J}'$ with the required total processing time. This guarantees that $\bar{r}^{\Psi,\sum p_j=P}(\mathbf{s},\mathcal{J}')$ is defined, but possibly it is much smaller than $\bar{r}^{\Psi}(\mathbf{s},\mathcal{J}')$. Next we modify $\mathbf{s}$ to $\mathbf{s}'$ by adding a huge number of slow machines. The condition that Ψ allows padding guarantees that the values of optima in $\bar{r}^{\Psi,\sum p_j=P}(\mathbf{s}',\mathcal{J}')$ can be witnessed by extensions of the partial inputs by arbitrarily small jobs. The capacity of the new machines is chosen to be sufficiently large, so that they can accommodate all these small jobs, thus guaranteeing that the additional restriction $\sum p_j = P$ has no significant influence on the value of the optima. At the same time, the new machines are chosen to be sufficiently slow so that the optimal bounds do not change significantly by adding the new machines.

Let $\mathcal{J}' = (p'_j)_{j=1}^n$ be the sequence $\mathcal{J}$ scaled by a factor of b (i.e., $p'_j = bp_j$ for all j) so that $P(\mathcal{J}') = \sum_{j=1}^n p'_j = \bar{P}$. Since Ψ allows scaling, we have $C^{*,\Psi}_{\max}[\mathcal{J}'] = b \cdot C^{*,\Psi}_{\max}[\mathcal{J}]$. To verify this, observe that scaling maps the extensions of $\mathcal{J}$ one-to-one to the extensions of $\mathcal{J}'$ and at the same time scaling does not change membership in Ψ, by the assumption that Ψ allows scaling. Consequently, we have $\bar{r}^{\Psi}(\mathbf{s},\mathcal{J}') = \bar{r}^{\Psi}(\mathbf{s},\mathcal{J})$ by Definition 4.2.

Let $O_i = C^{*,\Psi}_{\max}[\mathcal{J}'_{[i]}]$, i.e., the optimal makespan after the ith prefix of $\mathcal{J}'$. Choose a small $\sigma > 0$ so that $\sigma < s_m$ and $\sigma < \delta s_1/n$. Let $\mathbf{s}'$ be the sequence of speeds starting with $\mathbf{s}$ and continuing with $\lceil \bar{P}/(O_1\sigma) \rceil$ of values σ. The first condition on σ guarantees that $\mathbf{s}'$ is monotone and thus a valid sequence of speeds.

We claim that, for speeds $\mathbf{s}'$, we have $C^{*,\Psi,\sum p_j=P}_{\max}[\mathcal{J}'_{[i]}] \le O_i$. We extend $\mathcal{J}'_{[i]}$ by sufficiently many jobs of size at most σO_1 so that the total processing time is $\bar{P}$; this extension is in Ψ for a sufficiently small size of jobs, since Ψ allows padding. We can schedule the jobs of $\mathcal{J}'_{[i]}$ in time O_i on the original machines. Furthermore, we can schedule the remaining jobs with the total processing time at most $\bar{P}$ on the added machines of speed σ so that they complete before time O_1; this is guaranteed by the choice of the number of new machines and the upper bound on the job size. Since $O_1 \ge O_i$, the extension can be completed by time O_i and thus the extension witnesses our claim.

The claim, together with Definition 4.2 and $P(\mathcal{J}') = \bar{P}$ implies that

$$\bar{r}^{\Psi,\sum p_j=P}(\mathbf{s}',\mathcal{J}') \ge \frac{\bar{P}}{\sum_{i=1}^n s'_{n-i+1} O_i.}$$

At the same time we have

$$\bar{r}^{\Psi}(\mathbf{s},\mathcal{J}) = \bar{r}^{\Psi}(\mathbf{s},\mathcal{J}') = \frac{\bar{P}}{\sum_{i=1}^n s_{n-i+1} O_i.}$$

To complete the proof, it now suffices to compare the denominators in the previous two formulas. Using the second condition from the definition of σ, we have

$$\sum_{i=1}^n s'_{n-i+1} O_i \le \sum_{i=1}^n s_{n-i+1} O_i + n\sigma O_n \le \sum_{i=1}^n s_{n-i+1} O_i + \delta s_1 O_n \le (1+\delta) \sum_{i=1}^n s_{n-i+1} O_i.$$

Thus

$$\bar{r}^{\Psi,\sum p_j=P}(\mathbf{s}',\mathcal{J}') \ge \frac{\bar{P}}{\sum_{i=1}^n s'_{n-i+1} O_i.} \ge \frac{\bar{P}}{(1+\delta)\sum_{i=1}^n s_{n-i+1} O_i} = \frac{\bar{r}^{\Psi}(\mathbf{s},\mathcal{J})}{1+\delta}.$$

□

6.2 Two and three machines

We now examine the special cases of $m = 2, 3$. The linear program is trivial for $n = 1$. From this we can 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 processing time of all jobs is $\bar{P} \leq S \cdot T_1$.

For $m = 3$, it remains to solve the linear program for $n = 2$. In the next theorem we solve it explicitly to illustrate our techniques for obtaining closed formulas for a fixed number of machines.

Theorem 6.2 *The optimal approximation ratio for semi-online scheduling with known sum of the processing times on three machines is:*

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

Proof: 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:

$$\begin{array}{rcll} \textbf{maximize} & r = q_1 + q_2 & & \\ \textbf{subject to} & & & \\ 1 & = & s_1 O_2 + s_2 O_1 & (z_{norm}) \\ q_1 + q_2 & \leq & (s_1+s_2+s_3) O_1 & (z_1) \\ q_1 + q_2 & \leq & (s_1+s_2+s_3) O_2 & (z_2) \\ q_1 & \leq & s_1 O_1 & (z_{1,1}) \\ q_1 + q_2 & \leq & (s_1+s_2) O_2 & (z_{1,2}) \\ q_2 & \leq & s_1 O_2 & (z_{2,2}) \\ q_1 & \leq & q_2 & (z_{\leq}) \\ 0 & \leq & q_1 & (z_0) \end{array} \tag{7}$$

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. By the slackness conditions for linear programming, we only need to use the inequalities that are tight in the primal optimal solution.

