Optimal Preemptive Semi-Online Scheduling to Minimize Makespan on Two Related Machines

Leah Epstein[1⋆] and Lene M. Favrholdt[2⋆⋆]

[1] School of Computer Science, The Interdisciplinary Center, Herzliya, Israel. lea@idc.ac.il.

[2] Department of Mathematics and Computer Science, University of Southern Denmark, Odense, Denmark. lenem@imada.sdu.dk

keywords: Semi-Online, Scheduling.

Abstract. We study a preemptive semi-online scheduling problem. Jobs with non-increasing sizes arrive one by one to be scheduled on two uniformly related machines. We analyze the algorithms as a function of the speed ratio ($q \geq 1$) between the two machines. We design algorithms of optimal competitive ratio for all values of q, and show that for $q > 2$, idle time needs to be introduced. This is the first preemptive scheduling problem over list, where idle time is provably required.

1 Introduction

The problem: We consider semi-online scheduling on two uniformly related machines. The aim is to minimize the *makespan*, i.e., the maximum completion time on any machine. Jobs arrive one by one, and each job has to be assigned before the next job arrives. Each job has a size which is the time it takes to complete it on a machine with unit speed. The jobs arrive in an order of non-increasing sizes, hence this scheduling model is not on-line but semi-online. Each job is assigned without any knowledge of future jobs (except for a bound on their size that follows from the size of the current job). The scheduling model, where each job is assigned before

⋆ Research supported in part by the Israel Science Foundation, (grant No. 250/01-1)

⋆⋆ Supported in part by the Danish Natural Science Research Council (SNF) and in part by the Future and Emerging Technologies programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

the arrival of the next job, is called "Scheduling over list" or "Scheduling jobs one by one" (see [16]).

Preemption is allowed, hence each job may be cut into a few pieces. These pieces have to be assigned to possibly different machines, in non-overlapping time slots. (Non-preemptive algorithms are not allowed to cut the job and have to assign it continuously to one machine.) It is not required that a machine be busy all the time until the last piece of a job running on it is completed. A time where a machine is not running a job, while it did not yet complete all pieces of jobs assigned to it (i.e., it is assigned to run a job at a later time), is called *idle time*. Even though it is not common to design algorithms that use idle time, it is allowed and can be useful in order to keep room for later arriving jobs.

Preliminaries: In the model of uniformly related machines, machine i has a speed s_i. The processing time of a job (or a part of a job) of size p, on machine i is p/s_i. However, since we consider the case of two machines, the important parameter is the speed ratio $q \geq 1$ between the two machines. Hence, without loss of generality, we assume that the faster machine has speed 1, and the other machine has speed $1/q$.

The processing time of a job of size p (p on the fast machine and pq on the slow one) is also called the *load* of the job. The load of a machine is the sum of the loads of the jobs scheduled on it, plus the idle time of this machine.

The quality of a semi-online algorithm, similarly to on-line algorithms, is measured by the *competitive ratio* that is the worst case ratio between the cost (the makespan, in our case) of the semi-online algorithm and the cost of an optimal off-line algorithm which knows the whole sequence in advance. Naturally, the optimal off-line algorithm is also allowed to use preemption.

Previous work: Seiden, Sgall and Woeginger [14] study semi-online preemptive scheduling on *identical machines*. They show that the most difficult case among non-increasing sequences is a sequence of unit size jobs. They give a complete solution, where the competitive ratio is a function of the number m of machines.

There are several tight results for preemptive *on-line* scheduling, in all cases the lower bounds hold also for randomized algorithms. Chen, Van Vliet and Woeginger give a preemptive optimal algorithm and a matching lower bound for *identical machines* [1]. The competitive ratio of the algorithm is $m^m/(m^m-(m-1)^m)$. Seiden [13] also gives such an optimal algorithm. His algorithm, in many cases has a lower makespan than the one in [1], however (naturally) it has the same competitive ratio. A lower bound of the same value is given independently by Sgall [15]. For *two uniformly related machines*, the tight competitive ratio is known to be $(q+1)^2/(q^2+q+1)$, given in [5] and independently in [17]. Those results are extended for a class of m related machines with *non-decreasing speed ratios* in [3]. The tight result in that case is a function of all the speeds. A lower bound of 2 on the overall competitive ratio is given in [6]. The same lower bound can be derived from [3] since the lower bound in [6] is built using a set of machines with a geometric sequence of speeds (identical speed ratios).

