Charles University
Faculty of Mathematics and Physics

Abstract of Doctoral Thesis

Combinatorial algorithms for online problems: Semi-online scheduling on related machines

Tomáš Ebenlendr

Branch: I4 - Discrete Models and Algorithms

2011

Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

Autoreferát disertační práce

Kombinatorické algoritmy se zaměřením na online problémy: Semi-online rozvrhování na strojích s různými rychlostmi

Tomáš Ebenlendr

Obor: I4 - diskrétní modely a algoritmy

2011

Disertační práce byla vypracována v rámci doktorského studia, které uchazeč absolvoval v Matematickém ústavu Akademie věd České republiky, za podpory Matematicko-fyzikální fakulty Univerzity Karlovy v Praze v letech 2004-2011.

Uchazeč: Mgr. Tomáš Ebenlendr

Školitel: Doc. RNDr. Jiří Sgall, DrSc.
Katedra aplikované matematiky, MFF UK
Matematický ústav Akademie věd České Republiky
Institut teoretické informatiky

Školící pracoviště: Matematický ústav Akademie věd České Republiky
Institut teoretické informatiky

Opononenti: doc. RNDr. Roman Barták, Ph.D.
Katedra teoretické informatiky a matematické logiky
MFF UK

Dr. Leah Epstein
Department of Mathematics
University of Haifa
Mount Carmel, Haifa, 31905 Israel

Prof. dr. ing. Gerhard J Woeginger
Department of Mathematics and Computer Science
Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands

Autoreferát byl rozeslán dne

Obhajoba se koná dne v před komisí pro obhajoby prací oboru I4 - diskrétní modely a algoritmyna Matematicko-fyzikální fakultě Univerzity Karlovy, FIXME(kde)

S disertací je možno se seznámit na oddělení doktorandského studia MFF UK, Ke Karlovu 3, Praha 2.

Předsedou oborové rady I-4 je prof. RNDr. Jaroslav Nešetřil, DrSc.

1 Introduction

Scheduling is now a classical problem from the area of combinatorial optimization. Approximation and online algorithms for scheduling are extensively studied, because most scheduling problems are (NP-)hard to solve exactly and because many scenarios deal with input coming in parts, with the requirement of immediate output for each part. There is a strong connection between approximation and online algorithms because many natural approximation heuristics for scheduling are in fact (semi-)online algorithms. Our work studies online and semi-online algorithms for scheduling and their possibilities.

Our online model belongs to *one-by-one* scheduling, i.e., the algorithm sees only one job and it has to decide its schedule immediately and irrevocably. The algorithm has always possibility to start the job from the beginning of the time if there is a machine where the job can be scheduled. This particular variant was studied for a long time, the first well-known algorithm for online scheduling is the Graham's [20] list scheduling algorithm, which is 2-competitive on identical machines.

We are also interested in the semi-online scheduling. The semi-online algorithms work in the same setting as the online algorithm with the exception of some advantage over the online algorithm. We study advantages that are based on some knowledge of the input sequence. One such variant is scheduling with the known sum of processing times, where the algorithm knows the total processing time in advance. We study also other knowledge-based semi-online variants.

We want to minimize makespan, that is the last time when a machine processes a job. This scenario is also called load balancing. We consider machines with different speeds (uniformly related machines) both with and without preemptions. Allowing preemptions means that the algorithm is allowed to interrupt a job at any time and continue the processing of this job on another machine or at a later time. Chapter 2 contains all results on preemptive scheduling. We added also a lower bound for nonpreemptive scheduling in Chapter 3.

2 Presented work

Our main result is Section 2.2 and it contains a framework of optimal algorithms when preemptions are allowed. This framework applies not only to online scheduling, but also to various semi-online variants. Moreover these algorithms are deterministic, yet optimal among randomized algorithms. I.e., randomization does not help these problems. The framework constructs (deterministic) algorithms with competitive ratio given as a parameter. These algorithms do not fail (and maintain given competitive ratio) if there exist any (possibly randomized) algorithm

with this ratio for the given set of machines. I.e., if we know any R-competitive (randomized) algorithm, then we can run our (deterministic) algorithm with the parameter R and it successfully maintains to be R-competitive.

We also developed linear programs that compute the optimal competitive ratio for an arbitrary fixed set of speeds for online scheduling and all semi-online variants studied in this work. These linear programs can be found in Section 2.3. So the algorithms from our framework may solve the linear programs in its initialization stage to compute the parameter R. We obtain the exact value of R for every given set of machines this way.

The structure of the worst set of machines is unknown to us, thus we don't know the exact number of overall competitive ratio. We use the above mentioned linear programs to obtain new overall lower bounds for some of the semi-online variants. This cannot be done directly, as the speeds of the machines determine the linear inequalities. We obtain a quadratic (non-convex) program by taking the speeds as variables and we try to find some local optimum numerically by a computer. Verification of these bounds is just checking of the inequalities as we provide the values of all involved variables. Moreover these variables directly correspond to the input instance (the speeds of machines and the sizes of jobs). An important result is improving a lower bound of 2 based on a geometric sequence of machines and jobs [15] to a bound of 2.112 based on the input sequence that was found by a computer.

We show that one of the studied semi-online restrictions does not help to the algorithm in general, i.e., we show how to modify any hard input sequence to satisfy that particular restriction.

We provide new overall upper bound for one semi-online case, where the competitive ratio can be computed directly without solving linear programs. The online case as well as all semi-online cases are upper bounded by e-competitive randomized algorithm for online variant from the author's master thesis [c1]. Existence of that algorithm proves that algorithms from our framework are at worst e-competitive. Note that our algorithms are deterministic.