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

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 $(z_{\leq})$ 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 $(s_1+s_2)s_1/D \leq (s_1+s_2+s_3)s_2/D$. If we multiply both sides by D and subtract $s_1 s_2$, we can see that this is equivalent to case condition. Finally, we check that the objective value is $s_1 s_2/D + s_1^2/D = s_1(s_1+s_2)/(s_1^2+s_2^2)$ which is equal to the claimed bound.

To demonstrate the dual solution, we add up inequalities in (7) 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

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

Equivalently, we get $r = q_1+q_2 \leq s_1(s_1+s_2)/(s_1^2+s_2^2)$. 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: $s_1^2 \geq s_2(s_2+s_3)$.

Let $D = s_1(s_1+s_2+s_3) + s_2(s_1+s_2) = (s_1+s_2)^2 + s_1 s_3$.

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 $(z_{\leq})$ 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 $s_2(s_1+s_2+s_3)/D \leq s_1(s_1+s_2)/D$, 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 (7) 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 $(q_1+q_2)D - (s_1+s_2)(s_1+s_2+s_3) \leq 0$. This gives the upper bound $r = q_1+q_2 \leq (s_1+s_2)(s_1+s_2+s_3)/D$ which is equal to the claimed bound. □

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.

7 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 Ψ containing all the sequences where the first job is a maximal one, i.e., $p_1 \geq p_i$ for all $i \leq n$. Then $\mathrm{pref}(\Psi) = \Psi$ and $C^{*,\Psi}_{\max}[\mathcal{J}] = C^*_{\max}[\mathcal{J}]$ for any $\mathcal{J} \in \Psi$.

Using the same argument as in the proof of Observation 4.5, by Observation 4.4, the other jobs can be reordered as in the previous case, and we can maximize only over sequences with

non-decreasing processing times from the second job on. Furthermore, we can replace the first and last jobs (which are now the two largest jobs) by two jobs with processing time $(p_1 + p_n)/2$. By Observation 4.4, the value $\bar{r}^{\Psi}(\mathbf{s}, \mathcal{J})$ cannot decrease.

We proceed towards the formulation of the linear programs. In this case we need to solve separately $m-1$ mathematical programs, one for each $n = 2, \ldots, m-1$ and one for $n \geq m$, and take the maximum value. In all the programs we can use scaling and normalization, to obtain a linear program, using a variable p for p_1 instead of a constant $\bar{p}$.

The resulting linear programs are the following. For each $2 \leq n < m$, we have the following linear program:

$$
\begin{array}{rcll}
\textbf{maximize} & r = p + q_2 + \cdots + q_n & & \\
\textbf{subject to} & & & \\
1 & = & s_1 O_n + \cdots + s_n O_1 & (z_{norm}) \\
p & \leq & s_1 O_k & (z_{k,k}) \quad 1 \leq k \leq n \\
p + q_{j+1} + \cdots + q_k & \leq & (s_1 + \cdots + s_{k-j+1}) O_k & (z_{j,k}) \quad 1 \leq j < k \leq n \\
0 & \leq & q_2 & (z_{0,2}) \\
q_k & \leq & q_{k+1} & (z_{\leq,k}) \quad 2 \leq k \leq n-1 \\
q_n & = & p & \text{(substitution)}
\end{array}
$$

The constraint $(z_{n,n})$ is implied by $(z_{n-1,n})$, thus we can omit it.

For $n \geq m$, we have a single linear program. Here the variables are p for the processing time of the first job, q_1 for the total processing time of the small jobs, i.e., $q_1 = p_2 + \cdots + p_{n-m+1}$, and $q_i = p_{n-m+i}$, for $i = 2, \ldots, m$.

$$
\begin{array}{rcll}
\textbf{maximize} & r = p + q_1 + \cdots + q_m & & \\
\textbf{subject to} & & & \\
1 & = & s_1 O_m + \cdots + s_m O_1 & (z_{norm}) \\
p + q_1 + \cdots + q_k & \leq & S O_k & (z_k) \quad 1 \leq k \leq m \\
p & \leq & s_1 O_k & (z_{k,k}) \quad 1 \leq k \leq m \\
p + q_{j+1} + \cdots + q_k & \leq & (s_1 + \cdots + s_{k-j+1}) O_k & (z_{j,k}) \quad 1 \leq j < k \leq m \\
0 & \leq & q_1 & (z_{0,1}) \\
0 & \leq & q_2 & (z_{0,2}) \\
q_k & \leq & q_{k+1} & (z_{\leq,k}) \quad 1 \leq k \leq m-1 \\
q_m & = & p & \text{(substitution)}
\end{array}
$$

The constraint $(z_{1,m})$ is implied by (z_m) and $(z_{m,m})$ is implied by $(z_{m-1,m})$, thus these two constraints can be omitted.

To verify that the maximum of the m linear programs is $r^{\Psi}(\mathbf{s})$, we observe that from each input $\mathcal{J}$ we can construct a feasible solution of one of the linear programs with objective at least $\bar{r}^{\Psi}(\mathbf{s}, \mathcal{J})$ and vice versa. The first direction, from the instance to a feasible solution, is described during the construction of the linear programs.

