Preemptive scheduling on a small number of hierarchical machines

György Dósa* Leah Epstein†

Abstract

We consider preemptive offline and online scheduling on identical machines and uniformly related machines in the hierarchical model, with the goal of minimizing the makespan. In this model, each job can be assigned to a subset of the machines which is a prefix of the machine set. We design optimal offline and online algorithms for two uniformly related machines, both when the machine of higher hierarchy is faster and when it is slower, as well as for the case of three identical machines. Specifically, for each one of the three variants, we give a simple formula to compute the makespan of an optimal schedule, provide a linear time offline algorithm which computes an optimal schedule and design an online algorithm of the best possible competitive ratio.

1 Introduction

In this paper we study preemptive online scheduling for cases where distinct processors or machines do not have the same capabilities.

The most general non-preemptive online scheduling model assumes m machines $1, \ldots, m$ and n jobs, arriving one by one, where the information of a job j is a vector $p_j = (p_j^1, p_j^2, \ldots, p_j^m)$ of length m, where p_j^i is the processing time or size of job j if it is assigned to machine i. Each job is to be assigned to a machine before the arrival of the next job. The load of a machine i is the sum of the processing times on machine i of jobs assigned to this machine. The goal is to minimize the maximum load of any machine. This model is known as *unrelated machines* [1]. Many simplified models were defined, both in order to allow the design of algorithms with good performance (which is often difficult, or even impossible, for unrelated machines), and to make the studied model more similar to reality. In the sequel we describe a few models which are relevant to our study.

We consider online algorithms. For an algorithm $\mathcal{A}$, we denote its cost by $\mathcal{A}$ as well. The cost of an optimal offline algorithm that knows the complete sequence of jobs is denoted by OPT. In this paper we consider the (absolute) competitive ratio. The competitive ratio of $\mathcal{A}$ is the infimum $\mathcal{R}$ such that for any input, $\mathcal{A} \leq \mathcal{R} \cdot \text{OPT}$. If the competitive ratio of an online algorithm is at most $\mathcal{C}$, then we say that it is $\mathcal{C}$-competitive. For randomized algorithms, we replace the cost of the algorithm $\mathcal{A}$ by its expected cost $E(\mathcal{A})$, and in this case, the competitive ratio of $\mathcal{A}$ is the infimum $\mathcal{R}$ such that for any input, $E(\mathcal{A}) \leq \mathcal{R} \cdot \text{OPT}$.

Uniformly related machines [1, 4] are machines having speeds associated with them, where machine i has speed s_i and the information that a job j needs to provide upon its arrival is just its size, or processing time on a unit speed machine, which is denoted by p_j. Then we have $p_j^i = p_j/s_i$. If all speeds are equal, then we get *identical machines* [12].

*Department of Mathematics, University of Pannonia, Veszprem, Hungary, dosagy@almos.vein.hu.

†Department of Mathematics, University of Haifa, 31905 Haifa, Israel. lea@math.haifa.ac.il.

Restricted assignment [2] is a model where each job may be run only on a subset of the machines. A job j has a running time associated it, which is the time to run it on any of its permitted machines, which are denoted by M_j. Thus if $i \in M_j$, then we have $p_j^i = p_j$ and otherwise $p_j^i = \infty$. The hierarchical model represents a situation where there is a clear order between the strength of machines, in terms of the jobs they are capable of performing. In the hierarchical model, we get that the set M_j is a prefix of the machines for any j.

In this paper we consider the *restricted related hierarchical model*, where machine i has speed s_i, job j has size p_j on a unit speed machine, and may run on a prefix of the machines $1, \ldots, m_j$, i.e., $M_j = \{1, \ldots, m_j\}$. Therefore, $p_j^i = \frac{p_j}{s_i}$ if $i \leq m_j$ and otherwise $p_j^i = \infty$. Thus, in this model, there are at most m distinct possible subsets of permitted machines. For a job j whose set of permitted machines is $1, \ldots, m_j$, we say that the job is in the set P_{m_j}, or a P_{m_j} job. That is, the set of jobs for which the set of permitted machines is $1, \ldots, i$ is called P_i. We slightly abuse notation and for every value of i denote by P_i also the sum of all P_i jobs.

Possible applications of this model can be computer systems, where the computers differ not only in speed but also in the capacity of their memories. Each job has a memory threshold, which is the minimum memory that a computer must have in order to run it. This creates a hierarchy of the computers. Note that the speeds of computers are not necessarily related to their memories, and thus a computer that is higher in the hierarchy that is based on sizes of memories, is not necessarily faster than a computer that is lower in the hierarchy.

In this paper, we focus on small numbers of machines. We first study the case of two machines. In this case, it is reasonable to assume that the machine that is capable of running any job is faster, since this is a stronger machine. However, the opposite case can occur in real life as well, when the machine that cannot run every job, is more specialized, and works faster when it is running the jobs that it is capable of running. We further consider the case of three identical speed machines in the hierarchical model.

We study preemptive scheduling, where the processing of a job can be shared between several machines. Thus upon the arrival of job j, it may be cut into pieces to be packed as independent jobs, under the restriction that no two parts of the same job can run in parallel on different machines. The processing times of the different parts of a job are calculated accordingly (i.e., as if these are indeed independent jobs). The notion of preemptive scheduling is relevant only to models where the role of time is clear, and therefore it is irrelevant to the general case of unrelated machines. Note that in this model, idle time may be useful. Thus the load of a machine is the completion time of any part of job assigned to it. Alternatively, the load of a machine is the sum of processing times of parts of jobs assigned to this machine (as they are defined to be on this machine) plus the total duration during which the machine is idle (but did not complete all the parts of jobs it needs to run). The makespan is again the maximum load of any machine.

Previous results. The hierarchical model for general m was studied by Bar-Noy, Freund and Naor [3] (see also [7]). They designed a non-preemptive $e + 1 \approx 3.718$-competitive algorithm.

Jiang, He and Tang [13], and independently, Park, Chang and Lee [14] studied the problem on two identical speed hierarchical machines. They both designed non-preemptive $\frac{5}{3}$-competitive algorithms and showed that this is best possible. The paper [13] considered a preemptive model in which idle time is not allowed. This is a restricted type of preemptive scheduling, where all machines need to be occupied starting from time zero and until they complete to run the parts of jobs assigned to them. They designed a $\frac{3}{2}$-competitive algorithm and showed this is best possible.

It is known that in the restricted assignment model, preemption and even scheduling jobs

fractionally, i.e., possibly scheduling several parts of the same job in parallel, does not change the order of growth of the best possible competitive ratio, which is $\Theta(\log n)$ [2]. Preemptive scheduling on identical machines and uniformly related machines was widely studied. For many problems tight bounds on the competitive ratio are known. Chen, Van Vliet and Woeginger gave a preemptive optimal algorithm and a matching lower bound for *identical machines* [6] (see also [15]). The competitive ratio of the algorithm is $m^m/(m^m-(m-1)^m)$. A lower bound of the same value was given independently by Sgall [16]. For *two uniformly related machines*, with speed ratio $s \geq 1$ between the speeds of the two machines, the tight competitive ratio is known to be $(s+1)^2/(s^2+s+1)$, given by Epstein et al. [10] and independently by Wen and Du [17]. Those results were extended for a class of m uniformly related machines with *non-decreasing speed ratios* [9]. The tight bound in that case is a closed formula, which is a function of all the speeds. An optimal (in terms of competitive ratio) online algorithm for any set of speeds was given by Ebenlendr, Jawor and Sgall [8]. Given a combination of speeds, the competitive ratio is a solution of a linear program, and never exceeds $e \approx 2.718$. The paper shows that the result of Epstein [9] actually holds for a wider class of speed sets, and gives a closed formula for three machines and any combination of speeds.

A lower bound of 2 on the overall (maximum over all speed combinations) competitive ratio was given by Epstein and Sgall [11], and improved to 2.054 by Ebenlendr et al. [8].

Our results. We provide a complete solution for several problems. We consider the model of two hierarchical machines with all speed combinations as well as the model of three hierarchical machines of identical speeds. This gives three variants of the problem, two machines, where the first machine is faster, two machines, where the first machine is slower, and three identical machines.

In each of these three variants that we study, we construct a formula to compute the makepsan of an optimal schedule, and design a linear time offline algorithm which constructs such a schedule.

We design an online algorithm of best possible competitive ratio for each one of the three cases. The algorithms are deterministic but the lower bounds hold for randomized algorithms as well. All algorithms use idle time, and we show that deterministic algorithms which do not use idle time cannot achieve the same competitive ratios. The competitive ratios are as follows: $\frac{s(s+1)^2}{s^3+s^2+1}$ for two machines, where s is the speed ratio between the machines, and the first machine is faster, $\frac{(s+1)^2}{s^2+s+1}$ for two machines, where s is the speed ratio between the machines, and the second machine is faster, and $\frac{3}{2}$ for three machines.

2 Two machines, where machine 1 is faster

In this section we assume without loss of generality that the speed of machine 1 is $s \geq 1$, and the speed of the other machine is 1. Recall that P_1 is the set of jobs that must be assigned to machine 1, and P_2 is the set of jobs that can be assigned to any machine. Let $P_{\max}$ be the size of the largest job in the set P_2.

We start with some bounds that are valid for any solution and in particular, for an optimal offline algorithm.

Lemma 1 $\text{LB} = \max\left\{\frac{P_1}{s}, \frac{P_1+P_2}{s+1}, \frac{P_{\max}}{s}+P_1\frac{s-1}{s^2}\right\}$ *is a lower bound on the cost of any solution. Moreover,* $P_2 \leq \frac{P_1}{s}$ *holds if and only if* $\text{LB} = \frac{P_1}{s}$.