We ask for closed formulas for the competitive ratio for small numbers of machines in Section 2.4. We can acquire them by solving the linear programs symbolically (i.e., we let the parameters — the speeds of machines — to be variables) and we do this for up to four machines. We know that an optimal solution is a vertex of the bounding polytope. This polytope is just the intersection of halfspaces which represent the constraints. Then the vertices are intersections of hyperplanes — boundaries of these halfspaces. We use exhaustive search over all points that are generated by all possible intersections of dimension zero of these hyperplanes. There can be a large number of them (thousands), but nowadays algebraic software can help us to filter out most of the intersections that do not correspond

to an optimal solution for any valid values of parameters (machine speeds). So we are left with a few (typically ten to twenty) intersections and we can do the final search by hand. Note that the result corresponds to several intersections, the optimality of each of them depends on the actual values of parameters. Formula of the resulting competitive ratio which corresponds to one of the intersections is a ratio of two polynomials, because it is a solution of a set of linear equations. We analyzed the cases of small machines for most of the studied semi-online variants as well as for the online scheduling.

We augment the work by a new lower bound for the case where preemptions are not allowed in Chapter 3. This bound is based on counting how many jobs in a geometric sequence fit on one machine. Then taking the sum over all machines and comparing this sum to the number of jobs in the sequence gives a bound. If we let the common ratio of the geometric sequence to limit to 1, we obtain our new bound. This bound is in the form of a constraint on the value of a definite integral. We evaluate this bound numerically and we obtain the value of 2.564. The previous lower bound of 2.438 [3] was achieved by a computer verification of a large set of configurations of the output schedule.

3 Our model

We study online scheduling on uniformly related machines. It means that p/s of time is needed to process a job with processing time p on a machine with speed s. We are interested in the makespan (the length of the schedule). We use m to denote the number of machines and n for the number of jobs.

The one-by-one online scheduling is studied in this work, i.e., the algorithm sees only one job on the input and it has to fully determine the schedule of this job before it sees the next job (if there is one). That means the order of the jobs in the input sequence has major influence on the run of the algorithm. We say that the algorithm is in the *step* j if it schedules jth job. The word *time* is reserved for the time in the schedule (e.g., starting and finishing time of a job) and is totally unrelated to the steps of the algorithm.

We also study various semi-online variants of the online scheduling problem, that means we give some advantage to the online algorithm. The semi-online problems are in fact online problems, which were derived from some original online problem. We study problems derived by restricting the set of valid inputs. The semi-online algorithms are sometimes categorized as approximations, because the algorithms solving these problems are not valid in the original online problem.

4 Overview of previous and our results

The first remarkable scheduling algorithm, which is described by Graham [20], is already online, $(2 - \frac{1}{m})$-competitive greedy algorithm for makespan scheduling on m identical machines. This problem and algorithm is also known as list scheduling. This algorithm does not use preemptions, but the competitive analysis holds also in the preemptive setting.

Offline scheduling

The offline scheduling with the makespan objective is well understood, and results for uniformly related machines were usually obtained using similar methods as for identical machines. Algorithms for exact solutions are known when preemptions are allowed for identical machines [30] as well as for related machines [27, 19], however our algorithm can be also used here. It simplifies exactly to the algorithm from the master thesis of the author [c1]. We consider this simplified algorithm much easier to understand than the previous algorithms. The problem is NP-hard when preemptions are not allowed, but polynomial approximation schemes are known for identical machines [26] as well as for related machines [25]. That means that for arbitrarily small fixed ϵ there exist a polynomial algorithm with approximation ratio $(1+\epsilon)$. Thus we can solve the problem with an arbitrary fixed precision.

Online nonpreemptive scheduling

The online makespan scheduling exposes a different situation. Let us look at the nonpreemptive version first. The greedy algorithm (list scheduling) is an optimal deterministic algorithm for two and three identical machines [17] achieving competitive ratios of $\frac{3}{2}$ and $\frac{5}{3}$ respectively. No tight results are known for more machines, but there are known better algorithms than the greedy one. For $m = 4$, the best known algorithm is by Chen et al. [5] with competitive ratio of 1.733,the corresponding lower bound of $\sqrt{3} = 1.732$ is by Rudin and Chandrasekaran [32]. The latest deterministic algorithm for general m of identical machines is by Fleischer and Wahl [18] achieving a competitive ratio of 1.9201. The best known lower bound is by Rudin [31] and has value of 1.880.

Optimal randomized algorithm is known only for two identical machines, see Bartal et al. [2]. The lower bound of $1 + ((m/(m-1))^m - 1)^{-1}$ for any m identical machines was obtained by Chen et al. [4] and Sgall [35] independently. This and all the previously known lower bounds bounded also preemptive algorithms. A better lower bound is known for $m = 3$, with value of $27/19 + \epsilon$ for some small fixed ϵ by Tichý [38]. This bound uses the fact that jobs cannot be split and thus it is the

first bound that does not work in the preemptive setting. Randomized algorithms better than deterministic are also known for $m = 3, \ldots, 7$, the result by Seiden [34]. A better algorithm for general m with competitive ratio 1.916 was described by Albers [1]. This algorithm uses only one random bit in its initialization, i.e., we have two deterministic algorithms with the nice property, that the arithmetic mean of their competitive ratios is good for any input sequence.

Much less is known for the case of related machines. Best known general algorithms are 5.828-competitive deterministic and 4.311-competitive randomized one by Berman et al. [3]. They also give the first lower bounds that do not hold for identical machines. The deterministic one has value of 2.438 and is done by computer, it checked a large graph of the output schedules. We present a new lower bound of 2.564 in this work. Our lower bound uses the computer only to do one numerical optimization of a single-variable integral inequality.

The randomized lower bound of 2 (which holds also in the preemptive setting) was constructed by Epstein and Sgall [15]. Epstein et al. [14] give an algorithm for two related machines, they analyze the competitive ratio (and lower bounds) parametrically based on the ratio of the speeds of the two machines. Their algorithm is optimal for equal speeds as well as for one machine twice faster than the other.

Online preemptive scheduling

Now we will describe the situation for preemptive on-line scheduling. Scheduling with preemptions in the basic model is much simpler because we can split the load between machines when scheduling every single job. As we already said the lower bound of $1 + ((m/(m-1))^m - 1)^{-1}$ for any m of identical machines [4, 35] applies here. Chen et al. also constructed the matching optimal algorithm in [6].