Given a feasible solution of a linear program for $n < m$, the corresponding input is simply the sequence of n jobs with $p_1 = p$ and $p_i = q_i$ for $i = 2, \cdots, n$. Given a feasible solution of a linear program for $n \geq m$, we need to replace q_1 by a prefix of jobs with total processing time q_1. We choose $k = \max\{m, \lceil q_1/p \rceil\}$. The input instance has $n = m + k$ jobs and the processing times are $p_1 = p$, $p_2 = \cdots = p_{k+1} = q_1/k$, and $p_{k+i} = q_i$ for $i = 2, \ldots, m$. The choice of k guarantees that p is the maximal processing time. The fact that we replace q_1 by at least m jobs of equal size guarantees that the maximum in (1) is given either by P/S or

by P_i/S_i for some i such that P_i does not contain any of these jobs replacing q_1, using the observation after (1). Thus $C^{*,\Psi}_{\max}[\mathcal{J}_{[n-i+1]}] \leq O_i$ for all $i = 1, \ldots, m$, and the lower bound given by $\mathcal{J}$ is at least the objective of the linear program.

For this restriction, the best lower bound is given by a computer-generated hard instance with approximation ratio 1.88 with $m = 120$.[1] This instance is obtained by treating the above mathematical program as a quadratic program with variables p, q_j, O_i, and s_i. After some heuristic substitutions, the solver finds a reasonably good solution, which can be verified easily.

7.1 Three machines

Next we give a complete solution for $m = 3$, generalizing the analysis for $m = 2$ in [16].

Theorem 7.1 *The optimal approximation ratio for semi-online scheduling with known maximal processing time on three machines is*

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

Proof: We need to solve one linear program for $n = 2$ and one linear program for $m = 3$ (covering all the cases $n \geq 3$), and take the maximum.

The linear program for $n = 2$ is simple, as the only considered sequence has two jobs of size p and the bound can be written explicitly as

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

It turns out that optimal solution of linear program for $m = 3$, which we solve below, is always larger than the previous formula, we omit this calculation. The program for $m = 3$ is:

$$\begin{array}{rcll} \textbf{maximize} & r = 2p + q_1 + q_2 & & \\ \textbf{subject to} & & & \\ 1 & = & s_1 O_3 + s_2 O_2 + s_3 O_1 & (z_{norm}) \\ p + q_1 & \leq & (s_1 + s_2 + s_3) O_1 & (z_1) \\ p + q_1 + q_2 & \leq & (s_1 + s_2 + s_3) O_2 & (z_2) \\ 2p + q_1 + q_2 & \leq & (s_1 + s_2 + s_3) O_3 & (z_3) \\ p & \leq & s_1 O_1 & (z_{1,1}) \\ p & \leq & s_1 O_2 & (z_{2,2}) \\ p + q_2 & \leq & (s_1 + s_2) O_2 & (z_{1,2}) \\ 2p & \leq & (s_1 + s_2) O_3 & (z_{2,3}) \\ 0 & \leq & q_1 & (z_{0,1}) \\ 0 & \leq & q_2 & (z_{0,2}) \\ q_2 & \leq & p & (z_{\leq,2}) \end{array}$$

Case I $s_1 s_2 \geq (S - s_1 - s_2) S$**.**

This case holds for any $m \geq 2$ and we prove it here in this general form. Let $D = S^2 + s_1^2$.

[1] See the Maple output at `http://kam.mff.cuni.cz/~sgall/ps/semirel-pmax.mpl`

The primary optimal solution is $q_1 = (S - s_1)S/D, q_2 = \ldots = q_{m-1} = 0, p = s_1 S/D, O_1 = \ldots = O_{m-1} = S/D, O_m = (S + s_1)/D$. All inequalities can be easily verified. This gives a lower bound on the ratio of $(S^2 + s_1 S)/(S^2 + s_1^2)$.

To obtain an upper bound (and demonstrate the optimal dual solution) we sum the inequalities multiplied by the following non-negative coefficients: $z_{i,i} = s_{m-i+1}S(S + s_1)$ for $i = 1, \cdots, m-2$, $z_{m-1,m-1} = (s_1 s_2 - (S - s_1 - s_2)S)S$, $z_{m-1} = s_1(S - s_1)S$, and $z_m = s_1^2(S + s_1)$. The resulting inequality yields the bound after substituting the identity (z_{norm}).

Case II $s_1 s_2 \leq s_3 S$**.**

Let $D = S^2 + 2s_1^2 + s_1 s_2$.

The primary optimal solution is: $q_1 = (S - s_1)S/D$, $p = q_2 = s_1 S/D$, $O_1 = S/D$, $O_2 = (s_1 + S)/D$, $O_3 = (2s_1 + S)/D$. Again, all the inequalities can be easily checked as well as the resulting value of the lower bound.

To obtain an upper bound we sum the inequalities multiplied by the following non-negative coefficients: $z_{\leq,2} = z_1 = s_3 S - s_1 s_2$, $z_2 = s_2(2s_1 + S)$, $z_3 = s_1(2s_1 + S)$, $z_{1,1} = (s_2 + 2s_3)S$.