Proof. The first bound holds since all jobs in P_1 must be scheduled on the first machine. The second bound is valid due to the fact that the sum of all processing times is $P_1 + P_2$ and $s+1$ is the sum of machine speeds. In the third bound, if $P_{\max} \leq \frac{P_1}{s}$, then we get $\frac{P_{\max}}{s} + P_1 \frac{s-1}{s^2} = \frac{P_1}{s} + \frac{P_{\max} - \frac{P_1}{s}}{s} \leq \frac{P_1}{s}$. Thus, we only need to consider the case $P_{\max} > \frac{P_1}{s}$. Consider an assignment, and a job j of size $P_{\max}$. Let μ be the part of this job assigned to machine 1 and $P_{\max} - \mu$ the part assigned to machine 2. Clearly, we have that the first machine cannot complete all jobs assigned to it earlier than the time $\frac{P_1+\mu}{s}$. Thus, if $\mu \geq P_{\max} - \frac{P_1}{s}$, then we get the required lower bound on the makespan. On the other hand, in order for j to be completed, it has to run on machine 2 for a time of $P_{\max} - \mu$ and on machine 1 for time of $\frac{\mu}{s}$. These times do not necessarily need to be in this order or continuous, but there can be no overlap and thus we get that this job is completed no earlier than the time $P_{\max} - (s-1)\frac{\mu}{s}$. Now, if $\mu \leq P_{\max} - \frac{P_1}{s}$, then we get the required bound on the makespan.

For the second part, $P_2 \leq \frac{P_1}{s}$ holds if and only if $\frac{P_1+P_2}{s+1} \leq \frac{P_1}{s}$ holds. Thus we got that if $\text{LB} = \frac{P_1}{s}$, then $P_2 \leq \frac{P_1}{s}$. To complete the other direction, we note that if $P_2 \leq \frac{P_1}{s}$ is true, then $P_{\max} \leq P_2 \leq \frac{P_1}{s}$ holds, and so $\frac{P_{\max}}{s} + P_1 \frac{s-1}{s^2} \leq \frac{P_1}{s}$. ■

We use the value LB for our online algorithm. It is possible to show that the value LB is not only a lower bound on the makespan of an optimal solution, but is actually equal to this value.

Theorem 2 *Given a set of jobs J, the value* LB *is equal to the makespan of an optimal schedule, and a schedule of this cost can be constructed by a linear time algorithm.*

Proof. In order to prove the theorem, we need to consider the three cases, case by case, and show a linear time algorithm for each one of the cases. Clearly, the value LB can be computed in linear time. Let k be an index of a job of size $P_{\max}$. If $\text{LB} = \frac{P_1}{s}$, then due to Lemma 1, we have $P_2 \leq \frac{P_1}{s}$. Thus we schedule all P_1-jobs on the first machine, exactly during the time slot $[0, \text{LB}]$. All other jobs are scheduled on the second machine during the time slot $[0, P_2] \subseteq [0, \text{LB}]$. Next, if $\text{LB} = \frac{P_{\max}}{s} + P_1 \frac{s-1}{s^2}$, then we divide job k into two parts of sizes $\frac{P_1}{s}$ and $P_{\max} - \frac{P_1}{s}$. During the time slot $[0, \frac{P_1}{s}]$, we run the P_1-jobs on the first machine, and the first part of job k on the second machine. Note that given the value of LB we have $\text{LB} \geq \frac{P_1}{s}$, and $P_{\max} \geq \frac{P_1}{s}$. During the time slot $[\frac{P_1}{s}, \frac{P_{\max}}{s} - \frac{P_1}{s^2} + \frac{P_1}{s}]$, the first machine runs the remainder of job k, and the second machine runs all remaining P_2-jobs, which have a total size of $P_2 - P_{\max}$, until time $\frac{P_1}{s} + P_2 - P_{\max}$. We need to show that $\frac{P_1}{s} + P_2 - P_{\max} \leq \text{LB} = \frac{P_{\max}}{s} + P_1 \frac{s-1}{s^2}$. Using $\frac{P_1+P_2}{s+1} \leq \text{LB}$, we have $\frac{P_1}{s} + P_2 - P_{\max} = P_1 + P_2 - P_{\max} - P_1 + \frac{P_1}{s} \leq (s+1)\text{LB} - s\text{LB} = \text{LB}$.

It is left to consider the case $\frac{P_1+P_2}{s+1} = \text{LB}$. If $\text{LB} \geq P_{\max}$, then we assign all jobs of P_1 to the first machine until time $\frac{P_1}{s} \leq \text{LB}$. Afterwards, we keep assigning jobs to this machine until time LB. The rest of the jobs are assigned to the second machine, starting from time zero. There is enough space for them since $(s+1)\text{LB} \geq (P_1 + P_2)$. If some job was split between the two machines, then let a be the part assigned to the first machine, and b to the second machine. We have $a + b \leq P_{\max}$. The starting time on the first machine is $\text{LB} - \frac{a}{s}$ and the completion time on the second machine is b. We have $b \leq \text{LB} - \frac{a}{s}$ since $\frac{a}{s} + b \leq a + b \leq \text{LB}$. Thus there is no overlap.

Otherwise, we define $\mu = s\frac{P_{\max} - \text{LB}}{P_{\max}(s-1)}$. We assign a part of size $\mu P_{\max}$ of job k on the first machine from time $\text{LB} - \frac{\mu P_{\max}}{s}$ until time LB, and the rest on the second machine from time zero till time $\text{LB} - \frac{\mu P_{\max}}{s} = (1-\mu)P_{\max}$, by definition of μ. The remaining slots on the machines are non-overlapping. We next assign all P_1-jobs to the first machine starting from time zero, and afterwards, the remaining P_2-jobs, to empty slots. Since $\text{LB} \geq \frac{P_1+P_2}{s+1}$, there is enough space for all jobs. We need

to show that the space on the first machine is enough for the P_1-jobs, i.e., that $\frac{P_1}{s} + \frac{\mu P_{\max}}{s} \leq \text{LB}$. By the definition of μ, this is equivalent to, $\frac{P_1}{s} + \frac{P_{\max} - \text{LB}}{s-1} \leq \text{LB}$, or $\text{LB} \geq \frac{P_{\max}}{s} + \frac{P_1(s-1)}{s^2}$, which holds by the definition of LB. ■

We replace the notation LB with the notation OPT since we have shown that LB is exactly the cost of an optimal schedule. We continue with a lower bound on the competitive ratio.

Lemma 3 *Any randomized algorithm $\mathcal{A}$ has competitive ratio of at least $\alpha(s)$, where $\alpha(s) = \frac{s(s+1)^2}{s^2(s+1)+1}$.*

Proof. We specify a sequence which proves the statement. We use an adaptation of Yao's principle [18] for proving lower bounds for *randomized algorithms*. It states that a lower bound for the competitive ratio of deterministic algorithms on a fixed distribution on the input is also a lower bound for randomized algorithms and is given by $\frac{E(\mathcal{A})}{E(\text{OPT})}$.

Let $X \geq 1$ be a real number. The list of jobs is as follows. The first job is a P_1-job, where $p_1 = s$. The following two jobs are P_2-jobs, $p_2 = \frac{X+s}{s+1}$ and $p_3 = \frac{s(X-1)}{s+1}$. Note that we have $p_2 + p_3 = X$, and $p_3 = s\,(p_2 - 1)$. The fourth job is a P_2-job, $p_4 = sX + 1$. The fifth job is a P_1-job, $p_5 = sX + s^2X$. According to Yao's result, we consider a deterministic algorithm $\mathcal{A}$.

Denote by OPT_k, $\mathcal{A}_k$ the cost of an optimal solution and of algorithm $\mathcal{A}$ for the input which consists of the first k jobs. We have $\text{OPT}_3 = p_2 = \frac{s+X}{s+1}$ (by assigning job 2 to the second machine, and the other jobs to the first machine), $\text{OPT}_4 = X + 1$ (by assigning job 1 to machine 1 in parallel with a part of job 4 on machine 2, and the other jobs to machine 2, in parallel with the other part of job 4 on machine 1), and $\text{OPT}_5 = sX + X + 1$ (by assigning the P_1-jobs, and only these jobs, to machine 1).

Suppose that algorithm $\mathcal{A}$ is r-competitive, for some $r \geq 1$. Let c be the sum of parts of the second and third jobs which are assigned to the first machine, let d_1 be the sum of parts of these jobs assigned to the second machine when the first machine is idle (after the first three jobs are assigned), and d_2 the sum of parts of these jobs assigned to the second machine when the first machine is busy. Then $d_1 + d_2 + c = X$. Let b be the part of the fourth job which is assigned to the first machine, then a part of size $sX + 1 - b$ of this job is assigned to the second machine. Then the next inequalities hold.

$$\begin{aligned}
\mathcal{A}_3 &\geq 1 + \frac{c}{s} + d_1 \\
\mathcal{A}_5 &\geq 1 + \frac{c}{s} + \frac{b}{s} + (X + sX) \\
\mathcal{A}_4 &\geq d_2 + (sX + 1 - b) + \frac{b}{s}
\end{aligned}$$

All inequalities are lower bounds on the makespan at various times. In the first inequality, the total size assigned to run on the first machine after the first three jobs have arrived is $s + c$, and an additional size of d_1 is assigned to run on the other machine when the first machine is idle. The second inequality holds due to the total size assigned to the first machine after all jobs have arrived. The third inequality holds due to the time after four jobs have arrived. There are d_2 units of time busy on both machine before the fourth job is scheduled. The time to run this job due to the way it is split between machines is $\frac{b}{s} + p_4 - b$.

