Online scheduling of equal-length jobs on parallel machines

Jihuan Ding[1,2], Tomáš Ebenlendr[3], Jiří Sgall[3], and Guochuan Zhang[1]

[1] Dept. of Mathematics, Zhejiang Univ., Hangzhou 310027, China.
{dingjh,zgc}@zju.edu.cn

[2] College of Operations Research and Management Science, Qufu Normal Univ., Rizhao 276826, China. dingjihuan@hotmail.com

[3] Institute of Mathematics, AS CR, Žitná 25, CZ-11567 Praha 1, Czech Republic.
{ebik,sgall}@math.cas.cz

Abstract. We study on-line scheduling of equal-length jobs on parallel machines. Our main result is an algorithm with competitive ratio decreasing to $e/(e-1) \approx 1.58$ as the number of machine increases. For $m \geq 3$, this is the first algorithm better than 2-competitive greedy algorithm.

Our algorithm has an additional property called *immediate decision*: at each time, it is immediately decided for each newly released job if it will be scheduled, and if so, then also the time interval and machine where it is scheduled is fixed and cannot be changed later. We show that for two machines, no deterministic algorithm with immediate decision is better than 1.8-competitive; this lower bound shows that our algorithm is optimal for $m = 2$ in this restricted model. We give some additional lower bounds for algorithms with immediate decision.

1 Introduction

We study a problem in the area of real-time scheduling. We are given an input sequence of jobs with equal processing times p. Each job has its release time and deadline, specifying the time window in which it needs to be scheduled; all the parameters are integers. The desired output is a nonpreemptive schedule on m identical machines, possibly only for a subset of the jobs on input. Each scheduled job must be executed between its release time and deadline, and different jobs cannot overlap if they are scheduled on the same machine. The term "nonpreemptive" means that each job must be executed without interruptions, in a contiguous interval of length p. The objective is to maximize the number of completed jobs.

In the *online version*, each job is released at its release time, and its deadline is revealed at this time, too. The number of jobs and future release times are unknown. At each time step when some machine is available (it just completed a job or was idle), we have to decide whether to start some jobs, and if so, to choose which ones, based only on the information about the jobs released so far.

An online algorithm is called *c-competitive* if on every input instance it schedules at least $1/c$ as many jobs as the optimum schedule.

This type of problems is extensively studied. In addition to the results mentioned below, there is a vast literature on real-time scheduling of jobs with arbitrary (not equal) processing times, weights, and so on. In these more general variants, often preemption is necessary to achieve reasonable competitive ratio, and still the ratio depends on various parameters.

For our problem with equal jobs and non-preemptive scheduling, it is known that a greedy algorithm performs reasonably well, namely it is 2-competitive for any number of machines. Note that in this type of problems it is usual and natural that more machines make performance better, as it is possible to keep in some time interval some fraction of the total capacity of the machines available for later arriving jobs. Nevertheless, for this very basic problem, no non-trivial algorithm was known for $m \geq 3$ before this work.

1.1 Previous results

Most of the previous results deal only with the case of a single machine. In this case, it is known that a greedy algorithm is 2-competitive for this problem [2], and that this is optimal for deterministic algorithms [5]. There exists a 5/3-competitive randomized algorithm [3] and no better than 4/3-competitive randomized algorithm exists [5].

For two machines, independently Goldwasser and Pedigo [7] and Ding and Zhang [4] designed 1.5-competitive deterministic algorithms, and this competitive ratio is optimal. Their algorithms and analysis are fairly complicated, and it seems that this is necessary, as very precise timing is needed to obtain the optimal competitive ratio.

For more machines, 2-competitive greedy algorithm is known, which is an easy generalization of the single machine result. To our best knowledge, no better algorithms for $m \geq 3$, deterministic or randomized, were known prior to our work. A lower bound of 6/5 is known [4], using the same idea as the lower bound of 4/3 for randomized algorithms on a single machine. For deterministic algorithms, it actually gives a slightly higher lower bound, but it still approaches 6/5 for large m.

In the offline case, it is known that the problem is polynomially solvable for any fixed number of machines [1]. The complexity with the number of machines as the part of the input is still open.