The linear programs for $n = 2$ is trivial. The only remaining input sequence has two identical jobs. The resulting ratio is $r = 1 + \frac{s_1 s_2 - s_2^2}{2s_1^2 + s_1 s_2 + s_2^2}$. We verify that this ratio is strictly smaller than the resulting formula of program for $m = 3$:

We prove $\frac{s_1 s_2 - s_2^2}{2s_1^2 + s_1 s_2 + s_2^2} \leq \frac{s_1(s_2 + s_3)}{S^2 + s_1^2}$ by:

$$\begin{aligned}
(s_1 s_2 - s_2^2)(S^2 + s_1^2) &= 2s_1^3 s_2 + 2s_1^2 s_2 s_3 - s_1 s_2^3 + s_1 s_2 s_3^2 - s_2^4 - 2s_2^3 s_3 - s_2^2 s_3^2 \\
&\leq 2s_1^3 s_2 + 2s_1^2 s_2 s_3 + s_1 s_2 s_3^2 \\
&= s_1 s_2(2s_1^2 + s_1 s_3 + s_3^2) + s_1^2 s_2 s_3 \\
&\leq s_1(s_2 + s_3)(2s_1^2 + s_1 s_2 + s_2^2)
\end{aligned}$$

The second case $\frac{s_1 s_2 - s_2^2}{2s_1^2 + s_1 s_2 + s_2^2} \leq \frac{s_1 s_2 + 2s_1 s_3}{S^2 + 2s_1^2 + s_1 s_2}$ is proved by:

$$\begin{aligned}
(s_1 s_2 - s_2^2)(S^2 + 2s_1^2 + s_1 s_2) &= 3s_1^3 s_2 + 2s_1^2 s_2 s_3 - 2s_1 s_2^3 + s_1 s_2 s_3^2 - s_2^4 - 2s_2^3 s_3 - s_2^2 s_3^2 \\
&\leq 3s_1^3 s_2 + 2s_1^2 s_2 s_3 + s_1 s_2 s_3^2 \\
&= s_1 s_2(2s_1^2 + s_1 s_3 + s_3^2) + s_1^3 s_2 + s_1^2 s_2 s_3 \\
&\leq (s_1 s_2 + 2s_1 s_3)(2s_1^2 + s_1 s_2 + s_2^2)
\end{aligned}$$

□

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

8 Approximately known optimal makespan, $T \leq C^*_{\max} \leq \alpha T$.

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.

In this particular variant, we have two parameters, $\bar{T}$ and $\alpha > 1$. The semi-online restriction contains all job sequences with the optimal makespan between $\bar{T}$ and $\alpha\bar{T}$. We proceed

towards the formulation of the linear programs. Here we need to solve separately m mathematical programs, one for each $n = 1, \ldots, m-1$ and one for $n \geq m$, and take the maximum value.

We now describe the usual simplifications; they apply to all the m mathematical programs. Since we can permute the jobs and increasing the size of the last job does not decrease $C^{*,\Psi}_{\max}$, Observation 4.5 implies that we can restrict ourselves to non-decreasing sequences $\mathcal{J}$.

To express the constraint bounding the optimum, one can simply assert that $\bar{T} \leq O_k \leq \alpha\bar{T}$. We can see that the mathematical program scales and its value does not depend on the value of $\bar{T}$. Thus we can treat $\bar{T}$ as a variable and normalize the denominator of the objective function to be 1. To further simplify the linear program, we note that in the optimal solution we have $O_1 \leq O_2 \leq \cdots$, i.e., the values of the variables for the optima are ordered. Then the constraint for bounding the optima in terms of $\bar{T}$ simplifies to $O_n \leq \alpha O_1$ or $O_m \leq \alpha O_1$ and the variable $\bar{T}$ no longer appears in the program.

The resulting linear programs are the following. For each $n < m$, we have the linear program

$$
\begin{array}{rcll}
\textbf{maximize} & r = q_1 + q_2 + \cdots q_n & & \\
\textbf{subject to} & & & \\
1 & = & s_1 O_n + s_2 O_{n-1} + \cdots + s_n O_1 & \\
q_j + q_{j+1} + \cdots + q_k & \leq & (s_1 + s_2 + \cdots + s_{k-j+1}) O_k & \text{for } 1 \leq j \leq k \leq n \\
q_j & \leq & q_{j+1} & \text{for } j = 1, \ldots, n-1 \\
0 & \leq & q_1 & \\
O_k & \leq & O_{k+1} & \text{for } k = 1, \ldots, n-1 \\
O_n & \leq & \alpha O_1 &
\end{array}
$$

For $n \geq m$ we have a single linear program, where the variable q_1 denotes the sum of the first processing times, i.e., $q_1 = p_1 + p_2 + \cdots + p_{n-m+1}$.

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

To verify that the maximum of the m linear programs is $r^{\Psi}(\mathbf{s})$, we observe that from each input $\mathcal{J}$ we can construct a feasible solution of one of the linear programs with objective at least $\bar{r}^{\Psi}(\mathbf{s}, \mathcal{J})$ and vice versa. The first direction, from the instance to a feasible solution, is again described during the construction of the linear programs.

Given a feasible solution of a linear program for $n < m$, the corresponding input is simply the sequence of n jobs with $p_i = q_i$. Given a feasible solution of a linear program for $n \geq m$, the corresponding input has $n = 2m - 1$ jobs and the processing times are $p_1 = \cdots = p_m = q_1/m$ and $p_{m+i} = q_{i+1}$ for $i = 1, \ldots, m-1$. In both cases we set $\bar{T} = O_1$. Again, as for the previous restriction, the optima O_i are computed correctly and we obtain a desired lower bound on the approximation ratio.

8.1 Three machines

Next we give a complete analysis for $m = 3$, generalizing the analysis for $m = 2$ in [20].

Theorem 8.1 *The optimal approximation ratio for semi-online scheduling with approximately known optimum up to a factor α on three machines is*

$$r^{T\le C^*_{\max}\le \alpha T}(s_1,s_2,s_3) = \begin{cases} \dfrac{\alpha S}{(\alpha-1)s_1+S} & \text{for } S \ge \alpha(S-s_1) \\ \dfrac{\alpha S^2}{\alpha s_1 S + \alpha s_2(S-s_1)+s_3 S} & \text{for } \begin{cases} S \le \alpha(S-s_1), \\ S \ge \alpha s_3, \text{ and} \\ S^2 \ge \alpha(S-s_1)^2 \end{cases} \\ \dfrac{S^2}{S^2 - s_1s_2 - s_1s_3 - s_2s_3} & \text{for } \begin{cases} S \le \alpha s_3 \text{ and} \\ s_2 S \le s_1(S-s_1) \end{cases} \\ \dfrac{S^3}{s_1S^2 + s_2(S-s_1)S + s_3(S-s_1)^2} & \text{for } \begin{cases} S^2 \le \alpha(S-s_1)^2 \text{ and} \\ s_2 S \ge s_1(S-s_1) \end{cases} \end{cases}$$

Proof: We need to solve the linear programs for $n = 2$ and $m = 3$, and take the maximum. It turns out that optimal solution of linear program for $m = 3$ is always the maximal of these two.