We use the sequence of the first three jobs with probability $\frac{1}{s+1}$, the sequence of four first jobs with probability $\frac{1}{s+1}$, and the sequence of all five jobs with probability $\frac{s-1}{s+1}$. Thus the expected cost of an optimal offline algorithm is

$$\frac{1}{s+1}\left(\textsc{opt}_3+(s-1)\textsc{opt}_5+\textsc{opt}_4\right)=\frac{1}{s+1}\left(\frac{s+X}{s+1}+(s-1)\left(sX+X+1\right)+X+1\right)\ .$$

On the other hand, the expected cost of $\mathcal{A}$ is at least

$$\begin{aligned}&\frac{1}{s+1}\left(\mathcal{A}_3+(s-1)\mathcal{A}_5+\mathcal{A}_4\right)\\&\geq\frac{1}{s+1}\left(\left(1+\frac{c}{s}+d_1\right)+\left(s-1+c-\frac{c}{s}+b-\frac{b}{s}+X\left(s-1\right)\left(s+1\right)\right)+\left(d_2+\left(sX+1-b\right)+\frac{b}{s}\right)\right)\\&=\frac{s+s^2X+sX+1}{s+1},\end{aligned}$$

where the last equality follows from $c+d_1+d_2=X$.

Since the algorithm is r-competitive, we get,

$$s+s^2X+sX+1\leq r\left(\frac{s+X}{s+1}+(s-1)\left(sX+X+1\right)+X+1\right),$$

from which follows that

$$r\geq\frac{\left(s+1\right)^2\left(sX+1\right)}{s+X+s\left(s+1\right)\left(sX+1\right)}=\frac{\left(s+1\right)^2}{\frac{s+X}{sX+1}+s\left(s+1\right)}.$$

Letting X tend to ∞, the right hand side can get arbitrarily close to $\frac{(s+1)^2}{s(s+1)+1/s}$ and therefore the next inequality holds for every $s\geq 1$:

$$r\geq\frac{\left(s+1\right)^2}{s\left(s+1\right)+\frac{1}{s}}.$$

■

Now we turn to show that there exists an algorithm which achieves the previous lower bound. In the sequel we use $\alpha=\alpha(s)=\frac{(s+1)^2}{s^2+s+1/s}$, and introduce an α-competitive (thus optimal) algorithm. We define $t(s)=t=\frac{s^2+s-1}{s^2+s+1/s}$ and $1-t=\frac{1+1/s}{s^2+s+1/s}$; both are positive and smaller than 1 for any speed s.

Lemma 4 *The next properties hold for any $s\geq 1$ for t and α defined above.*

1. $1+\frac{t}{s}=\alpha$.
2. $s\left(1-t\right)\left(s+1\right)=\alpha$ *and thus* $\left(1-t\right)\left(s+1\right)\leq\alpha$.
3. $\left(1-t+t/s\right)\frac{(s+1)^2}{s+2}=\alpha$

Proof.

1. $1+\frac{t}{s}=\frac{s^2+s+1/s+s+1-1/s}{s^2+s+1/s}=\frac{s^2+s+s+1}{s^2+s+1/s}=\frac{(s+1)^2}{s^2+s+1/s}=\alpha.$

2. $s\,(1-t)\,(s+1)=s\frac{(1+1/s)(s+1)}{s^2+s+1/s}=s\frac{(s+1)^2}{s(s^2+s+1/s)}=\alpha.$

3. $(1-t+t/s)\,\frac{(s+1)^2}{s+2}=\frac{1+s+1}{s^2+s+1/s}\frac{(s+1)^2}{s+2}=\frac{(s+1)^2}{s^2+s+1/s}=\alpha.$

■

Let J be a given prefix of the input. We use the following definitions for the algorithm. We partition the time slot $[0,C]$ into sub-intervals according to the subset of the machines that are busy during the various intervals. There are three types of intervals that can be used to assign a new job, which are intervals where at least one machine is not busy. An interval is called "left hole", if the second machine is busy and the first machine is not, and "right hole", if the roles of machines are opposite. If both machines are not busy, then we call the interval "super hole". We use the same terms to denote the union of intervals of the same status, thus e.g., we refer by "super hole" to the union of intervals where both machines are not busy. The fourth type of interval where both machines are busy is called "dense", and the union of dense intervals is called "the dense part". The lengths of the left hole, right hole, super hole, and dense part be denoted by $L(J)$, $R(J)$, $S(J)$, and $D(J)$, respectively. These values refer to the situation *before* the last job in J is scheduled.

The algorithm will follow the next conditions.

a. All P_1-jobs are assigned to the first machine. P_2-jobs are always split into two parts of ratio $t:(1-t)$ between the sizes of these part. The assignment is done as follows. First, the t fraction of the job is assigned to the first machine, next the remaining part, which is a $(1-t)$ fraction of the job, is assigned to the second machine.

b. We denote by $C(J)=\alpha\textsc{opt}(J)$, where $\textsc{opt}(J)$ is the current value of the bound $\textsc{opt}$ (which is computed given the jobs of J). Every job is assigned to a set of time intervals which are fully contained in the interval $[0,C(J)]$. We prove later that such an assignment is always possible.

c. When a new job is assigned, we try to occupy as much as possible of the super hole. If necessary, the left or right hole are used as well. Consider the assignment of job j of size p_j, which is a P_2-job. We define $y(J)=\max\{(1-t)\,p_j-R(J),0\}$, where $J=\{1,2,\ldots,j\}$. If $y(J)$ is positive, then since we would like to assign a part of size $(1-t)p_j$ of j to the second machine, at least a part of size $y(J)$ must use the super hole on this machine. This means that we may use at most an interval of length $S(J)-y(J)$ of the super hole on the first machine, to schedule the part of size tp_j, which should run on the the first machine.

We are now ready to define our algorithm.

Algorithm 1.

Denote the next job by p_j. Let $C(J)$ be defined as $C(J)=\alpha\textsc{opt}(J)$ (where $J=\{1,2,\ldots,j\}$). Job j is scheduled within the interval $[0,C(J)]$ as follows.

1. $p_j\in P_1$. Schedule a part of j which is as large as possible into the super hole, if necessary, schedule the remainder of j into the left hole.

2. $p_j \in P_2$. First a part of size tp_j is assigned to the first machine, a maximum amount of it is assigned to the super hole, but no more than $S(J) - y(J)$. The remainder, if it exists, is assigned into the left hole. Next, the part of size $(1-t)\,p_j$ of the job is assigned to the second machine. First, a maximum amount is assigned to the super hole, and then the remaining part is assigned into the right hole.

We illustrate the action of the algorithm using the followings example. Consider a sequence of three jobs, where the first two are in P_2, and the third one is in P_1. Their sizes are 13, 26 and 13. Let $s = 2$, then we get $\alpha = \frac{18}{13}$, $t = \frac{10}{13}$ and $1 - t = \frac{3}{13}$. The values of $C(J)$ for the three jobs are 9, 18 and 24. Before the assignment of the first job, we have $J = \{1\}$ and there is only a super hole of size 9 (the lengths of the left hole, the right hole and the dense part are all zero). Therefore the value of $y(J)$ is 3, so only a part of length 6 of the super hole may be used on the first machine. According to the algorithm, a part of size 10 should be assigned to the first machine, to occupy a slot of size 5. The part of size 3 is assigned to the second machine. Since there is no right hole, it is assigned to the super hole (see Figure 1).

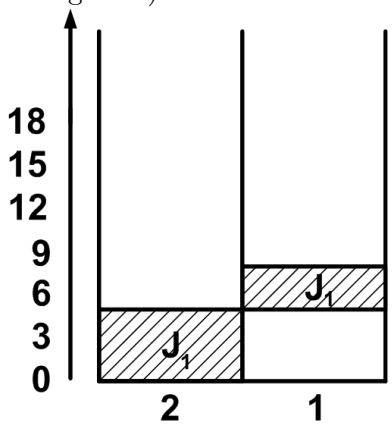

Figure 1: The assignment of the first job in the example for Algorithm 1

Next, before the second job is assigned, we have $J = \{1, 2\}$ and there is a super hole of length 10, a left hole of length 3, and a right hole of length 5. There is no dense part at this time (i.e., it has a length of zero). Therefore at this time $y(J) = 1$. The part of the job of size 20 should be assigned into time slots of total length 10 on the first machine. However, we have $S(J) - y(J) = 10 - 1$ and thus one unit of time is reserved for the second job in the left hole (the additional unit of time in the super hole is reserved to be used on the second machine). The part of the job of size 6 is assigned to the second machine, one unit of size into the super hole, and the rest into the right hole (see Figure 2).

Finally, before the last job is assigned, we have $J = \{1, 2, 3\}$ and there is a super hole of length 6, a left hole of length 3, a right hole of length 9, and a dense part of length 6. Since this job is in P_1, a part of the job of size 12 is assigned to the super hole (on the first machine) and the remainder into the left hole (see Figure 3).

We need to show that the algorithm can assign all the jobs, that is, that the designated intervals are large enough to contain each job.