1.2 Our results

Our main result is algorithm BestFit with competitive ratio decreasing to $e/(e-1) \approx 1.582$ for large m. The exact ratio for m machines is

$$R_m = \frac{1}{1 - \left(\frac{m}{m+1}\right)^m}.$$

For $m = 2$, this competitive ratio is 1.8. Compared to the known algorithms for $m = 2$, both the algorithm and its analysis are very simple; the proof uses a charging scheme extending the analysis of the greedy algorithm for $m = 1$.

Immediate decision. Our algorithm has an additional property introduced in [4] which is called *immediate decision*: At each time t, it is immediately decided for each newly released job j (i.e., with $r_j = t$) if it will be scheduled, and if so, then also the time interval and machine where it is scheduled are fixed and cannot be changed later.

The second main contribution of our paper is a study of the power of the restricted model with immediate decision. We give several lower bounds showing that this restriction is quite severe.

For $m = 2$, we show that no deterministic algorithm with immediate decision is better than 1.8-competitive. Thus our algorithm BestFit is optimal for $m = 2$ in this restricted model. This shows that immediate decision is a strong requirement, as for $m = 2$ the optimal competitive ratio in the unrestricted model is 1.5 smaller.

We also give a simple lower bound of $4/3$ for arbitrary m with immediate decision, which holds even for randomized algorithms.

Finally, we give some lower bounds for jobs with small processing times. With immediate decision for $m = 2$, we prove that deterministic algorithms are at least 1.6-competitive for $p = 1$ and $5/3$-competitive for $p = 2$. The previously mentioned lower bound of 1.8 holds for any $p \geq 3$.

Scheduling with immediate decision is somewhat similar to the model of jobs arriving one by one, also called list scheduling, with an additional restriction that the jobs are ordered according to their release times. The models are the same for instances with all the release times distinct. In a model with continuous time, it is possible to slightly perturb the release times and deadlines, and the models are thus equivalent. For large p, our model with discrete time approaches the continuous time model and thus also the model of jobs arriving one by one consistently with the release times.

The requirement of immediate decision should also be compared with the notion of *immediate notification* introduced in [6]. Immediate notification requires that upon release of each job, we immediately decide (i.e., notify the user) if the job will be completed or not; however, its exact timing and the choice of a machine may depend on subsequent jobs. It seems to be the case that in all cases studied so far, it is possible to provide immediate notification without changing the performance of the algorithms. In contrast, our results show that this is no longer the case when the requirement is strengthened from immediate notification to immediate decision. Then the competitive ratio must increase at least in the case of two machines.

1.3 Preliminaries

Each job is described by a pair of numbers (r_j, d_j) denoting its release time and deadline, respectively. All the release times, deadlines and starting times of jobs are assumed integral, as is usual in scheduling literature. Saying that a job is running at time t is equivalent to saying that it is running in time interval $[t, t+1)$. (However, we note that our algorithm and proofs work also for continuous time and non-integral r_j, d_j. The choice of the discrete model is mainly a matter of taste and tradition in the literature.)

Given a particular schedule, S_j denotes the starting time of job j. Each scheduled job needs to be scheduled so that $r_j \leq S_j \leq d_j - p$. Let $C(M_i)$ denote the completion time of machine M_i, i.e., the first time t such that no job is scheduled to run on machine M_i at or after t.

In the online problem, each job is released at time r_j. However, in the more complicated lower bounds using adversary arguments, it is often convenient to say that at time t, based on the previous actions of the algorithm, we release some set of jobs, possibly including (r_j, d_j) with $r_j > t$. This should be interpreted so that we commit to releasing such job(s) and we actually reveal them to the algorithm only at time r_j.

2 The algorithm

We now present our algorithm. It simply tries to schedule each job without any idle time on the most full machine where it can be completed by its deadline. So, for example, a sequence of jobs with huge deadlines is scheduled on a single machine, leaving all other machines available for future jobs.

Algorithm BestFit.

At each time t, as an invariant, each machine M_i is committed to execute jobs from t until its completion time $C(M_i)$ with no idle time inserted. A machine M_i is called *feasible* for job j if $C(M_i) \leq d_j - p$.

