Semi-Online Scheduling Revisited*

Susanne Albers† Matthias Hellwig‡

Abstract

Makespan minimization on m identical machines is a fundamental scheduling problem. The goal is to assign a sequence of jobs, each specified by a processing time, to parallel machines so as to minimize the maximum completion time of any job. Deterministic online algorithms achieve a competitive ratio of about 1.92. Due to this relatively high competitiveness and the lack of progress in the area of randomized online strategies, recent research has investigated scenarios where the online constraint is relaxed.

We study *semi-online scheduling* where at any time an online scheduler knows the sum of the jobs' processing times. This problem relaxation is well motivated by practical applications. The best known semi-online algorithm achieves a competitive ratio of 1.6 (Cheng, Kellerer, Kotov, 2005). The best known lower bound is equal to 1.565 (Angelelli, Nagy, Speranza, Tuza, 2004).

In this paper we present two contributions for semi-online scheduling. We develop an improved lower bound showing that no deterministic semi-online algorithm can attain a competitive ratio smaller than 1.585. This significantly reduces the gap between the previous lower bound and the upper bound of 1.6. Secondly we present a new semi-online algorithm that is based on an approach different from that of previous strategies. The algorithm is 1.75-competitive and hence does not achieve the best possible competitiveness. However, our algorithm is extremely simple and, unlike previous strategies, does not resort to job classes. The algorithm is more in the spirit of online algorithms not using any extra information. Hence our upper bound highlights the additional power of a small piece of advice when provided to an online algorithm.

Keywords: Competitive analysis, makespan minimization, online computation.

1 Introduction

Makespan minimization on parallel machines is a fundamental and extensively studied scheduling problem with a considerable body of literature published over the last forty years. In the basic problem setting we are given m identical parallel machines. A sequence of jobs $\sigma = J_1, \ldots, J_n$ has to be scheduled non-preemptively on these machines. Each job J_i is specified by an individual processing time p_i, $1 \leq i \leq n$. The goal is to minimize the *makespan*, i.e. the maximum completion time of any job in the schedule.

The performance of offline algorithms and deterministic online algorithms is well understood. In the offline scenario the entire job sequence is known in advance. Computing optimum schedules is NP-hard [17]. Hochbaum and Shmoys devised a famous polynomial time approximation scheme [20]. In the online scenario the jobs arrive one by one. Whenever a new job J_i arrives, its processing time p_i is known. However future jobs J_k, with $k > i$, and their processing times are unknown. Job J_i has to be scheduled irrevocably on one of the machines before the next job arrives. Following Sleator and Tarjan [26] an

*Work supported by the German Research Foundation.

†Department of Computer Science, Humboldt-Universität zu Berlin, Unter den Linden 6, 10099 Berlin. albers@informatik.hu-berlin.de

‡Department of Computer Science, Humboldt-Universität zu Berlin, Unter den Linden 6, 10099 Berlin. mhellwig@informatik.hu-berlin.de

online algorithm A is called *c-competitive* if, for every σ, A's makespan is at most c-times the optimum makespan. A series of papers, published mostly in the 1990s, narrowed down the competitiveness of deterministic online strategies. More precisely, the best competitive ratio achievable by deterministic online algorithms is in the range $[1.88, 1.9201]$. Much less is known for randomized online algorithms. To date we know of no randomized strategy that provably beats deterministic ones, for all values of m.

Due to the relatively high competitiveness of deterministic online algorithms and the lack of progress in designing randomized strategies, recent research has focused on studying scenarios where the online constraint is relaxed. An online algorithm is provided with some information on the job sequence σ or has some extra ability to process it. More specifically, the following scenarios have been addressed. (1) An online algorithm has some information on the jobs' processing times or their total sum [3, 4, 5, 11, 22]. (2) An online algorithm knows the optimum makespan [7, 13]. (3) An online strategy may reorder jobs in σ to a limited extent [12].

In this paper we investigate basic online makespan minimization assuming that, additionally, the sum $S = \sum_{i=1}^{n} p_i$ of the jobs' processing times is known. The resulting setting is referred to as *semi-online scheduling*. Obviously, information S can help an algorithm to make scheduling decisions. We believe that S is a very mild form of advice. We make no assumptions regarding the processing times of individual jobs and generally do not restrict the family of allowed job sequences. Availability of advice S is also motivated by practical applications. In a parallel server system there usually exist fairly accurate estimates of the workload that arrives over a given time horizon. In a shop floor a scheduler typically accepts orders (tasks) of a targeted volume for a given time period, say a day or a week.