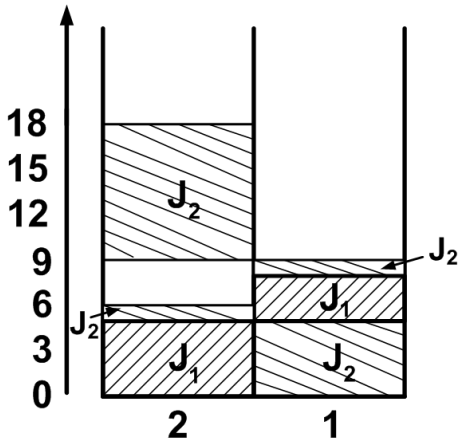

Figure 2: The assignment of the second job in the example for Algorithm 1

Theorem 5 *The algorithm is correct, i.e., it can assign all the jobs for any input.*

Proof. First we prove that the P_1-jobs and t part of the P_2-jobs always fit into the time-interval $[0, C(J)]$ on the first machine, and the $(1-t)$ part of the P_2-jobs always fit into the time-interval $[0, C(J)]$ on the second machine, where $C(J) = \alpha \text{OPT}(J)$. We first show that there is enough space, and later we show that no overlap is created. The key property of the algorithm is to schedule parts of jobs on the first machine ensuring that there is enough available space left on the second machine in this process.
Case a, $P_2 \leq P_1/s$. By Lemma 1, $\text{OPT} = P_1/s$. Then we have,

$$\frac{P_1 + tP_2}{s} \leq \frac{P_1 + t\frac{P_1}{s}}{s} = \left(1 + \frac{t}{s}\right)\frac{P_1}{s} \leq \alpha \text{OPT} \ ,$$

and

$$(1-t)\,P_2 \leq (1-t)\,\frac{P_1}{s} \leq \alpha \frac{P_1}{s} = \alpha \text{OPT} \ ,$$

where the first inequality holds due to Lemma 4, and the second one trivially holds since $1 - t < 1 < \alpha$.
Case b, $P_2 > P_1/s$, then,

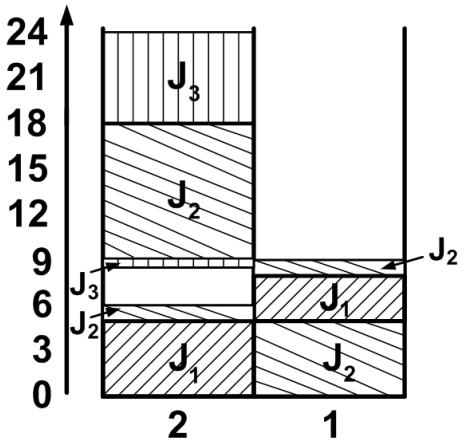

Figure 3: The assignment of the third job in the example for Algorithm 1

$$\frac{P_1 + tP_2}{s} = \frac{P_1}{s} + \frac{s+1-1/s}{s^2+s+1/s}P_2 = \frac{s+1}{s^2+s+1/s}P_1 + \frac{1/s^2}{s^2+s+1/s}P_1 + \frac{s+1-1/s}{s^2+s+1/s}P_2$$
$$\leq \frac{s+1}{s^2+s+1/s}P_1 + \frac{1/s}{s^2+s+1/s}P_2 + \frac{s+1-1/s}{s^2+s+1/s}P_2 = \frac{s+1}{s^2+s+1/s}\left(P_1+P_2\right)$$
$$= \frac{(s+1)^2}{s^2+s+1/s}\frac{P_1+P_2}{s+1} \leq \alpha \mathrm{OPT}$$

and

$$(1-t)\,P_2 = \frac{1+1/s}{s^2+s+1/s}P_2 \leq \frac{(s+1)^2}{s\left(s^2+s+1/s\right)}\frac{P_1+P_2}{s+1} \leq \alpha \mathrm{OPT}.$$

It therefore follows, that job parts that are assigned to the two machines always fit into the time interval $[0, C(J)]$. It remains to show that the assignment can be done properly, avoiding overlap.

All P_1-jobs are assigned to one machine, thus they cannot create overlap and it remains to deal with P_2-jobs. Let X be the first P_2-job for which the algorithm is not correct (we use X to denote its size as well). We define J to be the sequence of jobs up to X (including X). If $y(J) = 0$, then we get that $R(J) \geq (1-t)p_j$, and thus the part of X of size $(1-t)\,X$ can be assigned. Moreover, it is allowed to use all the empty space on the first machine for the part of X of size tX, we already showed that it is enough to assign this job. On the other hand, if $y(J) > 0$, then this means that a part of size $y(J)$ of the super hole was reserved for this part, unless the super hole is not large enough. However, we know that there is enough space for each job, thus $S(J) + R(J) \geq (1-t)\,X$, and the second part can always be assigned.

In the case $\textsc{opt}(J) = \frac{P_1}{s}$, we have $P_2 \leq \frac{P_1}{s}$ by Lemma 3, thus, not only the parts of size $(1-t)\,P_2$, but even the entire amount P_2 fits into the right hole, which is of size at least $\frac{P_1}{s}$. Thus $y(J) = 0$ in this case. Since we proved that on the first machine there is enough space for all parts of jobs, if we are allowed to use the entire space on that machine, then it is left to consider the other two options of the value of $\textsc{opt}$ and the case $y(J) > 0$.

We need to show that the amount of space allocated on the first machine is large enough to accommodate the part of size tX. This space includes the complete left hole plus a part of the super hole. We require that,

$$\begin{aligned} s\,(S(J) - [(1-t)\,X - R(J)]) + sL(J) \geq tX &\iff s\,(S(J) + R(J) + L(J)) \geq s\,(1-t)\,X + tX \\ &\iff s\,(C(J) - D(J)) \geq (s + t - st)\,X \iff C(J) - D(J) \geq (1 - t + t/s)\,X \\ &\iff D(J) \leq C(J) - (1 - t + t/s)\,X \end{aligned} \tag{1}$$

As it turned out, the exact sizes of the left hole, right hole and super hole do not matter, but only their total size, which needs to be large enough. That is, the dense part of the schedule must be small enough. Thus now we compute this dense part of the schedule as follows. First suppose the job that precedes X does not cause an increase of the dense part of the schedule. Note that the functions $\textsc{opt}(J)$ and $C(J)$ are non-decreasing in the variables P_1, P_2, and $P_{\max}$, thus, if (1) does not hold, then we can omit this job from the instance and get a counterexample. Clearly, if $D(J) = 0$, then (1) holds. This process must terminate before all jobs are removed since at least two jobs need to be assigned in order to create a non-empty dense part. Thus it can be supposed that the job right before X increases the dense part of the schedule. From this it follows that just before X arrives, (before the value of C is updated), there is no super hole. The dense part can increase only if the super hole is fully used, since if the super hole is not fully used on the first machine, then the space which is reserved within the super hole is filled on the second machine. Consider the moment of the execution of the algorithm just before assigning X, i.e. right after the last job before X was assigned.

The total size of parts of jobs assigned to the first machine is $P_1 + t\,(P_2 - X)$, and they occupy a total time which is $\frac{P_1 + t(P_2 - X)}{s}$. The total size of parts of jobs assigned to the second machine is $(1-t)\,(P_2 - X)$. Both machines may use the interval up to time $C(J\backslash X)$. Since there is (temporarily) no super hole, the second machine is busy during the time periods where the first machine is not busy, and thus the overlap when both machines are busy is,

$$D(J) = (1-t)\,(P_2 - X) + \frac{P_1 + t\,(P_2 - X)}{s} - C(J\backslash X) = \frac{P_1}{s} + (1 - t + t/s)\,(P_2 - X) - C(J\backslash X)$$

Substituting this value into (1), it remained to show that

$$\begin{aligned} &\frac{P_1}{s} + (1 - t + t/s)\,(P_2 - X) - C(J\backslash X) \leq C(J) - (1 - t + t/s)\,X, \text{ i.e.} \\ &\qquad \frac{P_1}{s} + (1 - t + t/s)\,P_2 \leq \alpha\,(\textsc{opt}(J) + \textsc{opt}(J\backslash X))\,. \end{aligned} \tag{2}$$

Case a, $\textsc{opt}(J) = \frac{P_1+P_2}{s+1}$. Let $Z \geq 0$ be a value such that $\frac{P_1+P_2}{s+1} = \frac{P_1}{s} + Z$. Then $P_2 = \frac{P_1}{s} + (s+1)\,Z$, and since $\frac{P_{\max}}{s} + P_1 \frac{s-1}{s^2} \leq \frac{P_1+P_2}{s+1} = \frac{P_1}{s} + Z$, the maximum size of any P_2-job is at most $\frac{P_1}{s} + sZ$. In particular, it follows that $X \leq \frac{P_1}{s} + sZ$. Therefore, $\textsc{opt}(J\backslash X) \geq \frac{P_1 + (P_2 - X)}{s+1} \geq$

$\frac{P_1+\left(\frac{P_1}{s}+(s+1)Z-\frac{P_1}{s}-sZ\right)}{s+1}=\frac{P_1+Z}{s+1}$. We get,

$$\frac{P_1}{s}+(1-t+t/s)\,P_2=\frac{P_1}{s}+\frac{s+2}{s^2+s+1/s}\left(\frac{P_1}{s}+(s+1)\,Z\right)$$

$$=\left(1+\frac{s+2}{s^2+s+1/s}\right)\frac{P_1}{s}+\frac{(s+2)\,(s+1)}{s^2+s+1/s}Z=\frac{s^2+2s+2+1/s}{s^2+s+1/s}\frac{P_1}{s}+\frac{(s+2)\,(s+1)}{s^2+s+1/s}Z$$

$$\leq\frac{(s+1)\,(2s+1)}{s^2+s+1/s}\frac{P_1}{s}+\frac{(s+1)\,(s+2)}{s^2+s+1/s}Z=\alpha\left(\frac{2s+1}{s+1}\frac{P_1}{s}+\frac{s+2}{s+1}Z\right)=$$

$$\alpha\left(\left(1+\frac{s}{s+1}\right)\frac{P_1}{s}+\left(1+\frac{1}{s+1}\right)Z\right)=\alpha\left(\frac{P_1}{s}+Z+\frac{P_1+Z}{s+1}\right)\leq\alpha\left(\textsc{opt}(J)+\textsc{opt}(J\backslash X)\right)$$