Here we list the linear program for $m = 3$ explicitly, followed by its optimal solutions:

$$\begin{array}{rcll} \textbf{maximize} & r = q_1 + q_2 + q_3 & & \\ \textbf{subject to} & & & \\ 1 & = & s_1O_3 + s_2O_2 + s_3O_1 & (z_{norm}) \\ q_1 & \le & (s_1+s_2+s_3)O_1 & (z_1) \\ q_1+q_2 & \le & (s_1+s_2+s_3)O_2 & (z_2) \\ q_1+q_2+q_3 & \le & (s_1+s_2+s_3)O_3 & (z_3) \\ q_2 & \le & s_1O_2 & (z_{2,2}) \\ q_2+q_3 & \le & (s_1+s_2)O_3 & (z_{2,3}) \\ q_3 & \le & s_1O_3 & (z_{3,3}) \\ O_3 & \le & \alpha O_1 & (z_\alpha) \\ O_1 & \le & O_2 & (z_{b,1}) \\ O_2 & \le & O_3 & (z_{b,2}) \\ 0 & \le & q_1 & (z_{0,1}) \\ 0 & \le & q_2 & (z_{0,2}) \\ q_2 & \le & q_3 & (z_{\le,2}) \end{array}$$

Case I: $S \ge \alpha(S - s_1)$.

This case holds for general values of m, and we prove it here in this general form. Let $D = (\alpha-1)s_1 + S$.

The primary optimal solution is $q_1 = S/D$, $q_2 = \cdots = q_{m-1} = 0$, $q_m = (\alpha-1)S/D$, $O_1 = \ldots = O_{m-1} = 1/D$, $O_m = \alpha/D$. It is easy to verify that this is feasible; note that $(z_{m,m})$ is equivalent to the case condition. The objective is the desired lower bound of $\alpha S/((\alpha-1)s_1 + S)$.

To prove the upper bound we first derive inequalities $O_1 \le O_k$ from $(z_{b,i})$. Now sum the corresponding inequalities multiplied by the following coefficients: $z_\alpha = (S-s_1)S/D$, $z_m = 1$, and inequalities $O_1 \le O_k$ multiplied by $\alpha s_k S/D$ for $2 \le k \le m-1$.

Case II: $S \le \alpha(S - s_1)$, $S^2 \ge \alpha(S - s_1)^2$, and $S \ge \alpha s_3$.

Let $D = \alpha s_1 S + \alpha s_2(S - s_1) + s_3 S$.

The primary optimal solution is $q_1 = S^2/D$, $q_2 = (\alpha(S - s_1) - S)S/D$, $q_3 = \alpha s_1 S/D$, $O_1 = S/D$, $O_2 = \alpha(S - s_1)/D$, $O_3 = \alpha S/D$. Due to the first case condition, $q_2 \ge 0$. We get the ratio $r = \alpha S^2/D$.

For the upper bound we sum the inequalities with following coefficients: $z_\alpha = s_3 S^2$, $z_2 = z_{3,3} = \alpha s_2 S$, $z_3 = s_3 S + \alpha s_1(s_1 + s_3)$.

Cases III and IV. The ratios in these cases are equal to the ratios for online scheduling. Thus the upper bound is trivial. To verify the lower bound, we use the same sequences as for online scheduling. In addition we need to verify that (z_α) is satisfied for these cases: it turns out to be equivalent to the case conditions $S \le \alpha s_3$ for Case III and $S^2 \le \alpha(S - s_1)^2$ for Case IV. We omit the proofs and sequences here as they can be found in [6].

The linear program for $n = 2$ has maximum

$$r = \min\left\{ \frac{s_1^2 + s_1 s_2}{s_1^2 + s_2^2}, \frac{\alpha s_1 + \alpha s_2}{\alpha s_1 + s_2} \right\}.$$

We omit the proof here. It can be easily verified that this formula is dominated by the formula in Case I. For Case II the verification is a bit harder, but it is possible to substitute the case condition $S \le \alpha(S - s_1)$ into the value of the approximation ratio to obtain a formula without α, and the rest is a mechanical calculation. In Cases III and IV the approximation ratio is the same as for the online scheduling, thus the linear program for $n = 2$ cannot give a larger bound and no verification is needed. □

9 Tightly grouped processing times, $p \le p_j \le \alpha p$

In this case we have two parameters $\bar{p}$ and α, and the restriction Ψ contains all sequences $\mathcal{J}$ with $\bar{p} \le p_j \le \alpha\bar{p}$, for all $j = 1, \ldots, n$. Once again, Ψ is closed under taking subsequences and, in particular, $\mathrm{pref}(\Psi) = \Psi$ and $C^{*,\Psi}_{\max}[\mathcal{J}] = C^*_{\max}[\mathcal{J}]$ for all $\mathcal{J} \in \Psi$.

The case of tightly grouped jobs is interesting, because it is not clear how to set up a single linear program for a sufficiently large n. The problem is that the possible total processing time of small jobs do not form a single interval, and thus the domain of the mathematical program is not convex. (Even for a fixed α it is not straightforward to form a single linear program for $n \ge n_0$ for any n_0.) However, we observe that for a fixed α and $\mathbf{s}$, starting from some n_0, the value of $\bar{r}^\Psi(\mathbf{s}, \mathcal{J})$ decreases with the number of jobs. Thus a constant number of linear programs is sufficient.