Previous work: We review the most important results relevant to our work and first address online scheduling without any extra information. Graham in 1966 [19] gave the first deterministic online algorithm. He showed that the famous *List* scheduling algorithm is $(2 - \frac{1}{m})$-competitive. Using new strategies, the competitive ratio was improved to $(2 - \frac{1}{m} - \epsilon_m)$ [16], where ϵ_m tends to 0 as $m \to \infty$, then to 1.986 [8] and 1.945 [21], and finally to 1.923 [1] and 1.9201 [16]. As for lower bounds, Faigle, Kern and Turan [14] showed that no deterministic online algorithm can achieve a competitiveness smaller than $2 - \frac{1}{m}$, for $m = 2$ and $m = 3$. For $m = 4$, Rudin and Chandrasekaran [24] recently gave a lower bound of $\sqrt{3} \approx 1.732$. For general m the lower bound was raised from 1.707 [14] to 1.837 [9] and 1.852 [1], and finally to 1.854 [18] and 1.88 [23].

For randomized online algorithms there exists a significant gap between the best known upper and lower bounds. For $m = 2$ machines, Bartal et al. [8] presented an algorithm that achieves an optimal competitive ratio of 4/3. For general m a 1.916-competitive algorithm was devised in [2]. The best known lower bound on the performance of randomized online algorithms is equal to $e/(e-1) \approx 1.581$.

We next consider semi-online scheduling, where an online algorithm knows the sum S of the jobs' processing times. The setting was first introduced by Kellerer et al. [22] who concentrated on $m = 2$ machines and gave a deterministic semi-online algorithm that achieves an optimal competitive ratio of $4/3$. Again for $m = 2$, two papers by Angelelli et al. [4, 5] refined the results assuming that, additionally, the job processing times are upper bounded by a known value. A setting with $m = 2$ uniform machines was studied in [6].

Semi-online scheduling on a general number m of identical machines was investigated by Angelelli, Nagy, Speranza and Tuza [3] and Cheng, Kellerer and Kotov [11]. The studies must have been done independently since none of the two papers cites the other one. Angelelli et al. [3] gave a deterministic semi-online algorithm that attains a competitiveness of $(1 + \sqrt{6})/2 \approx 1.725$ and showed a lower bound of 1.565, as $m \to \infty$, on the best possible competitive ratio of deterministic strategies. Cheng et al. [11] presented a deterministic 1.6-competitive semi-online algorithm and gave a lower bound of 1.5, for $m \geq 6$, on the competitiveness of deterministic strategies.

Our contribution: In this paper we present two contributions for semi-online

scheduling, complementing the existing results for classical makespan minimization. First we develop a new lower bound on the competitive ratio that can be achieved by deterministic semi-online algorithms. We show that the competitiveness is at least $c \geq 1.58504$, as $m \to \infty$. This ratio almost matches the upper bound of 1.6 presented by Cheng et al. [11]. Formally, the lower bound c is the root of the function $f(x) = 4x^3 - 8x^2 + 2x + 1$ that is in the range $[1.58504, 1.58505]$. We note that c is greater than $e/(e-1)$, which is a ratio often appearing in the analysis of online algorithms. Our lower bound proof consists of an explicit construction of a nemesis job sequence. It does not rely on numerical techniques or computer assisted proofs.

As a second result we present a very simple deterministic semi-online algorithm that is based on an approach different from that of previous strategies. The algorithms by Angelelli et al. [3] and Cheng et al. [11] both resort to job classes, i.e. incoming jobs are classified according to their processing times. The best known strategy by Cheng et al. [11] uses five job classes. The algorithm consists of sophisticated job packing schemes. Over the course of the algorithm and its analysis two scheduling phases with two associated stages and up to eight (or ten) machine types have to be considered.

Instead in this paper we develop an algorithm that does not resort to job classes. Our strategy is 1.75-competitive and hence does not achieve the best possible competitiveness. However, as mentioned above, the algorithm is very simple and can be stated in a single line (see Section 3). An incoming job is either scheduled on the least loaded machine or on the machine with the $\lceil m/2 \rceil$-th highest load. The decision which of the two machines to choose depends on the least loaded machine. The analysis of the algorithm relies on a potential function that keeps track of accumulated load on all the machines when the least loaded machine has a certain load. We remark that our scheduling algorithm is more in the spirit of online scheduling strategies not knowing S, which achieve a competitiveness around 1.92. Hence our upper bound also highlights the additional power of a small piece of advice when provided to an online algorithm.

Finally we show that our analysis is tight, i.e. our algorithm does not achieve a competitive ratio smaller than 1.75. Moreover, we observe that the algorithm can be extended easily to the scenario where an online scheduler knows the value of the optimum makespan.

2 A new lower bound

In this section we present a lower bound on the competitive ratio that can be achieved by deterministic semi-online algorithms. Consider the function $f(x) = 4x^3 - 8x^2 + 2x + 1$. This function has three real-valued roots, one of which is in the range $[1.58504, 1.58505]$. The lower bound is equal to this root. The other two roots of f are in the ranges $[-0.25, -0.24]$ and $[0.65, 0.66]$.