Case b, $\textsc{opt}=\frac{P_{\max}}{s}+P_1\frac{s-1}{s^2}=\frac{P_1+Y}{s}$, where $Y=P_{\max}-\frac{P_1}{s}\geq 0$. We define $P_2'=P_2-P_{\max}$. Using $\frac{P_1+P_2}{s+1}\leq\frac{P_1+Y}{s}$ we get $P_2-P_{\max}=P_2-Y+\frac{P_1}{s}\leq\frac{Y}{s}$, and thus $P_2'\leq\frac{Y}{s}$. Clearly, $X\leq P_{\max}$, and thus $\textsc{opt}(J\backslash X)\geq\frac{P_1+P_2'}{s+1}$. Let $0\leq\gamma\leq 1$ be a value for which $P_2'=\gamma\frac{Y}{s}$. Then we get,

$$\frac{P_1}{s}+(1-t+t/s)\,P_2=\frac{P_1}{s}+\frac{s+2}{s^2+s+1/s}\left(P_{\max}+P_2'\right)=\frac{P_1}{s}+\frac{s+2}{s^2+s+1/s}\left(Y+\frac{P_1}{s}+\gamma\frac{Y}{s}\right)$$

$$=\left(1+\frac{s+2}{s^2+s+1/s}\right)\frac{P_1}{s}+\frac{s+2}{s^2+s+1/s}\left(1+\frac{\gamma}{s}\right)Y=\frac{s^2+2s+2+1/s}{s\,(s^2+s+1/s)}P_1+\frac{(s+2)\,(s+\gamma)}{s\,(s^2+s+1/s)}Y$$

$$\leq\frac{(s+1)^2}{s^2+s+1/s}\frac{2s+1}{s+1}\frac{P_1}{s}+\frac{(s+1)^2}{s^2+s+1/s}\frac{s+1+\gamma}{s\,(s+1)}Y=\alpha\left(\frac{2s+1}{s+1}\frac{P_1}{s}+\left(\frac{1}{s}+\frac{\gamma/s}{s+1}\right)Y\right)$$

$$=\alpha\left(\frac{P_1+Y}{s}+\frac{P_1+\gamma\frac{Y}{s}}{s+1}\right)=\alpha\left(\frac{P_1+Y}{s}+\frac{P_1+P_2'}{s+1}\right)\leq\alpha\left(\textsc{opt}(J)+\textsc{opt}(J\backslash X)\right)$$

where the first inequality follows from the fact that $(s+2)\,(s+\gamma)\leq(s+1)\,(s+1+\gamma)$ holds for any possible value of γ, and $s^2+2s+2+1/s\leq(s+1)\,(2s+1)$ holds for any $s\geq 1$, which completes the proof. ■

We have seen that for any speed $s\geq 1$, Algorithm 1 is α-competitive. Comparing this ratio with Lemma 3, we conclude that Algorithm 1 has an optimal competitive ratio.

As can be seen above, Algorithm 1 uses idle time. An interesting question is whether this is done for convenience. The result of Jiang et al. [13] implies that for $s=1$, an algorithm which does not use idle time has competitive ratio of at least $\frac{3}{2}$. Substituting $s=1$ into our bound we see that using idle time reduces the competitive ratio. Therefore, for $s=1$, an algorithm of optimal competitive ratio (among such that do or do not use idle time) must use idle time. We show that this is true for all values of s, thus motivating the usage of idle time in our algorithm.

Claim 6 *For any $s\geq 1$, a deterministic algorithm which does not use idle time has competitive ratio which is strictly larger than $\alpha(s)=\frac{s(s+1)^2}{s^2(s+1)+1}$.*

Proof. Consider the following sequence and an algorithm $\mathcal{A}$ that does not use idle time. The first job is a P_2-job, where $p_1=s^2$. The second job is a P_1-job where $p_2=s^3$. The only way to avoid idle time is to assign the first job completely to one of the two machines. If the first job is assigned to the second machine, then the only possible schedule is that both machines run jobs from time zero till time s^2. If this job is assigned to the first machine, then we have that this machine is

running both jobs, and completes them at time s^2+s. For this input, the first option is optimal and so in the second case, the competitive ratio is $1+\frac{1}{s} > \alpha(s)$ for $s \geq 1$. In the first case, if $s > \sqrt{2}$, then the sequence which consists of the first job only gives the competitive ratio s. For $s > \sqrt{2}$ we have $s > \alpha(s)$ and we are done. Otherwise (for $s \leq \sqrt{2}$), the sequence continues with a P_2-job, $p_3 = s^2+s$. Since before the arrival of this job, the machines are balanced, it has to be assigned completely to one of the machines, to avoid idle time. If it is assigned to the second machine, then we get $\mathcal{A} = 2s^2+s$, whereas OPT $\leq s^2+s$ (by running only the third job on the second machine), which gives the competitive ratio $\frac{2s+1}{s+1} > \alpha(s)$ for all $s \geq 1$. Otherwise, the last job is a P_1-job, $p_4 = s^3+s^2$. This job must be assigned to the first machine and thus we get $\mathcal{A} = \frac{s^3+s^2+s+s^3+s^2}{s} = 2s^2+2s+1$, and OPT $= 2s^2+s$, which gives the competitive ratio $\frac{2s^2+2s+1}{2s^2+s}$. This value is strictly larger than $\alpha(s)$ for all $s \leq \sqrt{2}$. ■

3 Two machines, where machine 2 is faster

Assume that speed of the first machine is 1, and of the second machine is $s \geq 1$. Note that without hierarchy levels there exists a $\frac{(s+1)^2}{s^2+s+1}$-competitive optimal algorithm for the problem [10, 17]. We will see that this case is simpler than the previous one, since we will get this competitive ratio.

We again start with some bounds that are valid for any solution and in particular, for an optimal offline algorithm.

Lemma 7 LB $= \max\left\{P_1, \frac{P_1+P_2}{s+1}, \frac{P_{\max}}{s}\right\}$ *is a lower bound on the cost of any solution. Moreover,* LB $= P_1$ *if and only if* $P_1 \geq \frac{P_2}{s}$.

Proof. The first bound holds since all jobs in P_1 must be scheduled on the first machine, which has speed 1. The second bound is valid due to the fact that the sum of all processing times is P_1+P_2 and $s+1$ is the sum of machine speeds. The third bound holds since the job of size $P_{\max}$ must be completed.

To prove the second part, we have $P_1 \geq \frac{P_1+P_2}{s+1}$ if and only if $P_1 \geq \frac{P_2}{s}$, thus it remains to show that if $P_1 \geq \frac{P_2}{s}$, then we have $\frac{P_{\max}}{s} \leq P_1$. This holds since $P_{\max} \leq P_2$. ■

We use the value LB for our online algorithm. It is again possible to show that the value LB is not only a lower bound on the makespan of an optimal solution, but is actually equal to this value.

Theorem 8 *Given a set of jobs J, the value* LB *is equal to the makespan of an optimal schedule, and a schedule of this cost can be constructed by a linear time algorithm.*

Proof. Let k be an index of a job of size $P_{\max}$. If $\frac{P_{\max}}{s} + P_1 \geq$ LB, then we use the following schedule. All P_1-jobs are assigned on machine 1, from time zero until time P_1 and job k is assigned on machine 2, starting from time LB $- \frac{P_{\max}}{s}$ until time LB. The unused slots on the two machines are non-overlapping, thus we use them to assign the other jobs. There is enough room since $(s+1)$LB $\geq P_1+P_2$.

Otherwise, let μ be defined as $\mu = \frac{s\cdot\text{LB}-s\cdot P_1-P_{\max}}{(s-1)P_{\max}} \geq 0$. We assign a part of job k of size $\mu P_{\max}$ to the first machine, during the time interval $[P_1, P_1+\mu P_{\max}]$, and the rest of the job to the second machine during the time interval $[P_1+\mu P_{\max}, \text{LB}]$, which due to the definition of μ can accommodate exactly the part of size $(1-\mu)P_{\max}$ of job k. The rest of the P_2-jobs are

scheduled within the remaining time slots, which are non-overlapping. There is enough room since $(s+1)\text{LB} \geq P_1 + P_2$. ■

Once again, we replace the notation LB with the notation OPT since we have shown that LB is exactly the cost of an optimal schedule. We continue with a lower bound on the competitive ratio.

In this section we prove that the best possible competitive ratio is $\beta(s) = \beta = \frac{(s+1)^2}{s^2+s+1} = 1 + \frac{s}{s^2+s+1}$. We start with the lower bound.

Lemma 9 *Any randomized algorithm $\mathcal{A}$ has competitive ratio of at least $\beta(s)$, where $\beta(s) = \frac{(s+1)^2}{s^2+s+1} = 1 + \frac{s}{s^2+s+1}$.*

Proof. Any instance of the problem on uniformly related machines (with no hierarchy) is an instance of our problem where all jobs are P_2-jobs. Therefore, any lower bound for that case is valid for our problem. Therefore, since a lower bound of $\beta(s) = \frac{(s+1)^2}{s^2+s+1} = 1 + \frac{s}{s^2+s+1}$ on the competitive ratio of any algorithm for the problem on uniformly related machines with no hierarchy is given in references [10, 17], this implies the lower bound the more general problem with hierarchy. ■

We introduce a new, $\frac{(s+1)^2}{s^2+s+1}$-competitive algorithm. We use notations similar to those in the definition of Algorithm 1. We define $C(J) = \beta\text{OPT}(J)$, and each job is scheduled fully within the time interval $[0, C]$. We use the same definitions of left hole, right hole, super hole, and dense part, as in the previous section. In this case, a greedy approach that uses the second machine as much as possible, and prefers the super hole to the right hole, gives an algorithm of best possible competitive ratio.

Algorithm 2.

Let p_j be the next job, and define $C = \beta\text{OPT}(J)$ (where $J = \{1, 2, \ldots, j\}$). Job j is scheduled within the interval $[0, C]$ as follows.

1. $p_j \in P_1$. Schedule a part of j which is as large as possible into the super hole, if necessary, schedule the remainder of j into the left hole.

2. $p_j \in P_2$. Schedule a maximum part of the job into the super hole on the second machine, if necessary continue and assign a maximum part of the remainder into the right hole, the remaining part (if exists) is assigned into the left hole.

We show a simple example of the action of the algorithm. Consider a sequence of three jobs of sizes 7, 14 and 42, where the first job is in P_1 and the other two are in P_2. Assume that $s = 2$. We have $\beta = \frac{9}{7}$. The three values of $C(J)$ are 9, 9 and 27. The first job can only be assigned to the first machine. The second job can fit completely on the second machine. It is assigned so that the super hole (of size 2) is filled first, and the remainder is assigned to the right hole. The third job occupies the entire super hole and right hole, and a remainder of size 2 still remains, and is assigned to the left hole, to occupy the left hole completely as well (see Figure 4).