The study of *non-preemptive semi-online* scheduling has focused on LPT, a greedy algorithm that assigns each arriving job to the machine which would finish it first. Non-preemptive semi-online scheduling on *identical machines* is studied already by Graham [10]. He shows that the exact approximation ratio of LPT is $\frac{4m-1}{3m}$. This value equals $7/6$ for two machines and $11/9 \approx 1.22222$ for three machines. [14] shows that LPT is the best (non-preemptive) semi-online algorithm for two machines. They also give a lower bound of 1.18046 for three machines.

For *related machines*, Mireault, Orlin and Vohra [12] give a complete analysis of LPT on two related machines. They show that the interval $q \geq 1$ is partitioned into nine intervals, and introduce a function which gives the competitive ratio in each interval. [4] shows that for some of those intervals, LPT is optimal among semi-online algorithms, while in others it is not. For a general setting of m machines Friesen [7] shows that the overall approximation ratio of LPT is between 1.52 and 5/3. Dobson [2] claims to improve the upper bound to 1.583. However, his proof does not seem complete.

Our results: In this paper we give the exact competitive ratio for semi-online preemptive scheduling on two related machines as a function of the speed ratio q (see Figure 1). The result is valid for randomized algorithms as well as deterministic algorithms. We prove that, for $q > 2$, an algorithm for the problem must use idle time to be optimal. In previous work on preemptive on-line and semi-online scheduling $[1, 17, 5, 13, 14, 3]$, no algorithm uses idle time.

Structure of the paper: In Section 2 we state and discuss the main results of the paper. In the next two sections we present algorithms of optimal competitive ratio; for the case $q \leq 2$ in Section 3, and in Section 4 for $q > 2$. We finish the paper by proving matching lower bounds in Section 5.

2 Main Results and Discussion

In the following two sections we will prove the following theorem depicted in Figure 1.

Theorem 1. *The optimal competitive ratio of* deterministic or randomized *preemptive semi-online algorithms is*

$$r_1 = \frac{3(q+1)}{3q+2}, \qquad \text{for } 1 \leq q \leq 3, \text{ and}$$
$$r_2 = \frac{2q(q+1)}{2q^2+q+1}, \quad \text{for } q \geq 3.$$

This function reaches its maximum at $q = 1$, hence the optimal overall competitive ratio of deterministic or randomized *preemptive semi-online algorithms is 1.2.*

To prove the theorem we design two classes of algorithms.

The first class is defined for $1 \leq q \leq 2$, and gives algorithms that do not use idle time. Our class of algorithms for $1 \leq q \leq 2$ resembles

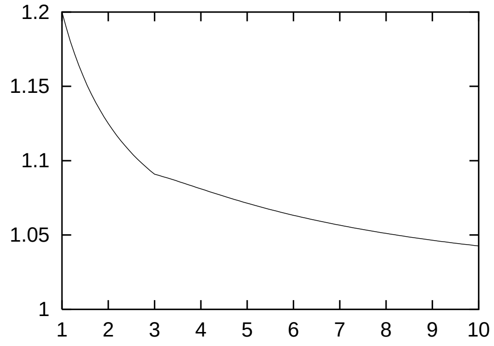


Fig. 1. The competitive ratio as a function of q

previously known algorithms. Comparing our result for the case $q = 1$ to previous results gives interesting insight to this problem. First, our result for $q = 1$ reduces to the result for two identical machines in [14]. If we compare the best semi-online competitive ratio for two related machines, in the preemptive case (6/5) and the non-preemptive case (7/6), we get the phenomenon that preemptive scheduling is harder! Of course the reason for this is that the optimal off-line algorithm is also allowed to preempt, hence there is no clear relation between the cases. However for on-line scheduling the situation is different and more natural. The tight competitive ratio for non-preemptive scheduling is $3/2$ [8, 9], whereas the tight competitive ratio for preemptive scheduling is lower, $4/3$ [1, 13, 15].

For $q > 2$, we design a different class of algorithms which uses idle time. It has been an open problem to determine a case where idle time is required. It seems reasonable that for preemptive models where the exact competitive ratio was not yet determined (as the general case of on-line scheduling on uniformly related machines), introducing idle time might lead to the design of algorithms with optimal competitive ratio. However, it is not clear how this can be done, and whether, indeed, this can lead to an optimal solution. It is not known how to design such an algorithm or how to perform the analysis in those cases.