For $m = 2$ related machines, the optimal algorithm for any set of speeds was given by Wen and Du [39] as well as Epstein et al. [14].

The best previous lower bound for a general number of related machines was the bound of 2 by Epstein and Sgall [15]. The author presented in his master thesis the 4-competitive deterministic and the e-competitive randomized algorithms.

We will show a deterministic algorithm that computes the optimal competitive ratio (even for randomized algorithms) for any given set of speeds using the linear programming and then it maintains this competitive ratio for any input sequence of jobs. This means that the randomization does not give any advantage with respect to the makespan for this problem. However we are not able to obtain a new non-trivial overall bound on the outcome of the linear program, thus we know that the overall competitive ratio is somewhere between 2.112 and $e = 2.718$. The lower bound is also our new result. It is a sequence for $m = 200$. Plugging this sequence into the lemma from [15] gives the desired bound. We list all processing

times of jobs and all speeds of machines, thus verification is simple, although it may be tedious. The speeds and processing times were found by a computer as a local extreme of a non-convex quadratic program. We also analyzed the case of four machines and we identify the formula of the competitive ratio as a function of speeds of machines.

Semi-online restrictions

We will present our algorithm in more general form as a framework to construct the optimal algorithm for many semi-online variants of preemptive scheduling on related machines. Namely these are the variants that can be expressed as restrictions of the set of allowed inputs.

We stress that our semi-online results are only on related machines with preemptions allowed, we will not repeat this in following paragraphs.

Known sum of processing times, denoted $\sum p_j = P$. The algorithm knows the total sum of processing times. That means, for example, that it also knows (for every step) if the job on input is the last (nonzero) one. This restriction, for non-preemptive version on two identical machines, was studied by Kellerer et al. in [29], which is probably also the first paper which studied and compared several notions of semi-online algorithms. The non-preemptive version on identical machines is further studied under the name online partition.

We will show that the overall ratio is the same as in the general online case, while it is much lower for small number of machines. We note that there is 1-approximation possible for two machines and we analyze the case of three and four machines.

Non-increasing processing times, denoted *decr*. Here each subsequent job must be smaller or equal to the previous one. The greedy algorithm (list) on identical machines was already analyzed by Graham [21]. The optimal algorithm for identical machines with preemptions was given by Seiden et al. [33] and for any speed combination of two related machines both without [12] and with [13] preemptions by Epstein and Favrholdt.

We prove that the worst sequence is that of all jobs equal for any sequence of speeds. Moreover the knowledge that all jobs are equal will not help the algorithm thus the competitive ratios of these two cases are equal.

We provide formula giving optimal competitive ratio for any fixed set of speeds and we provide upper and lower bounds on overall competitive ratio.

Known optimal makespan, denoted $C^*_{\max} = T$. The semi-online algorithm that outputs the optimal schedule was the main result of the author's master thesis [c1], the framework from this work is in fact a deep generalization of that algorithm.

The nonpreemptive version on identical machines is called bin stretching [11].

Known maximal processing time, denoted $p_{\max} = p$. The algorithm knows the value of maximal processing time. It is easy to see that any algorithm that works in the setting where the first job is the maximal one, can be emulated also here, yielding an algorithm with the same competitive ratio. The algorithm just schedules the maximal job virtually in the step zero, i.e., it creates reservation of timeslots for this job. Then the algorithm simply uses this reservation for first job of the maximal processing time. All other jobs are scheduled normally. This restriction was introduced by He and Zhang [24] for non-preemptive scheduling on two identical machines, with the proof that the greedy algorithm (list) is optimal. The complete analysis of the preemptive version on two related machines was given by He and Jiang [22]. Seiden et al. [33] studied the case of the identical machines and they show that the approximation ratio is the same for known maximal processing time as for the non-increasing processing times.

We show that this is not the case for general speeds. We provide the new lower bound of 1.908, which was found by the computer as a locally worst sequence on $m = 200$ machines. We also give a complete analysis for three and four machines.

Inexact partial information. The knowledge described above may not be given exactly. Here we consider the case when an interval of admissible values is known. For example, the algorithm is given $\bar{P}$ and α, and the following bound on total processing time: $\bar{P} \leq \sum p_j \leq \alpha \bar{P}$. Such variants were studied first by Tan and He [37] without preemptions and for two identical machines. The complete analysis of the preemptive version with approximately known optimum and maximal processing time on two machines and approximately known optimum on identical machines was given by Jiang and He [28].

We give a complete analysis for an approximately known optimum for three machines. This is denoted $T \leq C^*_{\max} \leq \alpha T$.

Tightly grouped processing times, denoted $p \leq p_j \leq \alpha p$. The bounds on processing times of the jobs are given here. This restriction is introduced by He and Zhang [24]. They prove that the greedy algorithm (list) is optimal for two identical machines without preemption. The complete analysis of the preemptive version on two related machines was given independently by Du [9] and He and Jiang [22]. The case of three identical machines has been also fully analyzed by He and Jiang [23].

We sketch a complete analysis for two machines, reproving and simplifying the results of [9], [22].

Combined restrictions. The algorithm may be provided with a combination of informations described above. Some combinations were studied by Tan and He [36] for two identical machines without preemption.

We present two variants, the knowledge of the total processing time combined

either with the known maximal processing time or with non-increasing jobs. We show that overall ratio is the same as if the algorithm does not know the total processing time in advance, although it differs for any fixed set of machines. We provide a formula giving optimal competitive ratio for any sequence of speeds in the case with non-increasing processing times. We also provide full analysis of scheduling on three and four machines in the case with know maximal processing time.

Other semi-online scenarios

Our framework applies directly in the semi-online variants described above. We list other semi-online variants below. Our framework is useful in some of these variants, but it requires some additional work.

Local knowledge. Our framework is designed to work with some global knowledge and all semi-online variants above fall in this category. But there have been studied models that contain some local information also. Our framework can be used there, formally it involves adding some information to each job and reformulating the semi-online restriction as a requirement of consistency on these informations. The algorithm gets all the additional information about the job together with its processing time.