Theorem 10 *The algorithm is correct, i.e. it can always assign all the jobs.*

Proof. Assume by contradiction that some job cannot be assigned. Suppose that J is a minimal counterexample in terms of number of jobs, let n be the last job and and denote its size by X (we use X to denote the job n as well). Clearly, X cannot be assigned and every job before X can

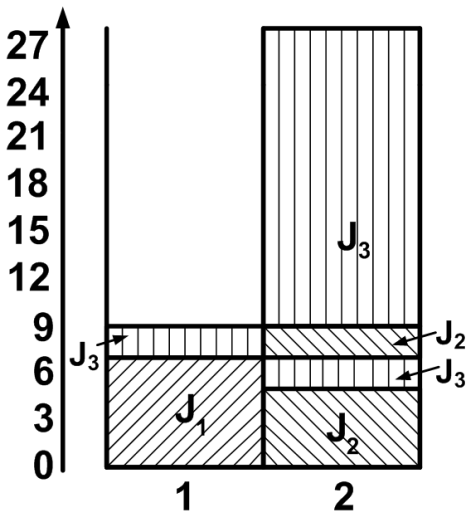

Figure 4: An example for Algorithm 2

be assigned. Note that X is not the only job in the counterexample. This is true since by the definition of the formula that computes the value OPT, in both cases (job 1 is a P_1-job or a P_2-job), it can always be assigned into the super hole.

First suppose that $X \in P_1$. If the job before the last job is a P_1-job as well, then by replacing the two last jobs with one with size $p_{n-1} + p_n$, the value of the OPT remains the same, therefore this modified job also does not fit into the relevant holes (the super hole and the left hole), and we get a counterexample with fewer jobs, which is a contradiction. Therefore, if X is a P_1-job, then it follows that its predecessor is a P_2-job; we denote job $n-1$ and its size by Y.

If job Y does not use any part from the left hole, then omitting it we get a smaller counterexample, which leads to a contradiction. Thus it uses a non-zero part of the left hole. Due to the definition of the algorithm, it follows that it totally uses the super hole and the right hole, i.e. the total running time assigned to the second machine, just after Y is assigned, is $C(J\backslash X)$ and there is no idle time on the second machine. Since $\text{OPT}(J\backslash X) = \max\left\{P_1 - X, \frac{P_1+P_2-X}{s+1}, \frac{P_{\max}}{s}\right\}$, it holds that $C(J\backslash X) \geq \frac{(s+1)^2}{s^2+s+1}\frac{P_1+P_2-X}{s+1}$. It follows that the total size of jobs assigned to the first machine (denoted by C_1) is at most the total size of all P_1-jobs, plus the total size of P_2-jobs that do not fit

on the other machine.

$$C_1 \le P_1 + P_2 - s\frac{(s+1)^2}{s^2+s+1}\frac{P_1+P_2-X}{s+1} = P_1 + P_2 - \frac{s\,(s+1)}{s^2+s+1}\,(P_1+P_2-X)$$
$$= \frac{1}{s^2+s+1}\,(P_1+P_2) + \frac{s\,(s+1)}{s^2+s+1}X = \frac{s+1}{s^2+s+1}\frac{P_1+P_2}{s+1} + \frac{s\,(s+1)}{s^2+s+1}X$$
$$\le \frac{(s+1)^2}{s^2+s+1}\max\left\{\frac{P_1+P_2}{s+1}, X\right\} \le \beta \max\left\{\frac{P_1+P_2}{s+1}, P_1\right\} \le \beta\textsc{opt}.$$

Since X can be assigned to any slot on the first machine, without any risk of overlap, it follows that X can be assigned within the dedicated interval, which is a contradiction.

Next, we consider the case $X \in P_2$. Let U be the last job before X which does not only use the super hole (i.e. if it is a P_1-job, then it uses some non-empty part of the left hole, and if it is a P_2-job, then it uses the some non-empty part of the right hole, and possibly a part of the left hole as well). If U does not exist, then we add a dummy job of size zero as a first job in the sequence.

This means that after job U is assigned, all further P_1-jobs are assigned into the super hole, onto the first machine. We denote these jobs and their total size by $Z \ge 0$. All further P_2-jobs after U and before X are assigned into the super hole onto the second machine. We denote these jobs and their total size by $Y \ge 0$. Since we are interested in the union of jobs of each type, and not in the specific jobs, we may assume that Y and Z are single jobs (possibly of size zero). The behavior of the algorithm on these jobs would be the same not matter how Y and Z are partitioned into jobs.

Consider the moment just after assigning U, and let J_0 be defined as $J_0 = J\backslash\{X, Y, Z\}$. According to the definition of the algorithm, $C(J_0) \ge \frac{(s+1)^2}{s^2+s+1}\frac{P_1+P_2-X-Y-Z}{s+1}$. After assigning U the whole super hole is used. If $U \in P_1$, then U fills the complete super hole on the first machine, and if $U \in P_2$, then it completes this super hole on the second machine. Let the left and right hole at this moment (after assigning U) be denoted simply as L and R (i.e., if T is the job arriving right after U, then $L = L(J_0 \cup T)$ and $R = R(J_0 \cup T)$). Then the total processing time assigned to the first machine is $C(J_0) - L$, and thus the processing time assigned to the second machine at this time is exactly $P_1 + P_2 - X - Y - Z - (C(J_0) - L)$. The jobs which arrive between U is assigned and X arrives are assigned to the super hole at each time. Specifically, the jobs with total size Y are assigned totally to the second machine, and the total processing time assigned to the second machine will become $P_1 + P_2 - X - Y - Z - (C(J_0) - L) + Y = P_1 + P_2 - X - Z - C(J_0) + L$.

The left hole increases only when jobs are assigned into the super hole on the second machine, and does not change if jobs are assigned to the first machine. Therefore, the left hole just before X arrives has size $L + \frac{Y}{s}$. This is the space available for X on the first machine. We need to show that the size of the super hole and right hole on the second machine is at least $X - L - \frac{Y}{s}$ and the remainder of X actually fits into the right hole and the super hole. It suffices to show that if the remained part of X is assigned to the second machine, then the total load of the second machine (not including idle time) is bounded from above by $\frac{(s+1)^2}{s^2+s+1}$-times OPT. We denote this value by C_2 and get:

$$
\begin{aligned}
C_2 &\leq \frac{P_1+P_2-X-Z-C(J_0)+L}{s}+\frac{X-L-Y/s}{s}=\frac{P_1+P_2-Z-C(J_0)-Y/s}{s} \\
&\leq \frac{P_1+P_2}{s}-\frac{(s+1)^2}{s\left(s^2+s+1\right)}\frac{P_1+P_2-X-Y-Z}{s+1}-\frac{Z}{s^2}-\frac{Y}{s^2} \\
&= \frac{s^2+s+1-s-1}{s\left(s^2+s+1\right)}\left(P_1+P_2\right)+\frac{s+1}{s\left(s^2+s+1\right)}X+\left(\frac{s+1}{s^2+s+1}-\frac{1}{s}\right)\frac{Y+Z}{s} \\
&\leq \frac{s\left(s+1\right)}{s^2+s+1}\frac{P_1+P_2}{s+1}+\frac{s+1}{s^2+s+1}\frac{X}{s}\leq\frac{(s+1)^2}{s^2+s+1}\max\left\{\frac{P_1+P_2}{s+1},\frac{X}{s}\right\} \\
&\leq \beta\max\left\{\frac{P_1+P_2}{s+1},\frac{P_{\max}}{s}\right\}\leq\beta\textsc{opt}=C\ ,
\end{aligned}
$$

where the second inequality follows from $s \geq 1$ and the bound on $C(J_0)$, and the third inequality follows from $\frac{s+1}{s^2+s+1} < \frac{1}{s}$. Since no idle time is enforced on the second machine, we get that X can be completely assigned by the algorithm. ■

As in the previous section, we show that idle time is necessary in order to obtain an optimal competitive ratio.

Claim 11 *A deterministic algorithm which does not use idle time has competitive ratio which is strictly larger than* $\beta(s) = 1 + \frac{s}{s^2+s+1}$.

Proof. Consider an algorithm $\mathcal{A}$ which does not use idle time. The sequence starts with the jobs $p_1 = s$ and $p_2 = 1$, where job 1 is a P_2-job and job 2 is a P_1 job. As in the previous section, there are exactly two possible assignments given that idle time cannot be used at any step. Either both machines run the two jobs until time 1, or machine 1 is running both jobs until time $s + 1$. In the second case $\textsc{opt} = 1$, which clearly gives competitive ratio which is much higher than $\beta(s)$. Otherwise, a third job, which is a P_2-job, of size $p_3 = s^2 + s$ arrives. We now have $\textsc{opt} = s + 1$. The machines are balanced, thus the best that can be done now without introducing idle time, is to run it on the faster machine. We get $\mathcal{A} = s + 2$. Since $\frac{s+2}{s+1} > \beta(s)$ for any $s \geq 1$, the claim is proved. ■

Note that the algorithm in this section as well as the algorithm of the previous section are two algorithms of optimal competitive ratio if $s = 1$.

4 Three identical speed machines

In this section, we investigate the case of three hierarchical machines with identical speeds. In this case we can give an optimal algorithm, which is a simple generalization of Algorithm 2.

We again start with lower bounds on the cost of any solution, and in particular on the optimal makespan. In this section $P_{\max}$ denotes the largest size of *any* job. Recall that P_1, P_2 and P_3 are the subsets of jobs that can be assigned to machines in the sets $\{1\}$, $\{1, 2\}$ and $\{1, 2, 3\}$, respectively.

We state and prove the next lower bound for general m. Thus suppose that there are m machines of identical speed.