Note that our algorithms for $q > 2$ introduce idle time only when the first job is scheduled. Further jobs do not introduce new idle time.

Once the gap resulting from the assignment of the first job is filled, the algorithm does not have idle time. This construction is simple enough to analyze, and leads to algorithms of optimal competitive ratio. The construction is not artificial, since it is not possible to design a traditional algorithm (without idle time) for these cases, as the following lemma shows.

In the proof of the lemma and in later proofs, we shall use the following observation. Let p be the size of the first job in the input sequence. Let P be total size of the jobs in the input sequence. Then the optimal makespan is $\max(p, \frac{P}{1+1/q})$. This means that if $P \geq p(1+\frac{1}{q})$, the optimal makespan is $\frac{P}{1+1/q} = \frac{qP}{q+1}$. This is a special case of a result from [11].

Lemma 1. *If $q > 2$, no algorithm that never uses idle time can have a competitive ratio of* $C = \max\{r_1 = \frac{3(q+1)}{2+3q}, r_2 = \frac{2q(q+1)}{2q^2+q+1}\}$.

Proof. Assume for the sake of contradiction that an algorithm exists which does not use idle time and has competitive ratio C. Consider such an algorithm and a sequence of two unit jobs. Since $C < 2$, and since after the arrival of one unit job, $\mathrm{OPT} = 1$, the first job is assigned completely to the fast machine. (Splitting the job would introduce idle time, and assigning it to the slow machine would violate the competitive ratio.) When the second job arrives, $\mathrm{OPT} = \frac{2q}{q+1}$, hence the maximum total size that can be accommodated on the fast machine is $r_i\frac{2q}{q+1} - 1$, and $\frac{1}{q}$ on the slow machine. For $2 < q \leq 3$, this adds to $\frac{3q^2+q+2}{3q^2+2q}$, and for $q \geq 3$, it adds to $\frac{2q^3+q^2+1}{2q^3+q^2+q}$. For $q > 2$, both are less than 1. □

Note that the break point in the competitive ratio function is $q = 3$ and not $q = 2$. Even though the algorithms for $1 \leq q \leq 2$ and $2 < q \leq 3$ are different, they have the same function as competitive ratio.

3 Algorithms for $q \leq 2$

In this section we present algorithms without idle time.

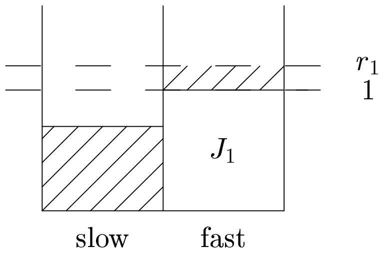
Lemma 2. *For $q \leq 2$, there exists a preemptive algorithm of competitive ratio $r_1 = \frac{3(q+1)}{3q+2}$ which does not use idle time.*

Proof. The algorithm works similarly to the algorithm for on-line scheduling [1] (with a different competitive ratio). We describe it in other words for convenience of proof.

The first job J_1 is scheduled at time 0 on the fast machine. Without loss of generality, we assume that it has size 1.

As long as the total size of jobs does not exceed $1+\frac{1}{q}$, $\mathrm{OPT} = 1$. Thus, we can use the interval 1 to r_1 on the fast machine without violating the competitive ratio. We use that interval first, and when it is full we go on to the interval $(1+\frac{1}{q}-r_1)q$ on the slow machine. When these two intervals are filled, the total size of jobs scheduled is $1+\frac{1}{q}$. At this point, some job may be only partially assigned. Denote this job by J_p. Since $(1+\frac{1}{q}-r_1)q = q+1-q(1+\frac{1}{3q+2}) = 1-\frac{q}{3q+2} = \frac{2q+2}{3q+2} < 1$, the load on the slow machine does not exceed 1, hence even if a job was split between the two machines, its two parts do not overlap in time.

We have the following situation. The loads are $r_1 = \frac{3q+3}{3q+2}$ (fast) and $(1+\frac{1}{q}-r_1)q = \frac{2q+2}{3q+2}$ (slow). Note that the ratio of the loads is $3:2$.