To form the linear programs for a fixed n is now routine, even though in this case we possibly need also values $n > m$ (in which case we use the notation $s_i = 0$ for $i > m$). Since Ψ is closed under permutations, we may restrict ourselves to sequences with non-decreasing processing times. The restriction on the processing times is now handled similarly as in the previous case the restriction of approximately known optimum. Also, we can normalize as

usual. The resulting linear program for a fixed n is:

$$\begin{array}{rcll} \textbf{maximize} & r = q_1 + q_2 + \cdots q_n & & \\ \textbf{subject to} & & & \\ 1 & = & s_1 O_n + s_2 O_{n-1} + \cdots + s_n O_1 & \\ q_1 + q_2 + \cdots + q_k & \leq & S O_k & \text{for } 1 \leq k \leq n \\ q_j + q_{j+1} + \cdots + q_k & \leq & (s_1 + s_2 + \cdots + s_{k-j+1}) O_k & \text{for } 1 \leq j \leq k \leq n \\ q_j & \leq & q_{j+1} & \text{for } j = 1, \cdots, n-1 \\ q_n & \leq & \alpha q_1 & \end{array}$$

Now we want to bound the number of the linear programs needed for a fixed α and $\mathbf{s}$. We observe that for a sufficiently large n, we have $C^{*,\Psi}_{\max}[\mathcal{J}] = P/S$. Indeed, to guarantee this for all sequences of length n, it is sufficient to guarantee that P/S is at least the bound in the second term of (1) for all k. For a given k, the worst possible sequence has exactly k jobs with processing times $\alpha\bar{p}$ and the remaining jobs with processing times $\bar{p}$. Then the required inequality is

$$\frac{n + k(\alpha - 1)}{S} \geq \frac{k\alpha}{S_k} \qquad \text{for all } k = 1, \ldots, m. \tag{8}$$

Clearly, for some n_1 (dependent on α and $\mathbf{s}$), (8) holds for any $n \geq n_1$.

Take $n_0 = n_1 + m - 1$; note that $n_0 \geq m$. Now, for any $\mathcal{J}$ with $n \geq n_0$ jobs and any $i \geq n - m + 1$, we have $C^{*,\Psi}_{\max}[\mathcal{J}_{[i]}] = (p_1 + p_2 + \cdots + p_i)/S$. Consequently, expanding the terms, we have

$$\bar{r}^{\Psi}[\mathbf{s}, \mathcal{J}] = \frac{(s_1 + \cdots + s_m)(p_1 + \cdots + p_n)}{s_1(p_1 + \cdots + p_n) + s_2(p_1 + \cdots + p_{n-1}) + \cdots + s_m(p_1 + \cdots + p_{n-m+1})}. \tag{9}$$

Consider this as a linear rational function in p_i's. The coefficient of p_1 is equal to S both in the numerator and in the denominator, while the whole ratio is at least 1. Thus, assuming $n > n_0$, by removing all the occurrences of p_1, the value of the fraction does not decrease. However, the obtained expression is equal to the one for $\bar{r}^{\Psi}[\mathbf{s}, \mathcal{J}']$, where $\mathcal{J}'$ is the sequence with the first job removed (which still has at least n_0 jobs). This shows that in computing $r^{\Psi}[\mathbf{s}]$, we can restrict ourselves to sequences $\mathcal{J}$ with $n \leq n_0$ jobs.

For $n = n_0$, we can further analyze the formula (9). Since the ratio is a linear rational function, it is monotone in each p_i (it can be decreasing or increasing, depending on the speeds, always decreasing in p_1). Thus the hardest sequence of n_0 jobs has $p_1 = 1$ and $p_i \in \{1, \alpha\}$ for $i = 2, \ldots, n_0$ (after scaling), and we can simply calculate the ratio for these n_0 sequences.

Thus we can compute $r^{\Psi}[\mathbf{s}]$ by taking the maximum among these n_0 sequences and the optima of the linear programs for $n < n_0$. Note that we have not tried to find a good bound for n_0 or for the number of the conditions in the linear programs. For special cases, e.g., for a small number of machines, it is possible to give better bounds.

9.1 Two machines

We now examine the case of $m = 2$, to obtain the results of [16] using our framework. Denote $s = s_1/s_2$; by scaling we can assume that $s_1 = s$ and $s_2 = 1$.

From the previous results for the online problem [25, 10], we know that in the online case, the hardest sequence has three jobs with $p_1 = p_2 = 1$ and $p_3 = 2s$. This sequence is in Ψ

whenever $\alpha \geq 2s$. Thus for $\alpha \geq 2s$, the optimal algorithm for the current restriction is the optimal online algorithm.

Now consider the case $\alpha < 2s$. The condition (8) reduces to a single condition $(n + (\alpha - 1))/(s + 1) \geq \alpha/s$, or equivalently $n \geq 1 + \alpha/s$. Since $\alpha/s < 2$, we can take $n_1 = 3$ and $n_0 = 4$. For $n = n_0 = 4$, the ratio (9) is decreasing in p_1, p_2, and p_3, thus maximized by a sequence $(1, 1, 1, r)$, i.e., with $p_1 = p_2 = p_3 = 1$ and $p_4 = \alpha$.