At time t, consider the newly released jobs one by one. If no feasible machine for job j exists, the job is rejected. Otherwise j is scheduled on a machine with largest $C(M_i)$, among all the feasible machines; job j is then scheduled to start execution at time $C(M_i)$.

For the analysis it is important to keep track of the order in which the jobs have been considered by the algorithm. We refer to this moment as to the *decision*. So, we have a linear ordering of jobs given by which job was decided earlier, which extends the ordering by release times. Also, a schedule at the time when a job was decided refers to the schedule at its release time, just before BestFit processed this job and fixed its schedule. Thus this schedule contains exactly all the jobs decided before the current one, possibly including some of the jobs with the same release time.

Before we present the proof for general m, we sketch the simplest case $m = 2$.

To prove that the algorithm is 1.8-competitive, we present a charging scheme. Consider a schedule A generated by the algorithm and an arbitrary (offline, adversarial) schedule Z. The charging scheme assigns each job in Z to some jobs in A. The assignment is allowed to be fractional, i.e., some fractions of a job in Z may be assigned to different jobs in A, assuming the total of these fractions is 1. We then show that each job in A is charged at most 1.8, which completes the proof.

The charging scheme is defined as follows: Let t be the starting time of some job i in Z. If at time t, both machines in A are idle, then job i is charged 1 to itself in A. If at t one machine is busy in A, then i is charged 0.4 to the job running on that machine in A and 0.6 to itself in A. In both previous cases, charging to i is well-defined: Job i has to be scheduled in A, since there exists a feasible machine for i, namely the machine which is running no job at time t in the final schedule, and thus was also feasible for i at when BestFit decided i. Finally, if both machines are busy at t in A, then i is charged to the two jobs running in A at time t so that 0.4 is charged to the job that was decided first by the algorithm and 0.6 is charged to the other job. If job j is scheduled on a machine with the larger completion time at the time when BestFit decided j, then at any time when it is running in A j is the first job decided of the two currently running; thus j is charged at most $2 * 0.4 + 1 = 1.8$ from the two jobs started during its execution in Z and from itself. If job j is scheduled on the other machine in A, then the first machine is busy at all times when j can start in Z and thus j is charged at most $2 * 0.6 + 0.6 = 1.8$.

The instance showing that BestFit is no better than 1.8-competitive for $m = 2$ and $p \geq 3$ consists of these jobs: 3 jobs $(0, 6p+2)$, scheduled by BestFit on the first machine from time 0 to $3p$; 2 jobs $(1, 3p+2)$, scheduled on the second machine from time 1 to $2p+1$; and finally 4 jobs $(2, 2p+2)$ that are rejected. The optimum schedules all 9 jobs (essentially in the reverse order).

Now we are ready to present the full proof for a general m.

Theorem 1. *The competitive ratio of the algorithm* BestFit *is*

$$R_m = \frac{1}{1 - \left(\frac{m}{m+1}\right)^m}.$$

For $m \to \infty$, R_m decreases to $e/(e-1) \approx 1.582$; $R_2 = 1.8$, and $R_3 = 64/37 \approx 1.730$.

Proof. Let

$$Y_k = (m+1)^{k-1} m^{m-k} \quad \text{and} \quad X_k = \frac{Y_k}{(m+1)^m - m^m}.$$

Note that Y_k and X_k both increase with k and $X_1 + \cdots + X_m = 1$.

The upper bound is proved by a charging scheme. Consider a schedule A generated by the algorithm and an arbitrary schedule Z. Let i be a job in schedule Z and t its starting time in Z. Let $j_1, \ldots, j_k$ be all the jobs running (or just started) at time t in the schedule A, in the order they were decided by BestFit. We charge the total of 1 of job i in Z to jobs $j_1, \ldots, j_k$, and i in the schedule A so that we charge the amount of X_α to each j_α, $\alpha = 1, \ldots, k$ and the amount of $X_{k+1} + \cdots + X_m$ to i. Charging to i is well-defined: If $k < m$ then i is scheduled in A, as there exists a feasible machine for it at the time when it is considered, namely the machine which is running no job at time t in A. Otherwise, if $k = m$, the amount to be charged to i is 0. We observe that the total charged is always equal to $X_1 + \cdots + X_m = 1$, and thus the definition of the charging scheme is sound.