From now on, we keep the ratio of $3:2$ between loads, so that the fast machine is always more loaded. Any arriving job of size x will be split into two pieces of size $\frac{3qx}{3q+2}$ (fast machine) and $\frac{2x}{3q+2}$ (slow machine). The ratio between the extra loads is $3:2$ as required. For the case where the total sum of sizes P is large enough ($P \geq 1+\frac{1}{q}$), $\mathrm{OPT} = \frac{qP}{q+1}$, the on-line load on the fast machine (which is larger) is $\frac{3qP}{3q+2}$, and the competitive ratio is r_1. The above is done also to the part of the job J_p that was not scheduled. This completes the description of the algorithm. To complete the proof we need to show the following.

a. The second part of J_p is scheduled properly, and

b. Any future job J is scheduled properly.

Proof of a. **Case 1:** *J_P was the second job to arrive.* Since it is scheduled on the fast machine only after time 1, we only need to show that on the slow machine, it will be completed no later than at time 1.

Let x be the size of the second part of J_P. We need to schedule $\frac{2x}{3q+2}$ of the job, (which consumes $\frac{2xq}{3q+2}$ units of time on the slow machine). Since the size of the second job is at most 1, and $\frac{1}{q}$ of it was scheduled, $x \leq 1 - \frac{1}{q}$. After scheduling all of J_p, the load on the slow machine is

$$\frac{2q+2}{3q+2} + \frac{2xq}{3q+2} \leq \frac{2q+2+2q(1-1/q)}{3q+2} = \frac{4q}{3q+2} \leq 1 \ ,$$

since $q \leq 2$.

Case 2: *J_p arrives as the third job or later.* In this case, the first job has size one and the second job has size less than $\frac{1}{q}$. The sequence of sizes is non-increasing, hence we get that all later jobs including J_p are no larger than $\frac{1}{q}$. Clearly, an invalid schedule cannot occur unless J_p is scheduled after time 1 on the slow machine, in which case, it is scheduled on the fast machine no earlier than at time r_1 (otherwise its total load would be more than 1, and the size more than $\frac{1}{q}$). Hence we only need to show that the load on the slow machine cannot exceed r_1. If it exceeds r_1, then the total size of jobs is more than

$$\frac{r_1}{q} + \frac{3}{2}r_1 = \frac{3(q+1)}{3q+2}\left(\frac{1}{q} + \frac{3}{2}\right) = \frac{3}{2}\left(1 + \frac{1}{q}\right) > 1 + \frac{2}{q} \geq 1 + \frac{1}{q} + x \ ,$$

which is a contradiction, since the total size is exactly $1 + \frac{1}{q} + x$ at this point.

Proof of b. In this case we only need to show that the interval on the slow machine is enough to schedule $\frac{2x}{3q+2}$. Let P be the total size of previous jobs. Then $\frac{3q}{3q+2}P$ is the load of the fast machine and $\frac{2q}{3q+2}P$ is the load of the slow one just before scheduling J. The extra load that J causes is $\frac{2qx}{3q+2}$. Hence we need $\frac{Pq}{3q+2} \geq \frac{2qx}{3q+2}$. This clearly holds, since $x \leq \frac{P}{2}$ because J is at least the third job. □

4 Algorithms for $q > 2$

In this section, we present algorithms that use idle time.

Lemma 3. *For every $q \geq 2$, there exists an algorithm with competitive ratio* $C = \max\{r_1 = \frac{3(q+1)}{2+3q}, r_2 = \frac{2q(q+1)}{2q^2+q+1}\}$.

Proof. We define the algorithm. The assignment of the first job causes idle time, later jobs will be scheduled during those gaps and will not introduce more idle time. Let $r_i = r_1$, if $2 \leq q \leq 3$, and $r_i = r_2$, if $q \geq 3$.

Assume without loss of generality that the first job has size 1. We split this job into 2 pieces of sizes $\frac{q-r_i}{q-1}$ and $\frac{r_i-1}{q-1}$. For $q \geq 2$, both sizes are fractions between 0 and 1. The first piece is scheduled at time 0 on the fast machine, and the second at time $\frac{q-r_i}{q-1}$ on the slow machine (until time r_i). In general, future jobs (or parts of jobs) assigned to the fast machine will be scheduled one after the other without any idle time. Jobs (or parts of jobs) assigned to the slow machine will be scheduled at the first idle time. Once no idle time is left, they will be scheduled after time r_i (it might be necessary to split some job and continue it after time r_i).