One example is the knowledge that the job is the last one. The additional information is just a single bit indicator for each job. The consistency requirement says that only the last job has this bit set. The information that the job is the last one is not useful by itself. But it can be combined with the knowledge that the last job has to be the maximal one [40, 16]. We have not performed any calculation.

Buffers. Limited reordering of jobs is allowed here. Namely, the algorithm has a buffer on limited number of jobs and it can store the job there instead of scheduling it. It is allowed to store the job to the buffer and schedule another job from the buffer in the same step. The algorithm has to schedule all jobs from the buffer when there is no further job on input. This variant was introduced for the non-preemptive version by Kellerer et al. [29], later tight bounds on identical machines were given by Englert et al. [10]. The preemptive version was recently studied by Dósa and Epstein [7].

Our framework cannot work here as is. But we can make a semi-online knowledge from the buffering part of each algorithm and use our framework there. It turns out that our framework gives the optimal algorithm for an arbitrary buffering strategy. Then it is much simpler to find the best buffering strategy, namely it can be proved that storing largest jobs works. We will publish the proofs in an upcoming article.

Resource augmentation. The semi-online variants, where the algorithm is given

some resources that are not available in the optimal schedule, are also studied. E.g., suppose that we want to know how we can perform by acquiring second set of machines. Such scenario is studied in [8] for two identical machines: The algorithm schedules to four machines, but its output is compared to the best schedule which uses only two machines. We give no hope in our framework to be useful here. The proof of the correctness of the algorithm from our framework is tightly connected to the knowledge of the optimal makespan under the conditions that the algorithm has. The resource-augmented algorithms clearly operate under different conditions.

5 Preliminaries

We schedule jobs $\mathcal{J}$ to the set of machines $\mathcal{M} = \{M_1, \ldots, M_m\}$. The machine M_i has speed s_i. We want to schedule jobs from the sequence $\mathcal{J} = (J_1, \ldots, J_n)$, but the algorithm has to provide irrevocable schedule of job J_j to see the job J_{j+1} or to learn the value n in the case that $j = n$. The job J_j has processing time p_j, that is the work that needs to be assigned to the job in a valid schedule. We denote the sequence of jobs $(J_1, \ldots, J_j)$ by $\mathcal{J}_{[j]}$.

Semi-online restrictions that we study are restrictions of the set of allowed inputs. We denote a general semi-online restriction by Ψ. We say that input $\mathcal{J}$ satisfies the restriction, if $\mathcal{J} \in \Psi$. The prefix of valid input needs not to be a valid input, thus we define $\mathrm{pref}(\Psi)$, the set of all prefixes of all valid input sequences. These prefixes are in fact sequences that the algorithm can see at some point.

We measure the makespan of the schedule. We use the notation $C^*_{\max}[\mathcal{J}']$ for the value of optimal makespan of schedule of jobs $\mathcal{J}'$. We extend this notion from the perspective of the algorithm to sequences from $\mathrm{pref}(\Psi)$ that are not in Ψ naturally, as a minimal makespan of any Ψ-valid extension of the considered sequence:

Definition 2.1.4 *For an input restriction Ψ and an input sequence $\mathcal{J}$, we define the optimal makespan as infimum over all possible end extensions of $\mathcal{J}$ that satisfy Ψ:*

$$C^{*,\Psi}_{\max}[\mathcal{J}] = \inf \{C^*_{\max}[\mathcal{J}'] \mid \mathcal{J}' \in \Psi \ \& \ \mathcal{J} \in \mathrm{pref}(\{\mathcal{J}'\})\}$$

This value is very simple to compute for studied semi-online restrictions as the value $C^*_{\max}$ is given by following formula [27, 19, c1]:

$$C^*_{\max}[\mathcal{J}] \quad = \quad \max\left\{\frac{P}{S}, \frac{P_1}{S_1}, \frac{P_2}{S_2}, \ldots, \frac{P_{m-1}}{S_{m-1}}\right\},$$

where P_i is the sum of i largest processing times and S_i is the sum of i fastest speeds.

6 The lower bound

We use the lower bound from [15] which holds for online algorithms:

Lemma 2.1.2([15]) *For any randomized R-competitive online algorithm* A *for preemptive scheduling on m machines, and for any input sequence $\mathcal{J}$ we have*

$$\sum_{j=1}^{n} p_j \quad \leq \quad R \cdot \sum_{i=1}^{m} s_i C^*_{\max}[\mathcal{J}_{[n-i+1]}].$$

For non-preemptive scheduling, the same holds if $C^*_{\max}$ refers to the non-preemptive optimal makespan.

Our algorithm proves this bound to be tight in the case of preemptive scheduling. Moreover we extend this bound to the semi-online case:

Lemma 2.1.5 *Given any randomized R-approximation semi-online algorithm* A *for preemptive scheduling on m machines with an input restriction Ψ, then for any input sequence $\mathcal{J} \in \Psi$ and for any subsequence of jobs $1 \leq j_1 < j_2 < \cdots < j_k \leq n$ we have*

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

Having a witness sequence $\mathcal{J}$ that forbids approximation ratio R according to Lemma 2.1.5, the adversary will submit jobs from this sequence one by one until it finds that makespan of the algorithm's output is larger than $RC^{*,\Psi}_{\max}[\mathcal{J}_{[i]}]$, where $\mathcal{J}_{[i]}$ is the sequence of jobs submitted so far. Then the adversary has to find sequence $\mathcal{J}'$ which begins with $\mathcal{J}$ and leads to the same makespan: $\mathcal{J}' = \operatorname{argmin}_{\mathcal{J}' \supseteq \mathcal{J}_{[i]} \& \mathcal{J}' \in \Psi}\{C^{*,\Psi}_{\max}[\mathcal{J}']\}$. (In fact it suffices to find some good approximation as the minimum needs not to exist). Then the adversary finishes the input sequence with jobs from $\mathcal{J}' \setminus \mathcal{J}_{[i]}$. Makespan of the output of the algorithm will not decrease, and the choice of sequence $\mathcal{J}'$ sequence ensures that $C^{*,\Psi}_{\max}$ will not increase. This way the adversary wins the game and proves that the algorithm is not R-competitive.