Lemma 12 *For a given input, we define* $\textsc{lb} = \max\left\{P_1, \frac{P_1+P_2}{2}, ..., \frac{P_1+P_2+...+P_m}{m}, P_{\max}\right\}$. *Then* LB *is a lower bound on the makespan for any solution for this input.*

Proof. The first m bounds hold since all machines have speed 1. Moreover, we can get similar bounds for subsets of the machines set. P_1-jobs are run only on machine 1, the union of P_1-jobs and P_2-jobs is run on the first two machines, and similarly, the union of P_1-jobs, P_2-jobs,..., P_i-jobs is run on the first i machines. The last bound holds since the largest job must be completed. ■

We use the value LB with $m = 3$ for our online algorithm. It is possible to show here as well, that the value LB is not only a lower bound on the makespan of an optimal solution, but is actually equal to this value. We again state and prove this result for any m.

Theorem 13 *Given a set of jobs J, the value* LB *is equal to the makespan of an optimal schedule, and a schedule of this cost can be constructed by a linear time algorithm.*

Proof. We assign jobs to the intervals $[0, \text{LB}]$ of the machines as follows. On each machine, we assign jobs starting from earlier times until later times. We start with machine 1, then machine 2, then machine 3, and so on. We first assign P_1-jobs, then P_2-jobs,..., and finally P_m-jobs. We have at most $m-1$ jobs that were split between machines. If a job is split, then it is scheduled until time LB on one machine, and starting time zero on the next machine. Since $\text{LB} \geq P_{\max}$, there is no overlap caused. Since $\text{LB} \geq P_1$, all P_1-jobs are assigned to the first machine. Since $\text{LB} \geq \frac{P_1+P_2}{2}$, all P_2 jobs are assigned to one of the first two machines, and so on, finally, since $\text{LB} \geq \frac{P_1+P_2+...+P_m}{m}$, there is enough space for all jobs to be assigned. ■

Similarly to before, we replace the notation LB with the notation OPT since we have shown that LB is exactly the cost of an optimal schedule. We continue with a lower bound on the competitive ratio.

We will show that the best competitive ratio for $m = 3$ is $\frac{3}{2}$. We start with proving a lower bound $\frac{2m}{m+1}$ for m machines.

Lemma 14 *Any randomized algorithm $\mathcal{A}$, for preemptive scheduling on m hierarchical machines of equal speeds, has competitive ratio of at least $\frac{2m}{m+1}$.*

Proof. We again apply Yao's method [18]. Our input consists of m sets of jobs; the sum of each set is 1. The jobs of the first set are P_m-jobs, the jobs of the second set are P_{m-1}-jobs, and so on, and the jobs of the last set are all P_1-jobs. The sizes of all jobs are $\frac{1}{m}$. The m inputs (one set, two sets, ..., and finally all m sets) arrive with equal probabilities. Consider a deterministic algorithm $\mathcal{A}$. Let $a_{i,j}$, $i = 1, ..., m$, $j = 1, ..., m+1-i$ be the total sizes of jobs from the i-th set assigned to the j-th machine, respectively. (The jobs from the i-th set are allowed to be assigned only to the first $m+1-i$ machines).

We use $\mathcal{A}_i$ and OPT_i to denote the makespan of $\mathcal{A}$ and an optimal algorithm for the sequence of i sets of jobs. We have $\text{OPT}_1 = \frac{1}{m}$, $\text{OPT}_2 = \frac{2}{m}$,..., $\text{OPT}_i = \frac{i}{m}$, and finally $\text{OPT}_m = \frac{m}{m} = 1$. Therefore, $E(\text{OPT}) = \frac{1}{m}\frac{1+2+...+m}{m} = \frac{m+1}{2m}$. We have $\mathcal{A}_1 \geq a_{1,m}$, $\mathcal{A}_2 \geq a_{1,m-1} + a_{2,m-1}$, $\mathcal{A}_3 \geq a_{1,m-2} + a_{2,m-2} + a_{3,m-2}$, and generally $\mathcal{A}_i \geq a_{1,m+1-i} + a_{2,m+1-i} + ... + a_{i,m+1-i}$. Finally, $\mathcal{A}_m \geq a_{1,1} + a_{2,1} + ... + a_{m,1}$. Thus $E(\mathcal{A}) \geq \frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{i} a_{j,m+1-i} = \frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{m+1-i} a_{i,j} = \frac{1}{m}\sum_{i=1}^{m} 1 = 1$. Let r be the competitive ratio of $\mathcal{A}$. We have $r \geq \frac{E(\mathcal{A})}{E(\text{OPT})}) \geq \frac{2m}{m+1}$. ■

Note that when $m = 3$, the value of the lower bound above is $\frac{3}{2}$.

Let OPT_j be the value of OPT for the sequence of the first j jobs. In the next algorithm again a greedy approach which uses machines with higher indices first, and allocates enough space on these machines, gives the desired result.

Algorithm 3.

0. Let j be the next job to be assigned. Let $i \in \{1,2,3\}$ be such that j is a P_i-job and let C_j be defined as $C_j = \frac{3}{2}\text{OPT}_j$.

1. Let $k \leq i$ be the maximum index for which there exists an available time interval for scheduling on machine k within the time-interval $[0, C_j]$. We use such available time-intervals as long as the job requires additional running time, as follows. First assign a maximum length in of intervals where all machine are idle (either fill these intervals or assign the complete job). Next, use a maximum length of intervals where only one of the other two machines is busy. Finally, use the intervals in which both other machines are busy.

2. If the complete job is scheduled, then go to 0 (and consider the next job). Otherwise go to 1.

Theorem 15 *Algorithm 3 is* 3/2*-competitive and thus it is optimal.*

Proof. We would like prove that all jobs can be scheduled. Suppose by contradiction that the algorithm is not 3/2-competitive, and let I be a counterexample. Without loss of generality, we can assume that all jobs in I are scheduled successfully, except for the last job which cannot be scheduled. We denote the last job as well as its size by X, and its index by n. Then $\text{OPT} = \text{OPT}_n$ and we define $C = C_n$.

We start with proving three lemmas regarding the structure of the counterexample. Afterwards, we examine the last job X carefully and show that it cannot exist, which proves the theorem.

Lemma 16 *Without loss of generality we may assume that for the counterexample* $\text{OPT} = \frac{P_1+P_2+P_3}{3}$ *holds.*

Proof. Otherwise we add tiny P_3-jobs to the example, which arrive right before the last job, and clearly the last job can not be scheduled in this case as well. We add a number of such jobs so that the value of OPT remains exactly the same, but the value $\frac{P_1+P_2+P_3}{3}$ grows and reaches this value. ■

We scale sizes of jobs in the example so that $\text{OPT} = \frac{P_1+P_2+P_3}{3} = 6$. Then it follows that $P_3 \geq 6$. The final value of $C = C_n$ on the complete input is $C = 9$. We get that if job X is assigned to in a valid way (without overlap between its parts), then the largest load exceeds 9.

Lemma 17 *Consider a time during the execution of the algorithm, before the last job has arrived. Suppose that a total processing time of* t *of the* P_3*-jobs is assigned to one of the first two machines. Then there is at least an amount* t *of processing time of* P_3*-jobs that is assigned to the third machine at this point in time during the execution of the algorithm.*

Proof. We prove this claim by induction. Clearly it is true before any jobs arrived. Assume it is true at some time during the execution of the algorithm. The situation may change only upon arrival of a P_3 job, assume this is job j. We have either that machine M_3 is occupied during the complete interval $[0, C_j]$ or that only this machine received parts of the new job. In the first case, the total sum of the jobs is at most $3\text{OPT}_j = 2C_j$, and thus the sum of P_3 jobs does not exceed twice the amount of P_3 jobs on M_3. Otherwise, we use the induction hypothesis. Since at least half of the sum of P_3 jobs was assigned to M_3, and job j is assigned completely to P_3, the percentage of parts of P_3 jobs on M_3 could only increase. ■

Lemma 18 *Consider a time during the execution of the algorithm, before the last job has arrived. Suppose that a total processing time of t of the P_3-jobs and the P_2 jobs is assigned to the first machine. Then there is at least an amount t of processing time of these jobs that is assigned to the second machine at this time during the execution of the algorithm.*

Proof. Consider a partition of the time axis into intervals in which both the first machines are in the same status during the entire interval, that is each machine that is not idle, executes a part of a single job continuously in this interval. We have the following situations.

- Both machines are idle.
- Both machines are running jobs from P_2 or P_3.
- The second machine is running a job from P_2 or P_3, or is idle, and the first machine is running a P_1 job.
- One machine is running a P_2 or P_3 job, and the other machine is idle.

The only situation which can lead to a contradiction to the claim is when we are in the fourth situation, and the idle machine is the second one. However, according to the definition of the algorithm, this is impossible, since this part of job would have been assigned to the second machine before trying to assign it to the first machine. This is true since clearly no part of this job is running on the third machine during this time slot. ■

We continue with the proof of Theorem 15, for which we use the following notations. For any subset $I' \subset I$, the total size of jobs in I' is denoted by $W(I')$, and for any job $Z \in I$, let $P(Z)$ be the total size of jobs up to Z (i.e., Z and all the jobs that arrive before it).

Case 1. $X \in P_3$. Then the total size of the time intervals where all three machines are busy is more than $9-X$ at the time just before the arrival of X. Let S be the set of that jobs which have at least some part which is assigned into a time when all machines are busy, and let Y be the last job in S, and j be its index. At this moment we have $C_j \geq \frac{3}{2}\frac{P(Y)}{3} = \frac{P(Y)}{2}$. Already at this time, the total time where all three machines are busy is more than $9-X$, since no other parts of jobs are assigned during such periods. On the other hand, at this moment there is not a super hole, i.e., there is no time interval when all machines are idle. This means that $C_j \geq 9 - X$ and the total size of jobs which have arrived so far including Y, is at least $P(Y) > (C_j-(9-X))+3(9-X) = \mathbf{2}\,(9 - X)+C_j$. Comparing the inequalities, we get,

$$\begin{aligned} 2C_j \geq P(Y) > 2\,(9 - X) + C_j, \text{ i.e.} \\ C_j > 2\,(9 - X)\,, \text{ and thus} \\ P(Y) > 4\,(9 - X) \end{aligned}$$

and thus the total size of all jobs is $W(I) = X + W(I - X) \geq X + P(Y) > X + 4\,(9 - X) = 36 - 3X \geq 18$, since $X \leq 6$, which leads to a contradiction.