Similarly to the previous algorithm, as long as the total size is at most $1+\frac{1}{q}$, new jobs are scheduled on the fast machine between time $\frac{q-r_i}{q-1}$ and r_i, and then on the slow machine, starting at time zero.

At the time when the total size of jobs is exactly $1+\frac{1}{q}$, there is still idle time on the slow machine (according to the definition of the algorithm, in order to cover all the idle time, the total size of jobs should be at least $r_i(1+\frac{1}{q})$). The loads of the machines, not including idle time, are r_i (fast) and $(1+\frac{1}{q}-r_i)q$ (slow). If $q \leq 3$, the ratio of these loads is $3:2$ as in the algorithm for $q \leq 2$. If $q \geq 3$, the loads without idle time are $r_2 = \frac{2q(q+1)}{2q^2+q+1}$ and $q+1-qr_2 = \frac{(q+1)^2}{2q^2+q+1}$, and the ratio is $2:(1+\frac{1}{q})$. Since $\mathrm{OPT} \geq 1$ already after the first job, the competitive ratio r_i is maintained up to this point.

From this time on, the total size of jobs, P, satisfies $P \geq 1+\frac{1}{q}$, hence $\mathrm{OPT} = \frac{qP}{q+1}$. If $q \leq 3$, we will keep the ratio $3:2$ between the total processing times assigned to the fast and the slow machines. Once the idle time is gone, this will be the ratio between the loads. If $q \geq 3$, we keep the ratio $2 : (1 + \frac{1}{q})$. This gives the desired competitive ratio.

We again need to show that the leftover of the job for which the total size reached $1 + \frac{1}{q}$ is assigned properly, and that future jobs are assigned properly.

Note that the fast machine is occupied during the time $[0, r_i]$, since for $q \geq 2$, $\frac{r_i - 1}{q-1} + r_i \leq 1 + \frac{1}{q}$.

Since the free time slots on the two machines before time r_i are disjoint, there is no difference between a complete job and leftovers of a job. Denote the job (or the leftover of a job) that is being scheduled by J.

Case 1: $2 \leq q \leq 3$. Let P be the total size of previous jobs (such that $P \geq 1 + \frac{1}{q}$). Let x be the size of the job J. The loads (not including idle time) on the fast and slow machines are $\frac{Pq}{q+1} \cdot \frac{3(q+1)}{3q+2}$ and $\frac{Pq}{q+1} \cdot \frac{2(q+1)}{3q+2}$ respectively. Thus, the available interval for a new job on the slow machine has length $\frac{Pq}{3q+2}$. The load that J incurs on the slow machine is $\frac{2xq}{3q+2}$, hence we need $x \leq \frac{P}{2}$. If J is at least the third job in the sequence, it is always true. Otherwise J is a leftover. Since $\frac{1}{q}$ of a unit of this job was scheduled already, $x \leq 1 - \frac{1}{q}$. For $q \leq 3$, $1 - \frac{1}{q} \leq \frac{1}{2}(1 + \frac{1}{q}) \leq \frac{1}{2}P$ holds.

Case 2: $q \geq 3$. In this case, the loads (not including idle time) are: $\frac{2Pq^2}{2q^2+q+1}$ (fast) and $\frac{(q^2+q)P}{2q^2+q+1}$ (slow). The interval available for a new job on the slow machine has length $\frac{P(q^2-q)}{2q^2+q+1}$, and the load of J on the slow machine is $\frac{(q^2+q)x}{2q^2+q+1}$. Thus, we need $x \leq \frac{P(q-1)}{q+1}$. If J is the third job or more, then $x \leq \frac{P}{2}$ which is at most $\frac{P(q-1)}{q+1}$, if $q \geq 3$. Otherwise, as in the previous case, $x \leq 1 - \frac{1}{q}$. Since $P \geq 1 + \frac{1}{q}$, we get

$$x \leq 1 - \frac{1}{q} = \frac{(q-1)(q+1)}{q^2+q} \leq \frac{q-1}{q+1} P \ .$$

□

This completes the proof of the upper bound which follows from Lemmas 2 and 3. Lemma 1 shows that the idle time in Lemma 3 is actually required.