Theorem 1 *No deterministic semi-online algorithm can achieve a competitive ratio smaller than c as $m \to \infty$, where c is the root of $f(x) = 4x^3 - 8x^2 + 2x + 1$ with $c \in [1.58504, 1.58505]$.*

Proof. Let A be any deterministic semi-online algorithm. In the following c always denotes the value as specified in the statement of the theorem. The adversary presents a job sequence σ in which the total processing time of the jobs is equal to $S = m + 16c^2 - 12c - 16$. We remark that the expression $16c^2 - 12c - 16$ is upper bounded by 5.2. The exact structure of σ depends on the behavior of A but in each case the adversary uses at most four different processing times that we denote by p_i, $1 \leq i \leq 4$. A job with a processing time of p_i is also referred to as a p_i-job. The following construction of σ works for any $m > 8$.

Initially, the adversary presents $m - 4$ jobs of processing time $p_1 = 1$. If A assigns two of these jobs to the same machine, then the lower bound proof is simple: The adversary presents four additional jobs with a processing time of $p_1 = 1$ as well as m jobs with a processing time of $p_2 = (16c^2 - 12c - 16)/m$.

Algorithm A has a makespan of 2 while the adversary has a makespan of $1 + p_2$ only. In this case the ratio of A's makespan to the adversary's makespan can be arbitrarily close to 2, as $m \to \infty$.

In the following we concentrate on the case that A places the $m - 4$ p_1-jobs on different machines. At this point A has four empty machines. The adversary presents four jobs of processing time $p_2 = c - 1$. We distinguish three cases.

(1) Algorithm A assigns a p_2-job to a machine already containing a p_1-job.

(2) Algorithm A assigns the p_2-jobs only to machines not already containing p_1-jobs, and two p_2-jobs are placed on the same machine.

(3) Algorithm A assigns all the p_2-jobs to different machines, none of which already contains a p_1-job.

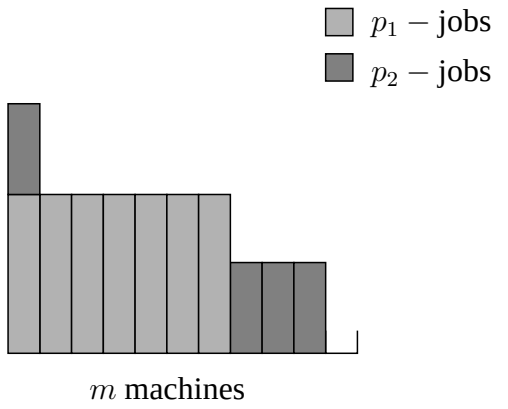

Figure 1: Case (1)

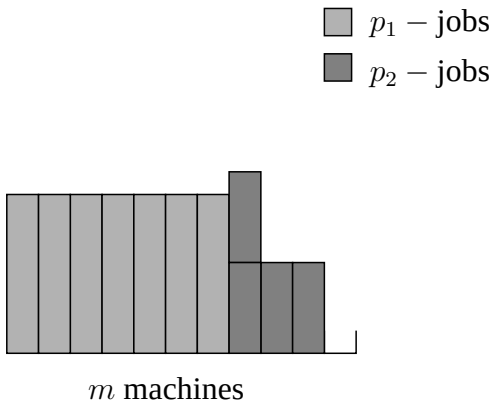

Figure 2: Case (2)

Figures 1 and 2 depict A's schedules in Cases (1) and (2), respectively. We next analyze the various cases.

Case (1): When the p_2-jobs are scheduled, A has a makespan of $p_1 + p_2 = 1 + c - 1 = c$ because there is a machine containing a p_1-job as well as a p_2-job. The adversary completes the request sequence by presenting four jobs of processing time $p_3 = 2 - c$ and m jobs of processing time $p_4 = (16c^2 - 12c - 16)/m$. The sum of the jobs' processing times is $S = (m-4) \cdot 1 + 4(c-1) + 4(2-c) + m(16c^2 - 12c - 16)/m = m + 16c^2 - 12c - 16$, as desired. The adversary constructs a schedule in which the p_1-jobs are assigned to different machines. Each p_2-job is paired with a p_3-job, yielding a total processing time of $c - 1 + 2 - c = 1$. Each such job pair is assigned to an empty machine. Finally each of the m machines receives a p_4-job. Thus the adversary's makespan is $1 + p_4$. The ratio of A's makespan to the adversary's makespan is $c/(1 + p_4)$ and this ratio tends to c as $m \to \infty$.

Case (2): As A has combined two p_2-jobs, one machine in A's schedule has a load of at least $2p_2 = 2(c - 1) > 1$. There are $m - 4$ additional machines containing a p_1-job and thus having a load of 1. Hence, when the p_2-jobs are scheduled, there exist at most three machines having a load smaller than 1. The adversary next reveals four jobs with a processing time of $p_3 = c$. Algorithm A must place at least one of them on a machine with a load of 1, incurring a makespan of $1+c$. The adversary completes the request sequence by presenting $m-8$ jobs of processing time $p_4 = (16c^2-20c-8)/(m-8)$. The sum of the jobs' processing times is $S = (m-4)\cdot 1+4(c-1)+4c+(m-8)(16c^2-20c-8)/(m-8) = m+16c^2-12c-16$, as claimed. The adversary can construct a schedule with a makespan of c: The $m - 4$ p_1-jobs are placed

on separate machines. Among these machines, four receive an additional p_2-job and $m-8$ get an extra p_4-job. The four p_3-jobs are placed separately on the four remaining empty machines. We have $p_1 + p_2 = c$ and $p_1 + p_4 < c$ because $p_4 < 0.5$, for $m > 8$. Thus no machine has a load greater than c. We conclude that the ratio of A's makespan to the adversary's makespan is at least $(1 + c)/c$ and this expression is greater than c, for our choice of c.