Case 2. $X \in P_2$. Then the total size of time intervals where the first two machines are busy is more than $9 - X$. Now let S be the set of jobs which have a non-empty part which is assigned into a time when the first two machines are busy simultaneously, let Y be the last job in S, and k be its index. Similarly to Case 1, since some part of Y is assigned to a moment when both first two machines are busy, the total load on the first two machines (excluding idle time) is at least

$9 - X + C_j$. Suppose that at this time (when Y is assigned) the sum of the sizes of all parts of P_3-jobs which are assigned to the first two machines is $t \geq 0$, and to the third machine is $q \geq 0$. Then it follows from Lemma 17 that $t \leq q$. We get, that $P(Y) > 9 - X + C_j + q$, and also holds that $C_j \geq \frac{3}{2}\frac{P(Y)}{3} = \frac{P(Y)}{2}$, from which we get,

$$\begin{aligned} 2C_j \geq P(Y) &\geq 9 - X + C_j + q, \text{ i.e.} \\ C_j &\geq 9 - X + q, \text{ and thus} \\ P(Y) &> 2\,(9 - X) + 2q. \end{aligned}$$

As we saw above, $P_3 \geq 6$. If $t + q \leq 6$, then after Y is assigned, there must arrive additional P_3-jobs with total size of at least $6 - t - q$. Thus the total size of all jobs is $W(I) = X + W(I - X) \geq X + P(Y) + (6 - t - q) > X + 2\,(9 - X) + 2q + 6 - t - q = 24 - X + q - t \geq 18$, which leads to a contradiction. In the other case, if $t + q > 6$, then it follows that $q > 3$. Then $W(I) = X + W(I - X) \geq X + P(Y) > X + 2\,(9 - X) + 2q > 18 - X + 6 \geq 18$, which again leads to a contradiction.

Case 3. $X \in P_1$. Then the load of the first machine before X arrives (excluding idle time) is more than $9 - X$, and after X is assigned, the load of this machine is larger than 9 and there is no idle time on it. Let S be the set of that jobs which have at least some part which is assigned to the first machine, before X arrives, let Y be the last job in S, which is not in P_1, and let j be its index. If Y does not exist, we clearly have a load of at more $P_1 \leq 6$ on the first machine. We denote by X' the sum of all P_1 jobs arriving after Y (including X). Since after Y, the only jobs that will be assigned to M_1 are P_1 jobs, we get that after Y is assigned, the load of the first machine is at least $9 - X'$ (excluding idle time), where $X' \leq P_1 \leq 6$. By definition, $Y \in P_2$ or $Y \in P_3$. If $Y \in P_2$, the fact that some part of Y is not assigned to M_2 means that this machine is completely full in the time interval $[0, C_j]$. Thus we get, that the total load of the first two machines (excluding idle time) is at least $9 - X' + C_j$. The proof for this case continues as in Case 2. If $Y \in P_3$, consider the current value of C_j. By the definition of the algorithm it holds that $C_j \geq \frac{P(Y)}{2}$, and since a part of Y is assigned to the first machine, the third machine is completely full in the time interval $[0, C_j]$. Furthermore, since $P_1 \leq 6$, and the load of the first machine will be more than 9 at the end of the algorithm, it follows that at this moment (right after assigning Y) the sum of sizes of P_2 and P_3 jobs being assigned to the first machine is more than 3. From this fact and Lemma 18, it follows that at this moment the load of the second machine (excluding idle time) is also more than 3. Therefore, $P(Y) > 6 + \frac{P(Y)}{2}$ and so $P(Y) > 12$. We get that the load (excluding idle time) of the second machine and third machine together, at this time, is more than 9. Adding the final load of the first machine, that is by our assumption more than 9 (and as mentioned above, this machine does not have idle time after X is assigned) we get that the sum of all job sizes is more than $9 + 3 + 6 = 18$. Therefore this case leads to a contradiction as well. ■

Claim 19 *An deterministic algorithm which does not use idle time has a competitive ratio which is strictly larger than* $\frac{3}{2}$.

Proof. Consider an online algorithm $\mathcal{A}$ which does not use idle time. The sequence starts with very small jobs of total size 1 that are all P_3-jobs. Then it continues with one P_2-job of size 1. At this point, since idle time is not allowed, every machine is busy for some continuous time period, which starts at time zero. (This period may possibly be empty for some machines, but not for all of them.) Denote this time for machine i by d_i. Let r be the competitive ratio. Since the P_2-job

cannot run on machine 3, and after the first batch of jobs $\textsc{opt} = \frac{1}{3}$, we have $r \geq 3d_3 = 3(2-d_1-d_2)$. Next, if $d_1 \geq 1$, a last job arrives. This job is a P_1-job of size 1, which must be assigned to machine 1. At this time $\textsc{opt} = 1$ and thus we have $r \geq 2$. Otherwise, we have $d_1 < 1$, but at least one of machines 1,2 must be busy until at least time 1, due to the P_2-job which cannot be completed before time 1. Thus we have $d_2 > d_1$ and $d_2 \geq 1$. Next, a P_1-job of size $d_2 - d_1$ arrives. After this arrival, the first two machines are balanced. Finally, a P_2-job of size $1 + d_2 - d_1$ arrives. After these jobs, $\textsc{opt} = 1 + d_2 - d_1 > 1$, and $\mathcal{A} = d_2 + 1 + d_2 - d_1$. We therefore have $\frac{1+2d_2-d_1}{1+d_2-d_1} \leq r$ or $1 + \frac{d_2}{1+d_2-d_1} \leq r$. Assume that $r \leq 1.5$. Then we get $d_1 + d_2 \geq 1.5$ (by the condition on d_3) and $d_1 + d_2 \leq 1$, which is a contradiction. ■

5 Conclusion

We designed algorithms of best possible competitive ratios for several preemptive problems with hierarchical machines. Some interesting features were revealed. The competitive ratio for two hierarchical machines of identical speed is $\frac{4}{3}$, the same as the result in reference [5] for two identical machines. This is true (i.e., matches the result in references [10, 17]) even for the case where machines have speeds, but only if the machine of lower hierarchy is faster. However, the result of reference [5] for three identical machines is $\frac{27}{19}$, whereas our tight result for three hierarchical machines of identical speed is $\frac{3}{2}$. Some questions remain open. In particular, the case of m hierarchical machines of identical speeds was not solved in this paper. We conjecture that the competitive ratio in this case is $\frac{2m}{m+1}$, and that this result can be achieved using an algorithm similar to the one in Section 4.

References

[1] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-line load balancing with applications to machine scheduling and virtual circuit routing. *J. ACM*, 44:486–504, 1997.

[2] Y. Azar, J. Naor, and R. Rom. The competitiveness of on-line assignments. *J. Algorithms*, 18:221–237, 1995.

[3] A. Bar-Noy, A. Freund, and J. Naor. On-line load balancing in a hierarchical server topology. *SIAM J. Comput.*, 31:527–549, 2001.

[4] P. Berman, M. Charikar, and M. Karpinski. On-line load balancing for related machines. *J. Algorithms*, 35:108–121, 2000.

[5] B. Chen, A. van Vliet, and G. J. Woeginger. Lower Bounds for Randomized Online Scheduling. *Information Processing Letters*, 51:219–222, 1994.

[6] B. Chen, A. van Vliet, and G. J. Woeginger. An Optimal Algorithm for Preemptive On-line Scheduling. *Operations Research Letters*, 18:127–131, 1995.

[7] P. Crescenzi, G. Gambosi, and P. Penna. On-line algorithms for the channel assignment problem in cellular networks. *Discrete Applied Mathematics*, 137(3):237–266, 2004.

[8] T. Ebenlendr, W. Jawor, and J. Sgall. Preemptive online scheduling: Optimal algorithms for all speeds. In *Proc. of the 14th Annual European Symposium on Algorithms (ESA2006)*, pages 327–339, 2006.

[9] L. Epstein. Optimal Preemptive On-Line Scheduling on Uniform Processors with Non-Decreasing Speed Ratios. *Operations Research Letters*, 29(2):93–98, 2001. Also in STACS 2001.

[10] L. Epstein, J. Noga, S. S. Seiden, J. Sgall, and G. J. Woeginger. Randomized Online Scheduling on Two Uniform Machines. *Journal of Scheduling*, 4(2):71–92, 2001.

[11] L. Epstein and J. Sgall. A Lower Bound for On-Line Scheduling on Uniformly Related Machines. *Operations Research Letters*, 26(1):17–22, 2000.

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

[13] Y.-W. Jiang, Y. He, and C.-M. Tang. Optimal online algorithms for scheduling on two identical machines under a grade of service. *Journal of Zhejiang University SCIENCE A*, 7(3):309–314, 2006.

[14] J. Park, S. Y. Chang, and K. Lee. Online and semi-online scheduling of two machines under a grade of service provision. *Operations Research Letters*, 34(6):692–696, 2006.

[15] S. Seiden. Preemptive Multiprocessor Scheduling with Rejection. *Theoretical Computer Science*, 262(1-2):437–458, 2001.

[16] J. Sgall. A Lower Bound for Randomized On-Line Multiprocessor Scheduling. *Inf. Process. Lett.*, 63(1):51–55, 1997.

[17] J. Wen and D. Du. Preemptive On-Line Scheduling for Two Uniform Processors. *Operations Research Letters*, 23:113–116, 1998.

[18] A. C. C. Yao. Probabilistic computations: towards a unified measure of complexity. In *Proc. 18th Symp. Foundations of Computer Science (FOCS)*, pages 222–227. IEEE, 1977.