This bound is tight, our algorithm computes R in its initialization. The correctness of our algorithm is implied by the property that R satisfies presented bound and that $C^{*,\Psi}_{\max}$ is computable. Our algorithm maintain its competitive ratio R then.

Numerical lower bound We optimized the formula in Lemma 2.1.2 by quadratic (non-linear) program for $m = 3, \ldots, 200$ machines. You can see graphical representation of the sequence proving the lower bound of 2.05 on $m = 100$ machines. We

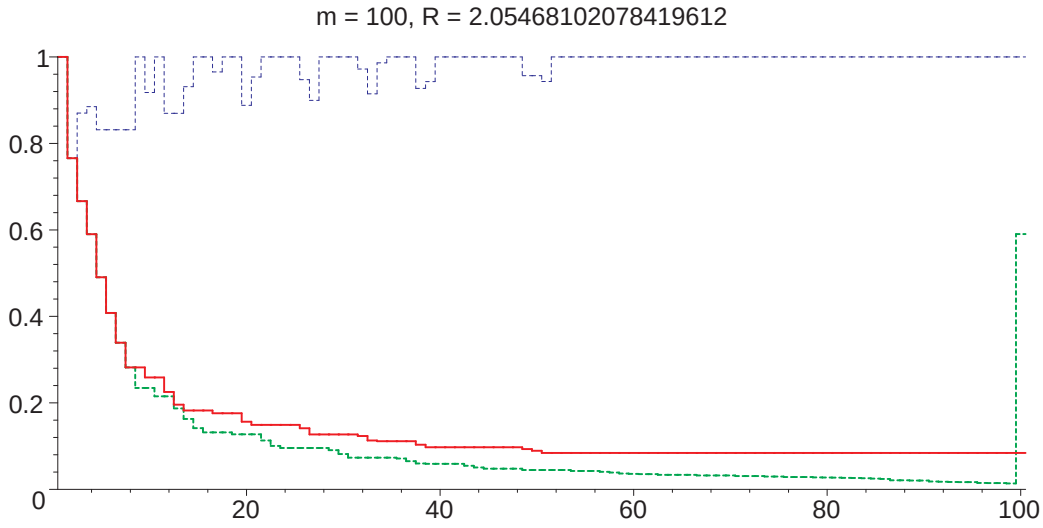

Figure 1: The instance for $m = 100$. The red bold curve shows speeds s_i, green dashed bold curve (bottom one) shows scaled job variables, and the thin dashed blue curve (top one) shows inverses of the speed ratios, i.e., s_{i-1}/s_i. The values of q_j are printed in reverse order, i.e., column i shows q_{m-i+1}. The values are scaled so that $q_m = 1 = s_1$. This ordering and scaling shows best the relations between job sizes and speeds.

provide the numerical values of the sequence on 200 machines in the attachment of the thesis. Inserting the values from this sequence into Lemma 2.1.2. proves the lower bound of 2.112 for the general online case, after

We also provide similar lower bound for the semi-online case of known maximal processing time. The sequence uses 200 machines and proves the bound of 1.908.

7 The optimal algorithm

The basic idea of the algorithm is to use the slowest machine(s) for processing of a job. This saves fast machines for large jobs that may come in the future. Our algorithm first computes its competitive ratio, and later uses this value for taking decisions about the schedule. The only condition on the computed value of the competitive ratio is that it does not violate bound provided by Lemma 2.1.2 (in the case of online scheduling) or by Lemma 2.1.5 (in the semi-online case).

In preemptive scheduling, we are allowed to use more machines for a schedule of one job (but not in parallel). Thus, we calculate the time where the job has to finish to maintain the competitive ratio. It involves computing the value of the

optimal schedule. This value is multiplied by the previously computed competitive ratio R. By this way we will get time T, which is the deadline for a job to maintain the R-competitiveness of the algorithm.

Knowing the time interval $[0, T)$ that we are allowed to use, we use as slow machines as we can to manage that the job ends exactly at that time. But we have many possibilities how to do that. We want to be able to process as many large jobs as possible in the future.

Now we introduce notion of a *virtual machine* which is generalization of a virtual machine from the author's master thesis [c1]. We index free space at each time. Let $V_k(t)$ be the k-th fastest free machine at time t. The function V_k represents k-th virtual machine. The idea is that if we have k large jobs, we cannot process them faster than scheduling them on first k virtual machines. Scheduling a job to virtual machine V_k at time t means exactly scheduling it to machine $V_k(t)$ at that time. Note that $V_k(t)$ may be undefined if there are less than k idle machines at time t. To schedule a job to V_k at time t where $V_k(t)$ is undefined means then to not schedule the job at time t to any machine.

We view the virtual machines as machines with speeds that change in time. Initially $V_i(t) = M_i$ for each t as there is no job scheduled. We also define $V_{m+1}(t)$ naturally as a function which is undefined for each t, this notation simplifies the description of the algorithm.

We want to preserve the fastest virtual machines as fast as possible (lexicographically, i.e., to optimize the speed of the fastest machine, then the second fastest, and so on), so our algorithm finds the slowest virtual machine that is able to process current job while maintaining competitive ratio. Let this machine be V_k. We will not schedule the job to any V_ℓ with $\ell < k$. If we consider any schedule of $k-1$ new jobs before scheduling the current job, then we can maintain completion times of these job even after scheduling our new job. The way we choose V_k maximizes the number $k-1$ for which this property holds.

Thus our algorithm has a pair V_k, V_{k+1} of adjacent virtual machines, such that only one of them can process current job in given time T. Now it saves a part of the faster virtual machine, by splitting the job between these two machines. Again we want to save as large part of the faster machine as possible, thus the job finishes exactly at given time being processed by either of these two virtual machines for each time $\tau < T$. See Figure 2.