To completely analyze the case of $m = 2$, it is sufficient to analyze the sequences with $n = 2$ and $n = 3$, and take the maximum of these and the sequence $(1, 1, 1, r)$. For analyzing the sequences with $n = 2$ and $n = 3$ we can directly use the formula for $\bar{r}^{\Psi}[\mathbf{s}, \mathcal{J}]$ instead of solving the linear program. We can take advantage of scaling so that $p_1 = 1$ and also of the fact that for $n = 3$ the optimum is P/S, as we proved above. For $n = 3$ it is then easy to see that the bound is increasing in p_3, thus we have $p_3 = \alpha$ in the worst case. In both cases $n = 2, 3$ it remains to find the value of p_2. A routine calculus calculation which we omit shows that the only possible extremal sequences are $(1, \min\{r, s\})$ and $(1, 1, r)$.

Finally, it turns out that for $r < 2s$, the optimum for the sequence $(1, 1, r)$ is P/S, as well as for its prefix $(1, 1)$ thus the sequence $(1, 1, r)$ always gives a larger bound than $(1, 1, 1, r)$, using our previous analysis of (9).

Thus the only possible extremal sequences are $(1, \min\{r, s\})$ and $(1, 1, r)$, yielding the result of [16].

10 Combined restrictions

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. Sometimes we need to be somewhat careful as some simplifications connected to the two restrictions are not compatible. We give two examples, in each we know the total processing time and in the first one also the maximal processing time, while in the second the jobs are non-increasing. In both cases Theorem 6.1 allows padding and thus the overall approximation ratio is the same as without knowledge of the total processing time.

10.1 Known total and maximal processing times, $\sum p_j = P, p_{\max} = p$

In this section we construct the linear program for the first combined semi-online restriction. We are given two parameters, $\bar{P}$ and $\bar{p}$. The restriction Ψ contains all inputs $\mathcal{J}$ with $P = \bar{P}$, $p_1 = \bar{p}$, and $p_j \leq \bar{p}$ for all $j \leq n$. The interesting feature of this variant is that not all the simplifications used before are possible, as they could change the ratio of P and the maximal processing time. Consequently, the results for small m split into more cases, depending on this ratio.

Similarly to the case $p_{\max} = p$, the restriction of knowing the maximal processing time is equivalent to the restriction to the set of inputs where the first job is the maximal one. This modified restriction allows padding and scaling, and thus the overall approximation ratio is the same as for the restriction $p_{\max} = p$ by Theorem 6.1.

Furthermore, similarly to the case of $p_{\max} = p$, we may restrict ourselves to partial inputs with non-decreasing processing times from the second job on. (However, the jobs in the extensions that define the optima may be small.) Using the same calculation as in the case of $\sum p_j = P$, it follows that we may restrict ourselves to partial inputs with $n < m$.

On the other hand, unlike in the previous cases, we cannot scale so that $P = \bar{P}$ and we cannot assume that the last job has the maximal processing time. These arguments break down, as the appropriate transformation changes the ratio of $\bar{P}$ and $\bar{p}$.

Let $\beta = \bar{P}/\bar{p}$. Note that $\beta \geq 1$, as otherwise Ψ is empty. Here we have two cases where the competitive ratio is 1, and both of them can be explained intuitively. One case is when β is so small that when the largest job is processed on the fastest machine, then all the other jobs can be completed in the same time on the second machine. The other case is the opposite extreme: β is so large so that we know that the optimum is always given by the total volume of the jobs, not the large jobs. It is also not very surprising that solutions are very different for the case of $\beta \leq 2$ when the maximal job is larger than all the remaining jobs and for the case of $\beta \geq 2$. Surprisingly, for β close to 2 there is always one subcase where the exact competitive ratio does not depend on β; we do not have an intuitive explanation for this phenomenon.

The mathematical program for a fixed $n < m$ is:

$$\begin{array}{rcll}
\textbf{maximize} & r = p + q_2 + \cdots + q_n & & \\
\textbf{subject to} & & & \\
1 & = & s_1 O_n + \cdots + s_1 O_n & \\
\beta p & \leq & S O_k & \text{for } 1 \leq k \leq n \\
p & \leq & s_1 O_k & \text{for } 1 \leq k \leq n \\
p + q_{j+1} + \cdots + q_k & \leq & (s_1 + \cdots + s_{k-j+1}) O_k & \text{for } 1 \leq j < k \leq n \\
p + q_2 + \cdots + q_n & \leq & \beta p & \\
q_n & \leq & p & \\
0 & \leq & q_2 & \\
q_{k-1} & \leq & q_k & \text{for } 3 \leq k \leq n
\end{array}$$

Theorem 10.1 *The optimal approximation ratio for semi-online scheduling with known total and maximal processing times on three machines for $\beta \leq 2$ is following:*

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

and for $\beta \geq 2$ is following:

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

We omit the proof of this theorem, see [5] for the solutions of the linear programs.

10.2 Known total processing time and non-increasing jobs, $\sum p_j = P, decr$

We conclude by a restriction Ψ containing all sequences with total processing time equal to a parameter $\bar{P}$ and non-increasing processing times. Similarly as for non-increasing processing times, we do not need to solve linear programs.

Again, similarly to the non-increasing jobs, we can assume that all the jobs in the partial input $\mathcal{J}$ are equal; we normalize their size to be 1. Once all jobs are equal we have

$$C^{*,\Psi}_{\max}[\mathcal{J}] = \max\left\{\frac{n}{S_n}, \frac{\bar{P}}{S}\right\}. \tag{10}$$

Note that again the jobs in the extensions defining $C^{*,\Psi}_{\max}[\mathcal{J}]$ are possibly smaller than 1.

It must hold that $\bar{P} \geq n$, as otherwise the sequence cannot be completed to satisfy $\sum p = \bar{P}$. If $n \geq m$, then all optima are equal, yielding approximation ratio of 1. I.e., if the algorithm gets first job smaller or equal P/m, it can always schedule all jobs optimally. So we can restrict ourselves only to cases with $n < m$.