Case (3): Algorithm A assigns the $m - 4$ p_1-jobs and the four p_2-jobs to different machines so that, after the assignment, each machine contains exactly one job and there is no empty machine in the schedule. The adversary presents two jobs of processing time $p_3 = 2c(c - 1) - 1$. Again we distinguish two cases.

(a) Algorithm A assigns a p_3-job to a machine containing a p_1-job or assigns both p_3-jobs to the same machine containing a p_2-job.

(b) Algorithm A assigns the p_3-jobs to two machines containing a p_2-job.

Figure 3 depicts A's schedule in Case (3a) if a p_3-job is assigned to a machine containing a p_1-job. Figure 4 shows the schedule in Case (3b).

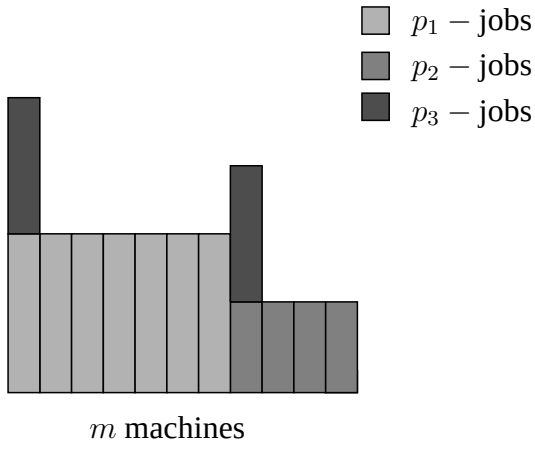

Figure 3: Case (3a)

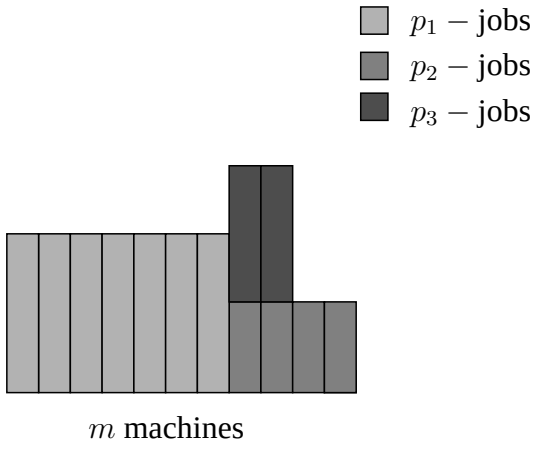

Figure 4: Case (3b)

Case (3a): When the p_3-jobs are scheduled, A has a makespan of at least $2c(c-1)$. This holds true if a p_3-job is assigned to a machine containing a p_1-job because $p_1+p_3 = 2c(c-1)$. This also holds true if both p_3-jobs are placed on the same machine containing a p_2-job because $p_2+2p_3 = c-1+2(2c(c-1)-1) = 2c(c-1)+(2c+1)(c-1)-2 > 2c(c-1)$ because $c > 1.5$. The adversary finishes the request sequence by sending $m-4$ jobs of processing time $p_4 = 6(2c(c-1)-1)/(m-4)$. The total processing time of the jobs is $S = (m-4)p_1+4p_2+2p_3+(m-4)p_4 = m-4+4(c-1)+2(2c(c-1)-1)+6(2c(c-1)-1) = m+16c^2-12c-16$. The adversary constructs the following schedule. Each p_1-job is assigned to a separate machine and will receive an additional p_4-job. The remaining four machines are used to schedule the p_2- and the p_3-jobs. More specifically, two machines receive two p_2-jobs each. The other two machines each receive a p_3-job. We have $2p_2 > p_3$ because this inequality is equivalent to $1 > 2(c-1)^2$ and is satisfied since $c < 1.6$. Moreover, for $m \geq 35$, we have $2p_2 > p_1 + p_4$ and the adversary's makespan is upper bounded by $2p_2 = 2(c-1)$. In summary, for $m \geq 35$ and hence for $m \to \infty$, the ratio of A's makespan to the adversary's makespan is $2c(c-1)/(2(c-1)) = c$.