After scheduling of the job we have to update the virtual machines to maintain the definition, because knowing V_k explicitly seems to be better to implement. This is done simply by choosing the not used virtual machine from the pair $V_k(t), V_k + 1(t)$ to be new $V_k(t)$ for each $t \in [0, T)$. We have to shift slower virtual machines also: $V_\ell(t) \leftarrow V_{\ell+1}(t)$ for $\ell = k+1, \ldots, m$ and $t \in [0, T)$. ($V_{m+1}(t)$ remains undefined.)

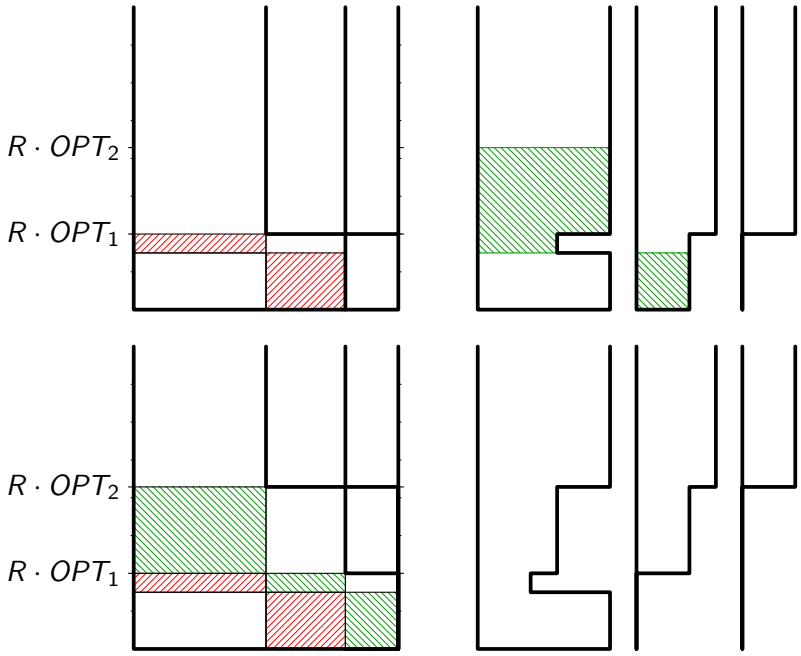

Figure 2: An illustration of a schedule of two jobs on three machines produced by RatioStretch. Vertical axis denotes the time, horizontal axis corresponds to the speed of the machines. The pictures on the left depict the schedule on the real machines, with bold lines separating the virtual machines. The pictures on the right show only the idle time on the virtual machines. The top pictures show the situation after the first job, with the second job being scheduled on the first two virtual machines. The bottom pictures show the situation with after the second job is scheduled and virtual machines updated.

Semi-online. Our algorithm remains nearly the same for semi-online or offline scheduling. Only two computations of parameters are replaced. First one, the value of the optimal makespan of jobs seen so far is replaced by computation (estimation) of minimal makespan that is achievable by some input sequence, that is consistent both with input seen so far and with the "semi-online" information provided to the algorithm, i.e., the number $C^{*,\Psi}_{\max}[\mathcal{J}_{[j]}]$ defined by Definition 2.1.4. Second computation is the estimation of the competitive ratio. Our algorithm achieves the best possible competitive ratio for any given set of machines provided that it is able to do these two computations correctly and precisely. We have also the option to provide imprecise bounds, as these may be simpler to compute. Namely, we can provide some upper bound R on the competitive ratio and we can provide a procedure that upper bounds optimal makespan. If this procedure

returns bound that is always at most α times bigger than the actual optimal makespan, then we get the algorithm with competitive ratio of αR.

Offline. The basic "semi-online" information is knowledge of optimal makespan in advance. Then our algorithm has competitive ratio 1, thus it can be compared with offline algorithms. Its time complexity is asymptotically the same as the complexity of the best known algorithm and also it issues the smallest number of preemptions in the worst case. The algorithm for the offline problem is identical to the algorithm from the author's master thesis [c1].

8 Linear programs

We can turn the bound from Lemma 2.1.2 and from simplified version of Lemma 2.1.5 into linear programs, provided that we are given fixed set of speeds. We show the linear program for the online scheduling here. The parameters $s_1 \geq \cdots \geq s_m$ are fixed. The variables are $q_1, q_2, \ldots, q_m, O_1, O_2, \ldots, O_m$. Variable q_1 corresponds to the sum of all processing times in $\mathcal{J}_{[n-m+1]}$, variables $q_2, \ldots, q_m$ to the processing times of the last $m-1$ jobs, and variables O_k correspond to $C^*_{\max}[\mathcal{J}_{[n-m+k]}]$.

$$
\begin{array}{rcll}
\textbf{maximize} & R(s_1, s_2, \ldots, s_m) = q_1 + q_2 + q_3 + \cdots + q_m & & \\
\textbf{subject to} & & & \\
q_1 + q_2 + \cdots + q_k & \leq & (s_1 + s_2 + \cdots + s_m)O_k & \text{for } k = 1, \ldots, m \\
q_j + q_{j+1} + \cdots + q_k & \leq & (s_1 + s_2 + \cdots + s_{k-j+1})O_k & \text{for } 2 \leq j \leq k \leq m \\
1 & = & s_1 O_m + s_2 O_{m-1} + \cdots + s_m O_1 & \\
q_j & \leq & q_{j+1} & \text{for } j = 2, \ldots, m-1 \\
0 & \leq & q_1 & \\
0 & \leq & q_2 &
\end{array}
$$

The numerical lower bound presented above was obtained by taking the speeds of machines as variables too.

We present also linear programs for other semi-online restrictions, with the exception of decreasing processing times. That one is so simple, that we were able to identify the exact formula for the competitive ratio as a function of speeds of machines.