Also, we can scale the whole sequence by $b^{-1} = \bar{P}/n$, i.e., the scaled sequence has $p_j = \bar{P}/n$. From Observation 4.4 we get that the scaled sequence yields larger lower bound. Thus we can restrict ourselves to instances with $\bar{P} = n$.

Let us denote $\hat{r}_n(\mathbf{s}) = \bar{r}^{\sum p_j = P, decr}(\mathbf{s}, \mathcal{J})$ for a sequence $\mathcal{J}$ with n jobs with $p_j = 1$ and for $\bar{P} = n$. Using Observation 4.3 and the previous transformations, we obtain that $r^{\sum p_j = P, decr}(\mathbf{s}) = \max_{n=1}^{m-1} \hat{r}_n(\mathbf{s})$.

Using (10) we obtain

$$(\hat{r}_n(\mathbf{s}))^{-1} = \sum_{k=1}^{n} s_{n-k+1} \cdot \max\left\{\frac{k}{nS_k}, \frac{1}{S}\right\}.$$

Note that because k/S_k is nondecreasing with k, the maximum for the first few values of k is $1/S$ and for the remaining ones it is $k/(nS_k)$.

Now we examine the cases for two, three, and four machines. The restriction $\sum p = P$ ensures that in case $m = 2$ the algorithm achieves approximation ratio 1.

For three machines, we only need to calculate $\hat{r}_2$. We have

$$(\hat{r}_2(\mathbf{s}))^{-1} = s_2 \cdot \max\left\{\frac{1}{2s_1}, \frac{1}{s_1 + s_2 + s_3}\right\} + \frac{s_1}{s_1 + s_2}.$$

The approximation ratio is maximized at $s_1 = 2, s_2 = s_3 = 1$, and its value is $r = \frac{12}{11} \approx 1.091$.

For four machines we take the maximum of $\hat{r}_2$ and $\hat{r}_3$. The first one is maximized when $s_2 = s_3 = s_4$ and has the approximation ratio is $(7\sqrt{2} + 2)/7 \approx 1.094$, which is the same as without the restriction $\sum p_j = P$. The formula for $\hat{r}_3$ is maximized at $s_1 = s_2 = 1$, $s_3 = s_4 = \frac{1}{2}$, where the ratio is $r = 10/9 \approx 1.111$.

Conclusions and open problems.

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 a variety of special cases.

10.3 Other restrictions

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.

We should note that there are also semi-online models that do not fit into our framework at all. For example, the algorithm may be allowed to store some job(s) in a buffer. This model was introduced for the non-preemptive version in [21], later tight bounds on identical machines were given in [8]. The preemptive version was recently studied in [3]. The technical reason why this semi-online variant cannot be analyzed using our techniques is that a schedule for an initial prefix of jobs cannot be obtained from the final schedule by removing the remaining job. This property is essential in our proof.

On the other hand, some models that do not exactly fit in our description in this paper, yet they can be analyzed in the same way. One example is the model studied in [26, 12]. Here we are guaranteed that the largest job comes last, which fits into our framework, and the scheduler learns that the jobs is the last one when it is released, which is more difficult to handle. To analyze this model, we need to allow the input instance to contain with each job some additional information, for example, here a bit indicating the last job. With this modification, all the proofs work with no changes. We have not performed the actual calculation. Analyzing this and similar models, as well as further extending the scope of our methods remains as a topic for future research.

10.4 Open problems

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

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 approximation 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] 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] G. Dósa and L. Epstein. Preemptive online scheduling with reordering. In *Proc. 17th European Symp. on Algorithms (ESA)*, volume 5757 of *Lecture Notes in Comput. Sci.*, pages 456–467. Springer, 2009.

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

[5] T. Ebenlendr. Semi-online preemptive scheduling: Study of special cases. In *Proc. 8th Int. Conf. on Parallel Processing and Applied Mathematics (PPAM 2009)*, *Lecture Notes in Comput. Sci.*, Springer, 2010. To appear. A full version is available as ITI Series 2010-495, Charles University, Prague, 2010.

[6] T. Ebenlendr, W. Jawor, and J. Sgall. Preemptive online scheduling: Optimal algorithms for all speeds. *Algorithmica*, 53:504–522, 2009.

[7] T. Ebenlendr and J. Sgall. Optimal and online preemptive scheduling on uniformly related machines. *J. Sched.*, 12:517–527, 2009.

[8] M. Englert, D. Özmen, and M. Westermann. The power of reordering for online minimum makespan scheduling. In *Proc. 49th Symp. Foundations of Computer Science (FOCS)*, pages 603–612. IEEE, 2008.

[9] 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.

[10] L. Epstein, J. Noga, S. S. Seiden, J. Sgall, and G. J. Woeginger. Randomized on-line scheduling for two uniform machines. *J. Sched.*, 4:71–92, 2001.

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

[12] L. Epstein and D. Ye. Semi-online scheduling with "end of sequence" information. *J. Comb. Optim.*, 14:45–61, 2007.

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

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

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

[16] Y. He and Y. Jiang. Optimal algorithms for semi-online preemptive scheduling problems on two uniform machines. *Acta Inform.*, 40:367–383, 2004.

[17] Y. He and Y. Jiang. Preemptive semi-online scheduling with tightly-grouped processing times. *J. Comput. Sci. Technol.*, 19:733–739, 2004.

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

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

[20] Y. Jiang and Y. He. Optimal semi-online algorithms for preemptive scheduling problems with inexact partial information. *Acta Inform.*, 44:571–590, 2007.

[21] 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.

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

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

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

[25] J. Wen and D. Du. Preemptive on-line scheduling for two uniform processors. *Oper. Res. Lett.*, 23:113–116, 1998.

[26] G. Zhang and D. Ye. A note on on-line scheduling with partial information. *Computers & Mathematics with Applications*, 44:539–543, 1999.