Case (3b): Algorithm A assigns the two p_3-jobs to two machines containing a p_2-job and hence the load on these machines is $p_2 + p_3 = c - 1 + 2c(c-1) - 1 = (2c+1)(c-1) - 1 > 1$. Thus when the

p_3-jobs are scheduled, there are only two machines in A's schedule that have a load smaller than 1. The adversary then presents three final jobs with a processing time $p_4 = 2p_3 = 2(2c(c-1)-1)$. Again, we have a total processing time of $S = (m-4)p_1 + 4p_2 + 2p_3 + 3p_4 = m - 4 + 4(c-1) + 8(2c(c-1)-1) = m + 16c^2 - 12c - 16$. Algorithm A must schedule one of the p_4-jobs on a machine having a load of at least 1. Hence its makespan is at least $1 + p_4 = 1 + 2p_3$. On the other hand, the adversary can construct a schedule with a makespan of $p_4 = 2p_3$: The $m - 4$ p_1-jobs are assigned to different machines. Four of these machines receive an additional p_2-job, which results in a load of $1 + p_2 = c < p_4$. The remaining four machines are used to schedule the p_3- and p_4-jobs. One machine is assigned the two p_3-jobs. The other three machines each receive a p_4-job. Hence the ratio of A's makespan to the adversary's makespan is $(1 + p_4)/p_4 = 1 + 1/p_4 = 1 + 1/(2p_3)$. We have $2(c-1)p_3 - 1 = 0$ because $2(c-1)p_3 - 1 = 4c(c-1)^2 - 2(c-1) - 1 = 4c^3 - 8c^2 + 2c + 1$ and c is a root of the function $f(x) = 4x^3 - 8x^2 + 2x + 1$. Hence $2p_3 = 1/(c-1)$ and we conclude that the ratio of A's makespan to the adversary's makespan is $1 + 1/(2p_3) = 1 + (c-1) = c$. □ □

3 A semi-online algorithm without job classes

In this section we present a semi-online algorithm that is based on an approach different from that of previous strategies [3, 11] and does not rely on job classes. The algorithm is called *Light Load*, or *LL* for short, because it tries to keep the least loaded machine, and in fact $\lfloor m/2 \rfloor$ machines, lightly loaded. During the scheduling process the algorithm always maintains a list of the m machines sorted in non-increasing order of current load. The *load* of a machine is the sum of the processing times of the jobs currently assigned to that machine. At any time, given the sorted list $M_1, \ldots, M_m$ of machines, M_j denotes the machine with the j-th highest load, $1 \leq j \leq m$. In particular, M_1 is a machine with the highest load and M_m is a least loaded machine. Let l_j denote the load of M_j, $1 \leq j \leq m$. Moreover, let $j_0 = \lceil m/2 \rceil$. Of specific interest is machine M_{j_0} having the $\lceil m/2 \rceil$-th highest load.

Algorithm *LL* processes a job sequence $\sigma = J_1, \ldots, J_n$ as follows. While $l_m \leq 0.25\frac{S}{m}$, i.e. while the least loaded machine M_m has a load of at most $0.25\frac{S}{m}$, a new job is assigned to this machine M_m. When $l_m > 0.25\frac{S}{m}$, *LL* prefers to schedule an incoming job J_i on machine M_{j_0}. The algorithm checks if such an assignment is possible without exceeding a load of $1.75\frac{S}{m}$. If indeed $l_{j_0} + p_i \leq 1.75\frac{S}{m}$, J_i is scheduled on M_{j_0}; otherwise J_i is assigned to the least loaded machine M_m. A summary of the algorithm is given below.

Algorithm Light Load (LL): Job J_i is assigned to M_{j_0} if $l_m > 0.25\frac{S}{m}$ and $l_{j_0} + p_i \leq 1.75\frac{S}{m}$, and to M_m otherwise.

We explain the choice of the algorithm's parameters. The proof that *LL* is 1.75-competitive crucially depends on the definition $j_0 = \lceil m/2 \rceil$. We will show that if $l_m > 0.75\frac{S}{m}$ and hence a new job cannot necessarily be scheduled such that the resulting makespan is upper bounded by $1.75\frac{S}{m}$, then the job sequence contains $m+1$ large jobs of processing time greater than $0.5\frac{S}{m}$. In order to secure the existence of these jobs, we need to show that (a) M_{j_0} has a load of at most $1.25\frac{S}{m}$ and (b) M_m had a load of at most $0.25\frac{S}{m}$ over a long time horizon. The load bounds of (a) and (b) only hold if $j_0 = \lceil m/2 \rceil$. In this case it is possible to balance load between $\lceil m/2 \rceil$ heavily loaded and $\lfloor m/2 \rfloor$ lightly loaded machines. Any other choice of j_0 will lead to a higher competitive ratio. Moreover, *LL* works with a load bound of $0.25\frac{S}{m}$ for machine M_m. This ensures that an assignment of a small job of processing time at most $0.5\frac{S}{m}$ does not exceed the critical load threshold of $0.75\frac{S}{m}$.