5 Lower Bounds

To prove matching lower bounds we use a simplified version of a lemma proven in [6].

Lemma 4. *Consider a sequence of at least 2 jobs, where J_{n-1}, J_n are the last 2 jobs. Let $OPT(J_i)$ be the preemptive optimal off-line cost after the arrival of J_i. The competitive ratio of any preemptive randomized on-line algorithm is at least*

$$\frac{Pq}{OPT(J_{n-1}) + qOPT(J_n)}$$

where P is the total size of all jobs in the sequence.

As in [14], we show that the most difficult cases are sequences of identical jobs. We apply the "recipe" for two sequences, consisting of two or three unit size jobs:

Lemma 5. *The competitive ratio of any deterministic or randomized on-line algorithm is at least*

$$\max\left\{r_1 = \frac{3(q+1)}{2+3q}, r_2 = \frac{2q(q+1)}{2q^2+q+1}\right\} .$$

Proof. Consider two sequences consisting of 2 and 3 unit size jobs. The optimal off-line makespan is 1 after the first job, $\frac{2q}{q+1}$ after the second and $\frac{3q}{q+1}$ after the third (if it arrives). Substituting into the bound in Lemma 4 we get that both r_1 and r_2 are valid lower bounds for all $q \geq 1$. □

This completes the proof of Theorem 1.

6 Conclusion

We designed algorithms of optimal competitive ratio for all possible combinations of two uniformly related machines. A natural open problem would be to extend this to a general set of m uniformly related machines. It can even be interesting to extend it just to a restricted class of such machines, as in [3], where the solution for preemptive *on-line* scheduling on m related machines with non-decreasing speed ratios is given. It would certainly also be interesting to give a complete solution for any combination of m machines for *on-line* (as well as semi-online) scheduling, or to extend what is known to more classes of machines.

Acknowledgments: We would like to thank Jiří Sgall for suggesting the problem and reading the paper. We would also like to thank Joan Boyar for reading and commenting on the paper.

References

1. B. Chen, A. van Vliet, and G. J. Woeginger. An Optimal Algorithm for Preemptive On-line Scheduling. *Operations Research Letters*, 18:127–131, 1995.
2. Gregory Dobson. Scheduling independent tasks on uniform processors. *SIAM Journal on Computing*, 13(4):705–716, 1984.
3. 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.
4. L. Epstein and L. M. Favrholdt. Non-Preemptive Semi-Online Scheduling on Two Related Machines. Work in progress, 2001.
5. 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.
6. L. Epstein and J. Sgall. A lower bound for on-line scheduling on uniformly related machines. *Operations Research Letters*, 26(1):17–22, 2000.
7. Donald K. Friesen. Tighter bounds for LPT scheduling on uniform processors. *SIAM Journal on Computing*, 16(3):554–560, 1987.
8. G. Galambos and G. Woeginger. An on-line scheduling heuristic with better worst case ratio than Graham's list scheduling. *Siam Journal on Computing*, 22(2):349–355, 1993.
9. R.L. Graham. Bounds for certain multiprocessor anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.
10. R.L. Graham. Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math*, 17:416–429, 1969.
11. E. Horwath, E. C. Lam, and R. Sethi. A level algorithm for preemptive scheduling. *J. Assoc. Comput. Mach.*, 24:32–43, 1977.
12. P. Mireault, J.B. Orlin, and R.V. Vohra. A parametric worst case analysis of the LPT heuristic for two uniform machines. *Operations Research*, 45:116–125, 1997.

13. S. Seiden. Preemptive Multiprocessor Scheduling with Rejection. *Theoretical Computer Science*, 262(1-2):437–458, 2001.
14. S. Seiden, J. Sgall, and G. Woeginger. Semi-online scheduling with decreasing job sizes. *Operations Research Letters*, 27(5):215–221, 2000.
15. J. Sgall. A lower bound for randomized on-line multiprocessor scheduling. *Inf. Process. Lett.*, 63(1):51–55, 1997.
16. J. Sgall. On-line scheduling. In *A. Fiat and G. J. Woeginger, editors,* Online Algorithms: The State of the Art, volume 1442 of *LNCS*, pages 196–231. Springer-Verlag, 1998.
17. J. Wen and D. Du. Preemptive on-line scheduling for two uniform processors. *Operations Research Letters*, 23:113–116, 1998.