To prove that the algorithm is R_m-competitive, it remains to show that each job in A is charged at most the amount of R_m; then the competitive ratio R_m follows by summing over all jobs. Suppose that some job j is scheduled on a machine with the kth largest $C(M_i)$ (since the first $k-1$ machines are not feasible). Then at any time from r_j up to (and including) $d_j - p$ at least $k-1$ machines are executing jobs decided before j. Thus j may be charged at most $X_k + \cdots + X_m$ from itself. Furthermore, at any time j is running in A, it is at most kth job decided by BestFit. It follows that the charge to j from each of the at most m other jobs started in Z while A is running j is at most X_i for some $i \leq k$, and this is at most X_k due to monotonicity of X_i's. The total charge to j is at most $mX_k + X_k + \cdots + X_m$. We claim that $mX_k + X_k + \cdots + X_m = R_m$, for any k: We have $mX_k + X_k = mX_{k+1}$ for any $k < m$, thus all the values are equal, and the last value is $mX_m + X_m = R_m$. This completes the proof of the desired competitive ratio.

An instance showing that BestFit is no better than R_m-competitive is this: Let $p > m$. First we present Y_m jobs $(0, 2Y_m p + m)$. Then, for each $k = m-1, m-2, \ldots, 1$, we have Y_k jobs $(m-k, Y_{k+1}p + m)$. Finally, mY_1 jobs $(m, Y_1 p + m)$ arrive.

It can be verified that the algorithm first schedules all Y_m jobs with $r_j = 0$ on one machine, then all Y_{m-1} jobs with $r_j = 1$ on the next machine, and so on, and it rejects the mY_1 jobs with $r_j = m$. On the other hand, the optimal solution is idle until time m, then schedules the mY_1 jobs with $r_j = m$, then schedules the Y_1 jobs with $r_j = m-1$, and so on, completing all the jobs. The ratio is equal to R_m. □

3 The lower bounds for immediate decision

If the instance starts by a modest number of jobs with a very large deadline, BestFit schedules them on one machine close to time 0. Apparently this gives a big advantage to the adversary, who can then focus on the region where these jobs are scheduled. It would seem reasonable to try to improve the performance by spreading these jobs somehow uniformly over the whole feasible time interval. However, perhaps surprisingly, for $m = 2$, we can prove that no such strategy helps and in fact BestFit is an optimal algorithm with immediate decision.

In the lower bound proof for $m = 2$, we try to force the algorithm to schedule the first jobs so that both processors are busy at some time steps. Then we release pairs of tight jobs that must be rejected. Typically, we create two such problematic times after scheduling of 5 jobs. This results in two additional pairs, i.e., a total of 9 jobs of which the algorithm schedules only 5. The optimum always schedules all the jobs. The first jobs have a very long feasible intervals, so in an optimal schedule they can always be moved so that they do not conflict with any other jobs. The optimal schedule of tight jobs is determined, so it remains to verify in each case that the remaining jobs can be scheduled without conflicts with the tight jobs. The number of cases is relatively high also due to the fact that some of the jobs may be rejected.

Theorem 2. *Let A be a deterministic algorithm with immediate decision for $m = 2$ machines and $p \geq 4$. Then A is no better than 1.8-competitive.*

Proof. We start by releasing three jobs $(0, 100p)$. Now we wait for the algorithm to decide the schedule of these jobs. Note that this means that time advances to 1 and we cannot release more jobs with $r_j = 0$. We note that optimal schedule can always schedule these jobs so that they do not overlap with a feasible time interval of any other job. Thus in the rest of the proof it is sufficient to verify that the other jobs can be scheduled. First we analyze the case of all three jobs accepted. We renumber them so that their start times are $S_1 \leq S_2 \leq S_3$.