Theorem 2 *Algorithm LL achieves a competitive ratio of 1.75.*

In the remainder of this section we prove the above theorem. We show that for any job sequence σ

$$LL(\sigma) \leq 1.75 \cdot OPT(\sigma) \tag{1}$$

where $LL(\sigma)$ and $OPT(\sigma)$ denote the makespan of *LL* and an optimal offline algorithm *OPT*, respectively. The proof is by induction on the length n of the job sequence σ. For job sequences consisting of a single job J_1 there is nothing to show because *LL* and *OPT* both have a makespan equal to the processing time p_1 of J_1. Suppose that (1) holds for job sequences of up to $n-1$ jobs. We will prove that (1) is also satisfied for sequences consisting of n jobs.

Let $\sigma = J_1, \ldots, J_n$ be an arbitrary job sequence of length n. By induction hypothesis *LL* schedules the first $n-1$ jobs such that a performance ratio of 1.75 is maintained, i.e. *LL* assigns each job such that its resulting makespan is at most 1.75 times the optimum makespan for the job sequence processed so far. In the following we investigate the assignment of J_n and prove that the scheduling step also maintains the desired performance guarantee. We concentrate on the case that the assignment of J_n causes an increase in *LL*'s makespan since otherwise there is nothing to show.

If *LL* schedules J_n on machine M_{j_0}, we are easily done because by the definition of the algorithm $l_{j_0} + p_n \leq 1.75\frac{S}{m}$. Since J_n is the last job of σ, the ratio $\frac{S}{m}$ is a lower bound on the optimum makespan and hence $l_{j_0} + p_n \leq 1.75 \cdot OPT(\sigma)$. Moreover, if *LL* schedules J_n on the least loaded machine M_m and $l_m \leq 0.75\frac{S}{m}$, the analysis is simple: If $p_n \leq \frac{S}{m}$, then *LL*'s resulting makespan is $l_m + p_n \leq 0.75\frac{S}{m} + \frac{S}{m} \leq 1.75\frac{S}{m} \leq 1.75 \cdot OPT(\sigma)$. If $p_n > \frac{S}{m}$, then $l_m + p_n \leq 0.75p_n + p_n = 1.75p_n \leq 1.75 \cdot OPT(\sigma)$ because the optimum makespan of σ cannot be smaller than the processing time of any job.

Therefore we can restrict ourselves to the case that *LL* schedules J_n on M_m and $l_m > 0.75\frac{S}{m}$ immediately before the assignment. Let $l_m = (0.75 + \epsilon)\frac{S}{m}$, for some $\epsilon > 0$. We have $\epsilon < 0.25$ because M_m is a least loaded machine and hence its load l_m is smaller than $\frac{S}{m}$. If we had $l_m \geq \frac{S}{m}$, then all machines would have a load of at least $\frac{S}{m}$ and the total load on the m machines before the arrival of J_n would be $m \cdot \frac{S}{m} = S$. Hence the total processing time of jobs in σ would be at least $S + p_n > S$, contradicting the fact that total processing volume equals S. Thus $0 < \epsilon < 0.25$. If $l_m + p_n \leq 1.75\frac{S}{m}$, we are again done. Hence we assume $l_m + p_n > 1.75\frac{S}{m}$. We obtain $p_n > 0.75\frac{S}{m}$ because, as just argued, $l_m < \frac{S}{m}$. Therefore, $p_n > (0.5 + \epsilon)\frac{S}{m}$.

In the following we will show that immediately before the assignment of J_n, each machine in *LL*'s schedule contains a job of processing time at least $(0.5 + \epsilon)\frac{S}{m}$. This implies that including J_n, the job sequence σ contains $m+1$ jobs of processing time at least $(0.5 + \epsilon)\frac{S}{m}$ each. Two of these jobs must be scheduled on the same machine in an optimal schedule and hence $OPT(\sigma) \geq (1 + 2\epsilon)\frac{S}{m}$. Using this property we can finish the proof of Theorem 2. If $p_n \leq (1 + 2\epsilon)\frac{S}{m}$, then *LL*'s resulting makespan is $l_m + p_n \leq (0.75 + \epsilon)\frac{S}{m} + (1 + 2\epsilon)\frac{S}{m} = (1.75 + 3\epsilon)\frac{S}{m} \leq 1.75(1 + 2\epsilon)\frac{S}{m} \leq 1.75 \cdot OPT(\sigma)$. If $p_n > (1+2\epsilon)\frac{S}{m}$, then the resulting makespan is $l_m + p_n \leq (0.75+\epsilon)\frac{S}{m} + p_n \leq (0.75+\epsilon)p_n/(1+2\epsilon) + p_n = (1.75 + 3\epsilon)p_n/(1 + 2\epsilon) \leq 1.75 \cdot p_n \leq 1.75 \cdot OPT(\sigma)$.

It remains to prove that immediately before the assignment of J_n each machine in *LL*'s schedule contains a job of processing time at least $(0.5 + \epsilon)\frac{S}{m}$. To this end we will show that while at most j_0 machines have a load of at least $(0.75 + \epsilon)\frac{S}{m}$ each, the least loaded machine has a load of not more than $0.25\frac{S}{m}$ (Lemma 1). Moreover, after that time, the load of machine M_{j_0} does not grow too large (Lemma 2).