Small number of machines We solve this programs for small number of machines (up to four) symbolically and we obtain formulas for the competitive ratio. We use the duality of linear programming, which says that it suffices to examine all possible basic solutions. From a geometrical point of view, we take all intersections of dimension zero of hyperplanes defined by the linear conditions, and then we test

if such an intersection is a feasible and optimal solution. Note that the feasibility and the optimality of such a solution also depends on the actual values of the parameters, so the result typically splits into several cases. Searching through all such intersections would be tedious work as it requires solving a system of linear equations and then examining the feasibility of the result. Most of this work can be automated nowadays as there is various algebraic software available.

We developed such an automation, we used Maple 9.5 for this. It builds the system of linear equations which corresponds to the examined intersection, it solves this system and then it examines the solution. It needs to check (polynomial) inequalities, thus we developed simple heuristic. The outcome of this heuristic is that the examined inequality is always satisfied, never satisfied, or that it is not sure about the satisfiability of the inequality. The last case occurs either if the inequality is too complex or if there are values of parameters (speeds of machines) for which is the inequality satisfied and other values for which is the inequality not satisfied.

The number of the intersections — the solution candidates of our linear programs — is typically thousands. Our automation typically reduces this number to less than twenty interesting intersections. We checked this cases by hand, with the help of tools that we developed for the automation. We list the solutions that are both feasible and optimal for some nonempty set of the parameter values in the thesis, together with the resulting formulas for the competitive ratio. We do this both for the online scheduling and for various semi-online restrictions.

9 The nonpreemptive lower bound

We present a lower bound on deterministic algorithms for related machines without preemption in last chapter. It is based on finding a bound on maximal amortized frequency of scheduling jobs to one machine in a particular input instance.

Let $\alpha > 1$ be fixed. The input instance consists of n jobs. The processing times of these jobs form the geometric sequence with the common ratio α. We schedule them to n machines, their speeds form the geometric sequence with the common ratio α^{-1}, the first machine has speed 1. Note that adversary stops the input sequence as soon as the algorithm produces schedule that is more than R times greater than the optimal makespan of jobs released so far. The R is the lower bound.

We can see that the optimal schedule of any prefix of $\mathcal{J}$ is to schedule them in reverse order on the fastest machines. Thus the optimal makespan is equal to the processing time of the last job.

Consider jobs $\mathcal{J}_i = (J'_1, J'_2, \ldots, J'_{n_i})$ scheduled to the machine M_i. From the competitiveness of the algorithm we get that only machines with speed $s_i > R^{-1}$

are used, moreover we get following bound on the jobs scheduled to M_i:

$$p'_j \;\geq\; \max\left\{\frac{\sum_{k=1}^{j-1} p'_k}{Rs_i - 1}, 1\right\} \quad \text{for } j = 1, 2, \ldots, n_i\,.$$

. We use that $\frac{p_n}{p_1} \leq \alpha^{n-1}$ to bound the number n_i of jobs scheduled to M_i with $s_i > R^{-1}$ by:

$$n_i \;\leq\; (R+1) + \frac{n}{\log_\alpha\left(\frac{Rs_i}{Rs_i - 1}\right)}\,.$$

We know that $s_i = \alpha^{-i}$ and $n = \sum_{i=1}^{n} n_i = \sum_{i=1}^{\lfloor \log_\alpha R \rfloor} n_i$, we combine this with the bound on n_i to obtain:

$$1 \leq \frac{(R+1)}{n}\left\lfloor \frac{\ln(R)}{\ln(\alpha)} \right\rfloor + \sum_{i=1}^{\left\lfloor \frac{\ln(R)}{\ln(\alpha)} \right\rfloor} \frac{\ln(\alpha)}{-\ln(1 - \alpha^i R^{-1})}\,.$$

Now we bound this sum by an appropriate integral and take the limit $n \to \infty$ and $\alpha \to 1$. We obtain the following inequality:

$$1 \leq \int_0^1 \frac{\ln(R)}{-\ln(1 - R^{-x})} dx\,.$$

Solving this inequality numerically gives $R > 2.564$. Thus we have that no algorithm for deterministic nonpreemptive scheduling on related machines can be 2.564-competitive.

References

[1] Susanne Albers. On randomized online scheduling. In *Proc. 34th Symp. Theory of Computing (STOC)*, pages 134–143. ACM, 2002.

[2] Yair Bartal, Amos Fiat, Howard Karloff, and Rakesh Vohra. New algorithms for an ancient scheduling problem. *J. Comput. Systems Sci.*, 51:359–366, 1995.

[3] Piotr Berman, Moses Charikar, and Marek Karpinski. On-line load balancing for related machines. *J. Algorithms*, 35:108–121, 2000.

[4] Bo Chen, Andre van Vliet, and Gerhard J. Woeginger. Lower bounds for randomized online scheduling. *Inform. Process. Lett.*, 51:219–222, 1994.

[5] Bo Chen, Andre van Vliet, and Gerhard J. Woeginger. New lower and upper bounds for on-line scheduling. *Oper. Res. Lett.*, 16:221–230, 1994.

[6] Bo Chen, Andre van Vliet, and Gerhard J. Woeginger. An optimal algorithm for preemptive on-line scheduling. *Oper. Res. Lett.*, 18:127–131, 1995.

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

[8] György Dósa and Yong He. Semi-online algorithms for parallel machine scheduling problems. *Computing*, 72:355–363, 2004.

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

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

[11] Leah Epstein. Bin stretching revisited. *Acta Inform.*, 39:97–117, 2003.

[12] Leah Epstein and Lene M. Favrholdt. Optimal non-preemptive semi-online scheduling on two related machines. In *Proc. 27th Symp. on Mathematical Foundations of Computer Science (MFCS)*, volume 2420 of *Lecture Notes in Comput. Sci.*, pages 245–256. Springer, 2002.

[13] Leah Epstein and Lene M. Favrholdt. Optimal preemptive semi-online scheduling to minimize makespan on two related machines. *Oper. Res. Lett.*, 30:269–275, 2002.

[14] Leah Epstein, John Noga, Steven S. Seiden, Jiří Sgall, and Gerhard J. Woeginger. Randomized on-line scheduling for two uniform machines. *J. Sched.*, 4:71–92, 2001.