1. There exist two times $t_2 > t_1 \geq p$, where two jobs are running. This includes the case of two jobs overlapping for more than one time step.
   [In this case the algorithm essentially gives up.]
   We release two tight jobs $(t_1 - p + 1, t_1 + 1)$ and two tight jobs $(t_2, t_2 + p)$. This is possible since $t_1 - p + 1 \geq 1$. The algorithm has to reject all four new jobs. The optimal schedule schedules all seven jobs, since $t_2 \geq t_1 + 1$. Thus the competitive ratio is no better than 7/3.
2. We have $S_3 - S_2 \leq 2p - 1$ and $S_2 \geq p$. Since the previous case does not hold, we also have $S_3 - S_2 \geq p - 1$.
   [This case matches the behavior of BestFit.]
   We release two jobs, 4 and 5, with $(r_j, d_j) = (S_3 - p - 2, S_3 + 2p - 1)$. This is possible as $S_3 - p - 2 \geq S_2 - 3 \geq 1$.
   If any of jobs 4 and 5 gets scheduled, then it overlaps the schedule of jobs 2 or 3.
   Depending on the schedule of jobs 4 and 5, we schedule two pairs of tight jobs, choosing from pairs with (r_j, d_j) equal to $(S_3 - p - 1, S_3 - 1)$, $(S_3 - 1, S_3 + p - 1)$, or $(S_3 + p - 1, S_3 + 2p - 1)$. We choose the pairs as follows.
   (a) If the algorithm scheduled both jobs 4 and 5, it can be verified that two of the slots for the tight jobs contain a time when both machines are busy. Release these two pairs.
   (b) If the algorithm schedules only one of jobs 4 and 5, there will be one slot for the tight jobs containing a time when both machines are busy. Release this pair and an arbitrary additional pair.
   (c) If the algorithm rejects both jobs 4 and 5, release arbitrary two pairs.

   In every case it can be checked that the algorithm schedules at most 5 of the total 9 jobs. Also, the optimum schedules all jobs, as 4 and 5 can be scheduled in the unused slot for tight jobs.
3. We have $S_3 - S_2 \geq 2p$ and $S_2 \geq p$.
   [This case is most interesting, as it kills attempts at algorithms that try to spread the jobs with long feasible intervals.]

We release job 4 with $(r_4, d_4) = (S_2 - 2, S_2 + 2p - 1)$. Depending on its schedule, we release two tight jobs 5 and 6 as follows: if $S_4 \le S_2 - 1$, then they are $(S_2 - 1, S_2 + p - 1)$, otherwise $(S_2 + p - 1, S_2 + 2p - 1)$ (including the case when job 4 is rejected). The algorithm can schedule only one job of 4, 5, and 6.

We continue similarly by job 7 with $(r_7, d_7) = (S_3 - 2, S_3 + 2p - 1)$. Depending on its schedule, we release two tight jobs 8 and 9 as follows: if $S_7 \le S_3 - 1$, then they are $(S_3 - 1, S_3 + p - 1)$, otherwise $(S_3 + p - 1, S_3 + 2p - 1)$ (including the case when job 7 is rejected). The algorithm can schedule only one job of 7, 8, and 9.

Note that $r_7 \ge d_4 - 1$. This guarantees that the optimum can schedule both jobs 4 and 7, no matter which tight jobs are released.

Overall, the competitive ratio is at least 9/5.

4. It remains to handle the case when $S_2 < p$.

   [It is not very smart if the algorithm overlaps the jobs at the very beginning, but it seems that unfortunately we need to handle this case separately, revisiting some of the previous cases.]

   We release two tight jobs 4 and 5 with a slot $(1, p + 1)$, which get rejected, and two jobs 6 and 7 with $(r_j, d_j) = (1, 100p)$. Out of the four jobs 1, 2, 4, and 5, the algorithm schedules only two.

   If there is a time $t \ge p + 1$ when both machines are busy, we conclude the proof by another pair of tight jobs. Similarly, if one of the jobs 6 and 7 is rejected, release a pair of tight jobs with $r_j \ge p + 1$ overlapping job 3.