In the following let time t be the point of time immediately after job J_t is scheduled, $1 \leq t \leq n$. At any time a machine in *LL*'s schedule is called *full* if its load is at least $(0.75 + \epsilon)\frac{S}{m}$. When J_n arrives, the least loaded machine and hence any machine in *LL*'s schedule is full. We consider the past scheduling steps of the jobs $J_1, \ldots, J_{n-1}$. Let t_j, $1 \leq j \leq m$, be the first point of time when exactly j machines are full in *LL*'s schedule, i.e. the assignment of J_{t_j} causes the j-th machine to become full. We have $1 \leq t_1 < \ldots < t_m \leq n-1$. Of particular interest is the time t_{j_0} when exactly j_0 machines are full. The next lemma states that at this time the least loaded machine M_m has a load of at most $0.25\frac{S}{m}$.

Lemma 1 *At time t_{j_0} there holds $l_m \leq 0.25\frac{S}{m}$.*

Proof. We will assume $l_m > 0.25\frac{S}{m}$ and derive a contradiction to the fact that the total processing time of jobs in σ is S. For any time t, $1 \leq t \leq n$, let $L(t)$ be the total load on the m machines, i.e. $L(t)$ is the sum of the processing times of the jobs $J_1, \ldots, J_t$. For the further analysis we need a potential function Φ whose definition is based on a machine set $\mathcal{M}$. Let $\mathcal{M}(t_{j_0})$ be the set of machines that are full at time t_{j_0}. At times $t > t_{j_0}$ we update this set whenever a machine becomes full. More specifically, for any time $t > t_{j_0}$, set $\mathcal{M}(t)$ is defined as follows. If $t \neq t_j$, for all $j = j_0 + 1, \ldots, m$, then the machine set remains unchanged and $\mathcal{M}(t) = \mathcal{M}(t-1)$. If $t = t_j$, for some j with $j_0 + 1 \leq j \leq m$, then $\mathcal{M}(t_j)$ is obtained from $\mathcal{M}(t_j - 1)$ by deleting the machine having the smallest current load in $\mathcal{M}(t_j - 1)$. In case of ties, the machine $M_j \in \mathcal{M}(t_j - 1)$ with the highest index in the machine ordering $M_1, \ldots, M_m$ at time $t_j - 1$ is chosen. Since at any time t_j, for $j = j_0 + 1, \ldots, m$, exactly one machine is deleted from the set and $j_0 = \lceil m/2 \rceil \geq m - \lceil m/2 \rceil = m - j_0$, set $\mathcal{M}(t)$ is non-empty at all times t with $t_{j_0} \leq t < t_m$. For any machine M_j and any time t let $l_j(t)$ denote its current load. Define

$$\Phi(t) = \sum_{M_j \in \mathcal{M}(t)} (l_j(t) - 0.75\tfrac{S}{m}).$$

Intuitively, Φ is the total load in excess to $0.75\frac{S}{m}$ on the machines of $\mathcal{M}(t)$. Since every machine of $\mathcal{M}(t_{j_0})$ has a load of at least $(0.75 + \epsilon)\frac{S}{m}$ and machine loads can only increase, Φ is always non-negative.

We next argue that at all times t with $t_{j_0} \leq t < t_m$, all machines of $\mathcal{M}(t)$ are among the j_0 machines having the highest load in *LL*'s current schedule. More formally, at any time t, $t_{j_0} \leq t < t_m$, let $\mathcal{H}(t)$ denote the set consisting of the j_0 machines $M_1, \ldots, M_{j_0}$ with highest current load. We will show $\mathcal{M}(t) \subseteq \mathcal{H}(t)$. We assume w.l.o.g. that whenever machines are sorted according to their load after a scheduling step, only the rank of the machine that received the new job changes. The relative order of all the other machines remains unchanged. In other words, machines having equal load appear in the same order before and after the scheduling step. This property can always be maintained by simply renumbering machines with equal load.

Obviously $\mathcal{M}(t_{j_0}) \subseteq \mathcal{H}(t_{j_0})$ is satisfied because at time t_{j_0} there exist exactly j_0 full machines in *LL*'s schedule and $\mathcal{M}(t_{j_0})$ contains all these machines. So suppose that $\mathcal{M}(t-1) \subseteq \mathcal{H}(t-1)$ holds, where $t_{j_0} < t \leq t_m - 1$, and consider the scheduling step at time t. Algorithm *LL* assigns the incoming job J_t either to machine M_{j_0} with the j_0-th highest load or to machine M_m with the smallest load. If J_t is placed on the current machine M_{j_0}, then the set $\mathcal{H}$ does not change and $\mathcal{H}(t-1) = \mathcal{H}(t)$. If J_t is assigned to the least loaded machine M_m and the machine does not become full, then again $\mathcal{H}(t-1) = \mathcal{H}(t)$ because set $\mathcal{H}$ only contains full machines. Hence set $\mathcal{H}$ can change only if J_t is assigned to M_m causing the machine to become full.