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

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

[17] Ulrich Faigle, Walter Kern, and Gyorgy Turán. On the performane of online algorithms for partition problems. *Acta Cybernet.*, 9:107–119, 1989.

[18] Rudolph Fleischer and Michaela Wahl. On-line scheduling revisited. *J. Sched.*, 3:343–353, 2000.

[19] Teofilo F. Gonzales and Sartaj Sahni. Preemptive scheduling of uniform processor systems. *J. ACM*, 25:92–101, 1978.

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

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

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

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

[24] Yong He and Guochuan Zhang. Semi on-line scheduling on two identical machines. *Computing*, 62:179–187, 1999.

[25] Dorit S. Hochbaum and David Shmoys. A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach. *SIAM J. Comput.*, 17, 1988.

[26] Dorit S. Hochbaum and David B. Shmoys. Using dual approximation algorithms for scheduling problems: Theoretical and practical results. *J. ACM*, 34:144–162, 1987.

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

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

[29] H. Kellerer, V. Kotov, M. G. Speranza, and Z. Tuza. Semi on-line algorithms for the partition problem. *Oper. Res. Lett.*, 21:235–242, 1997.

[30] R. McNaughton. Scheduling with deadlines and loss functions. *Management Sci.*, 6:1–12, 1959.

[31] John F. Rudin, III. *Improved Bound for the Online Scheduling Problem*. PhD thesis, The University of Texas at Dallas, 2001.

[32] John F. Rudin, III and R. Chandrasekaran. Improved bound for the online scheduling problem. *SIAM J. Comput.*, 32:717–735, 2003.

[33] Steve Seiden, Jiří Sgall, and Gerhard J. Woeginger. Semi-online scheduling with decreasing job sizes. *Oper. Res. Lett.*, 27:215–221, 2000.

[34] Steve S. Seiden. Randomized online multiprocessor scheduling. *Algorithmica*, 28:173–216, 2000.

[35] Jiří Sgall. A lower bound for randomized on-line multiprocessor scheduling. *Inform. Process. Lett.*, 63:51–55, 1997.

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

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

[38] Tomáš Tichý. Randomized on-line scheduling on 3 processors. *Oper. Res. Lett.*, 32:152–158, 2004.

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

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

List of publications

The doctoral thesis is a superset of following results. First result was the semi-online algorithm for known optimal makespan in the author's master thesis and it was published on the international conference STACS [c1] and later also in Journal of Scheduling [j1]. This result was later generalized to an optimal online algorithm [c2] (conference ESA) and [j2] (journal Algorithmica). At last it was further generalized to the whole class of semi-online problems in [c3] (conference STACS)) and [j3] (journal Theory of Computer Systems). The special cases of a small number of machines was published on the conference PPAM [c4].

Publications related to the thesis

[j1] Tomáš Ebenlendr and Jiří Sgall. Optimal and online preemptive scheduling on uniformly related machines. *Journal of Scheduling*, 12:517–527, 2009.

[c1] Tomáš Ebenlendr and Jiří Sgall. Optimal and online preemptive scheduling on uniformly related machines. In *Proc. 21st Symp. on Theoretical Aspects of Computer Science (STACS)*, volume 2996 of *Lecture Notes in Comput. Sci.*, pages 199–210. Springer, 2004.

Conference version of [j1].

[j2] Tomáš Ebenlendr, Wojciech Jawor, and Jiří Sgall. Preemptive online scheduling: Optimal algorithms for all speeds. *Algorithmica*, 53:504–522, 2009.

[c2] Tomáš Ebenlendr, Wojciech Jawor, and Jiří Sgall. Preemptive online scheduling: Optimal algorithms for all speeds. In *Proc. 14th European Symp. on Algorithms (ESA)*, volume 4168 of *Lecture Notes in Comput. Sci.*, pages 327–339. Springer, 2006.

Conference version of [j2].

[j3] Tomáš Ebenlendr and Jiří Sgall. Semi-online preemptive scheduling: One algorithm for all variants. *Theory of Computing Systems*, pages 1–37, 2010.

[c3] Tomáš Ebenlendr and Jiří Sgall. Semi-online preemptive scheduling: One algorithm for all variants. In *Proc. 26th Symp. on Theoretical Aspects of Computer Science (STACS)*, volume 3 of *LIPIcs*, pages 346–360. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2009.

Conference version of [j3].

[c4] Tomáš Ebenlendr. Semi-online Preemptive Scheduling: Study of Special Cases. In *Proc. 8th Parallel Processing and Applied Mathematics*, volume 6068 of *Lecture Notes in Comput. Sci.*, pages 11–20. Springer, 2010.

Other publications

[c5] Jihuan Ding, Tomáš Ebenlendr, Jiří Sgall, and Guochuan Zhang. Online scheduling of equal-length jobs on parallel machines. In *Proc. 15th European Symp. on Algorithms (ESA)*, volume 4698 of *Lecture Notes in Comput. Sci.*, pages 427–438. Springer, 2007.

[c6] Tomáš Ebenlendr, John Noga, Jiří Sgall, and Gerhard Woeginger. A note on semi-online machine covering. In *Proc. 3rd International Workshop in Approximation and Online Algorithms*, volume 3879 of *Lecture Notes in Comput. Sci.*, pages 110–118. Springer, 2006.

[c7] Tomáš Ebenlendr, Marek Krčál, and Jiří Sgall. Graph balancing: a special case of scheduling unrelated parallel machines. In *Proc. 19th Symp. on Discrete Algorithms (SODA)*, pages 483–490. ACM/SIAM, 2008.

[c8] Tomáš Ebenlendr and Jiří Sgall. A Lower Bound for Scheduling of Unit Jobs with Immediate Decision on Parallel Machines. In *Proc. 6rd International Workshop in Approximation and Online Algorithms*, volume 5426 of *Lecture Notes in Comput. Sci.*, pages 43–52. Springer, 2009.