   Otherwise, renumber jobs 3, 6, and 7 to 1, 2, and 3, subtract t from all times, and iterate the case of three released and scheduled jobs once more. Since the three jobs do not overlap, we end up in case 2 or 3. It can be easily verified that the optimum can schedule all the jobs from the first and second iterations together. The ratio is at least 9/5 for the second iteration, so including the 4 jobs in the first iteration, the overall ratio is strictly worse than 9/5.

Now it remains to analyze the case that algorithm rejected some of the first three jobs. If the algorithm rejects at least two jobs then the ratio is at least 3.

If the algorithm rejects one job, renumber the jobs so that $S_1 \le S_2$ and 3 is rejected. If $S_2 < 3$, release two tight jobs $(1, p + 1)$, and the ratio is 5/2. If $S_2 \ge 3$, we release job 4 with $(r_4, d_4) = (S_2 - 2, S_2 + 2p - 1)$. Depending on its schedule, we release two tight jobs 5 and 6 as follows: if $S_4 \le S_2 - 1$, then they are $(S_2 - 1, S_2 + p - 1)$, otherwise $(S_2 + p - 1, S_2 + 2p - 1)$ (including the case when job 4 is rejected). The algorithm can schedule only one job of 4, 5, and 6. Thus it schedules only 3 jobs, while the optimum schedules all 6. □

For an arbitrary m, we prove a lower bound of 4/3 even for randomized algorithms. This is an easy adaptation of the standard bounds for a single machine.

Theorem 3. *Let A be an algorithm (possibly randomized) with immediate decision for m machines and $p \ge 2$. Then A is no better than 4/3-competitive.*

Proof. Release m jobs $(0, 2p + 1)$. If the expected number of jobs that start at time 0 or 1 is at least $m/2$, release m jobs $(1, p + 1)$. Otherwise, release m jobs $(p, 2p)$. In both cases, the algorithm schedules at most $3m/2$ jobs, while the optimum is $2m$. □

4 Jobs with a small length

The previous lower bounds hold only for sufficiently large p. It is an interesting question what happens for smaller values of p, in particular for unit jobs, i.e., $p = 1$. Note that without the requirement of immediate decision, for $p = 1$, it is easy to generate an optimal schedule online for any number of machines: Always schedule the m jobs with the smallest deadlines among those which are still feasible.

With immediate decision for $m = 2$, we prove that deterministic algorithms are at least 1.6-competitive for $p = 1$ and 5/3-competitive for $p = 2$. Finally, for $p = 3$, we revisit the proof of Theorem 2 and show that by handling one case more carefully we obtain the same lower bound of 1.8 as for $p \geq 4$.

For the case of unit jobs, we need an auxiliary lemma.

Lemma 1. *Suppose we have a discrete interval $I = [1..N]$, non-negative integral weights w_i, $i \in I$, such that $\sum_{i \in I} w_i = N/2$, and an arbitrary number $k \leq n$. Then there exist three disjoint subintervals I_1, I_2, I_3 of I, each with total weight $\sum_{i \in I_\alpha} w_i \geq |I_\alpha|/2$, with total length $|I_1| + |I_2| + |I_3| = k$, such that in addition I_1 contains 1 and I_3 contains N.*

Proof. If there exists an interval I_2 of length k with weight at least $k/2$, we set $I_1 = I_3 = \emptyset$ and we are done. Otherwise by averaging there exists two intervals $[1..i]$ and $[i'..N]$, with total length k and total weight at least $k/2$. Thus one of these intervals has weight at least half of its length. W.l.o.g. assume that the interval $[1..i]$ has weight at least $i/2$. Since the interval $[1..k]$ has weight less than $k/2$, it follows that for some j, $i \leq j < k$, the interval $[1..j]$ has weight exactly $j/2$. Then we apply the lemma recursively on the interval $I - [1..j]$ and with $k - j$ in place of k. We merge the interval $[1..j]$ with I_1 resulting from the recursive application. □

Theorem 4. *Let A be a deterministic algorithm with immediate decision for $m = 2$ and $p = 1$. Then A is no better than 1.6-competitive.*

Proof. Let M be large. In the first phase, we release $2M$ jobs $(0, 4M)$. If some job is not scheduled, we assign it to any time arbitrarily but so that at no time two jobs are scheduled or assigned.

We apply Lemma 1 to the interval $[0..4M - 1]$ and $k = 3M$ with the weights equal to the number of jobs scheduled at or assigned to the given time. It follows that there exist three intervals $[t_i, t'_i)$, $i = 1, 2, 3$, such that denoting $T_i = t'_i - t_i$ their lengths, their total length is $T_1 + T_2 + T_3 = 3M$ and each interval contains at least $T_i/2$ jobs. Denote n_i the number of jobs scheduled in $[t_i, t'_i)$ or assigned to a time in $[t_i, t'_i)$.

Next, in the second phase, for each interval $i = 1, 2, 3$, we release $\max\{3T_i/2 - n_i - 2, 0\}$ jobs $(t_i + 1, t'_i)$. Note that there are at most T_i new jobs. Assign all unscheduled jobs and reassign all the jobs from the first phase assigned originally to this interval to times in this interval arbitrarily but so that at no time two jobs are scheduled or assigned. (It may be necessary to reassign some job from the first phase, since the algorithm schedules a job to the same time.)

As there are total $3T_i/2-2$ jobs in the interval $[t_i, t'_i)$ from the first two phases, there are at least $T_i/2 - 2$ times when two jobs are scheduled or assigned. For exactly $T_i/2 - 4$ of such times $t > t_i + 1$ release two tight jobs $(t, t + 1)$ in the third phase. (We have to omit the first two times in each interval, since after observing the assignment of the jobs from the second phase released at $t_i + 1$ we cannot use them.)

The optimum can schedule all the jobs: The tight jobs take less than $T_i/2$ times in each interval, and the at most T_i jobs from the second phase can be scheduled in the remaining available $T_i/2$ times in the interval. The jobs from the first phase exactly fit outside the three chosen intervals.

In the first two phases, there are at most $2M + T_1 + T_2 + T_3 = 5M$ jobs which may be scheduled by the algorithm. In the third phase, there are exactly $2(T_1/2 + T_2/2 + T_3/2) - 24 = 3M - 24$ jobs, all rejected by the algorithm. Thus the competitive ratio is at least $(8M - 24)/5M$, which is arbitrarily close to 1.6 for large M. ◻

Theorem 5. *Let A be a deterministic algorithm with immediate decision for $m = 2$ machines and $p = 2$. Then A is no better than $5/3$-competitive.*

Proof. First, we release M jobs $(0, 100M)$, for a large M. Next we search the current schedule from beginning to end for times where both machines are busy. We start at time 1, as we are not allowed to release other jobs with $r_j = 0$. Whenever we find such time t, we release two jobs $(t, t + 2)$. We group them with the two jobs scheduled at t and also with the last job scheduled before t, if it is not included in the previous group. So we have groups of at most 5 jobs, such that the algorithm rejects two jobs in each group. We continue the search at time $t + 2$.

Now no two remaining jobs (i.e., those not in any group) overlap in the schedule. Moreover for each remaining job j, we released no tight job overlapping with the interval $[S_j, S_j + 4)$.

Next, we examine the remaining jobs again from beginning of the schedule to end. We skip the first job as it may be scheduled at time 0 or 1 (this is why we need M large). Suppose that we are examining job j.

If there is no remaining job scheduled in time $[S_j, S_j + 4)$, we release two jobs $(S_j - 1, S_j + 4)$. Then, if one of them is scheduled and starts at time $S_j + 1$ or earlier, we release two tight jobs $(S_j, S_j + 2)$; otherwise we release two tight jobs $(S_j + p, S_j + p + 2)$. We create a group of j and these four jobs. The algorithm schedules at most 3 out of this group of 5 jobs.

Otherwise, there is some job j' with $S_{j'} \le S_j + 3$. We release only one job $(S_j - 1, S_j + 4)$. Again we continue with two tight jobs $(S_j, S_j + 2)$ or $(S_j + 2, S_j + 4)$, so that the algorithm can schedule only one of the three new jobs. Again j, j' and these three jobs form a new group. We skip j' and continue examining the following jobs.

Now we have all jobs but the first one in groups of 5. Each group has at most 3 jobs scheduled by A. In an optimal schedule, all tight jobs can be scheduled, then all jobs with $d_j = r_j + 5$ can be scheduled with $S_j = r_j + 1$ or $S_j = r_j + 3$. Finally, there is plenty of room for the remaining jobs (released at time 0). Thus any optimal schedule schedules all the jobs.

For M large enough, this proves that no algorithm is $(5/3 - \varepsilon)$-competitive for $\varepsilon > 0$. □

Theorem 6. *Let A be a deterministic algorithm with immediate decision for $m = 2$ machines and $p \geq 3$. Then A is no better than 1.8-competitive.*

Proof. We need to revisit the proof for $p \geq 4$. All cases but 2 can be analyzed exactly same way. Case 2 must be handled more carefully. If the algorithm does not reject any job among 1, 2, 3, and $S_2 = 3$, $S_3 = 5$ we cannot follow the case 2 as we would need to release a job with $r_4 = 0$. In this case we release six jobs $(1, 10)$. Algorithm can accept only two of them, so overall it schedules only 5 out of 9 jobs. □

5 Conclusions, open problems, acknowledgments

The main open problem in this area remains to design optimal or at least good algorithms for the unrestricted model and $m > 2$. Despite the good progress we have been able to achieve using algorithms with immediate decision, one would expect that also for $m > 2$, the best algorithms will use the flexibility of the unrestricted model. However, no such algorithms are known.

We know that for $m = 2$, immediate decision increases the optimal competitive ratio and our new algorithm is optimal in the restricted model. For $m \geq 3$ we would expect the same to be true, but we have no lower bounds. It would be also interesting to prove more in the case of unit jobs, either for $m = 2$ or for larger m.

Finally, virtually nothing is known about randomized algorithms for $m \geq 2$.

We are grateful to anonymous referees for many comments that helped us to improve the presentation of this paper. T. Ebenlendr and J. Sgall were partially supported by Institutional Research Plan No. AV0Z10190503, by Inst. for Theor. Comp. Sci., Prague (project 1M0545 of MŠMT ČR), and grant 201/05/0124 of GA ČR. G. Zhang was partially supported by NSFC (60573020).

References

1. P. Baptiste, P. Brucker, S. Knust, and V. Timkovsky: *Ten notes on equal-execution-time scheduling.* 4OR **2** (2004), pp. 111-127.

2. S. K. Baruah, J. Haritsa, and N. Sharma: *On-line scheduling to maximize task completions*. J. Comb. Math. Comb. Comput., **39** (2001), pp. 65–78. A preliminary version appeared in Proc. 15th Real-Time Systems Symp., IEEE, 1994, pp. 228–236.
3. M. Chrobak, W. Jawor, J. Sgall, and T. Tichý: *Online scheduling of equal-length jobs: Randomization and restarts help*. In Proc. 31st International Colloquium on Automata, Languages, and Programming (ICALP), vol. 3142 of Lecture Notes in Comput. Sci., Springer, 2004, pp. 358–370. To appear in SIAM J. Comput.
4. J. Ding and G. Zhang: *Online scheduling with hard deadlines on parallel machines*. In Proc. 2nd International Conf. on Algorithmic Aspects in Information and Management (AAIM), vol. 4041 of Lecture Notes in Comput. Sci., Springer, 2006, pp. 32–42.
5. S. A. Goldman, J. Parwatikar, and S. Suri: *Online scheduling with hard deadlines*. J. Algorithms, **34** (2000), pp. 370–389.
6. M. H. Goldwasser and B. Kerbikov: *Admission control with immediate notification*. J. Sched., **6** (2003), pp. 269–285.
7. M. H. Goldwasser and M. Pedigo: *Online, non-preemptive scheduling of equal-length jobs on two identical machines*. In Proc. 10th Scandinavian Workshop on Algorithm Theory (SWAT), vol. 4059 of Lecture Notes in Comput. Sci., Springer, 2006, pp. 113–123.