If at time t set $\mathcal{H}$ does not change, then $\mathcal{M}(t) \subseteq \mathcal{M}(t-1) \subseteq \mathcal{H}(t-1) = \mathcal{H}(t)$ and we are done. So assume that $\mathcal{H}$ does change. As argued in the last paragraph J_t is placed on the least loaded machine M_m and this machines becomes full. Thus $t = t_j$, for some j with $j_0 < j \leq m - 1$. At this time the former machine M_m joins $\mathcal{H}$ while the former machine M_{j_0} leaves the set. Note that M_{j_0} is a least loaded machine in $\mathcal{H}(t-1)$. At time t_j, the least loaded machine in $\mathcal{M}(t_j - 1)$ is removed from this set; in case of ties the highest indexed machine is chosen. Since both sets $\mathcal{M}(t_j - 1)$ and $\mathcal{H}(t_j - 1)$ lose least loaded machines, property $\mathcal{M}(t_j - 1) \subseteq \mathcal{H}(t_j - 1)$ implies $\mathcal{M}(t_j) \subseteq \mathcal{H}(t_j)$.

We will show that if $l_m > 0.25\frac{S}{m}$ at time t_{j_0}, then the following inequality holds for $j = j_0, \ldots, m$.

$$L(t_j) - \Phi(t_j) > 0.25S + j_0 \cdot 0.5\tfrac{S}{m} + (j - j_0)\tfrac{S}{m} \tag{2}$$

Using (2) for $j = m$ and observing again that the potential is non-negative, we obtain that at time t_m and hence before the arrival of J_n, the total processing time of jobs scheduled so far is at least

$$L(t_m) > 0.25S + \lceil \tfrac{m}{2} \rceil 0.5\tfrac{S}{m} + (m - \lceil \tfrac{m}{2} \rceil)\tfrac{S}{m} \geq 1.25S - (\tfrac{m}{2} + \tfrac{1}{2})0.5\tfrac{S}{m} = S - 0.25\tfrac{S}{m}.$$

Since J_n has a processing time of $p_n > (0.5+\epsilon)\frac{S}{m}$, the total processing time of jobs in σ is at least $L(t_m) + p_n > S + 0.25\frac{S}{m} > S$. We obtain the desired contradiction.

It remains to show (2), for $j = j_0, \ldots, m$, assuming that $l_m > 0.25\frac{S}{m}$ holds at time t_{j_0}. The proof is by induction on j. First consider j_0. At time t_{j_0} exactly j_0 machines are full and these machines $M_1, \ldots, M_{j_0}$ each have a load greater than $0.75\frac{S}{m}$. By assumption $l_m > 0.25\frac{S}{m}$. Hence machines $M_{j_0+1}, \ldots, M_m$ each have a load greater than $0.25\frac{S}{m}$ and the total load on these $m-j_0$ machines is greater than $(m-j_0)\cdot 0.25\frac{S}{m}$. We obtain that at time t_{j_0} the total load on the m machines is

$$\begin{aligned} L(t_{j_0}) &> \sum_{j=1}^{j_0} l_j(t_{j_0}) + \sum_{j=j_0+1}^{m} l_j(t_{j_0}) \\ &= j_0 \cdot 0.75\tfrac{S}{m} + \textstyle\sum_{j=1}^{j_0}(l_j(t_{j_0}) - 0.75\frac{S}{m}) + (m-j_0)0.25\frac{S}{m} \\ &= 0.25S + j_0 \cdot 0.5\tfrac{S}{m} + \Phi(t_{j_0}). \end{aligned}$$

The last equation holds because machines $M_1, \ldots, M_{j_0}$ form set $\mathcal{M}(t_{j_0})$. Inequality (2) then follows for $j = j_0$.

Next suppose that (2) holds for index j. We show that it is also satisfied for $j+1$. We first argue that

$$L(t) - \Phi(t) > 0.25S + j_0 \cdot 0.5\tfrac{S}{m} + (j-j_0)\tfrac{S}{m} \tag{3}$$

holds for any $t = t_j, t_j+1, \ldots, t_{j+1}-1$. By induction hypothesis the above inequality holds for $t = t_j$. At times t with $t_j < t < t_{j+1}$ the machine set $\mathcal{M}(t)$ does not change and is equal to $\mathcal{M}(t_j)$. At any time t the incoming job J_t increases the total load on the m machines by p_t, i.e. $L(t) = L(t-1) + p_t$. The potential Φ only increases by p_t if J_t is assigned to a machine in $\mathcal{M}(t_j)$. Hence the left hand side of (3) does not decrease at times $t = t_j+1, \ldots, t_{j+1}-1$.

At time t_{j+1} another machine becomes full. Since there exist already j_0 full machines and $j+1 > j_0$, an additional full machine can only be created if the incoming job $J_{t_{j+1}}$ is placed on the currently least loaded machine M_m. An assignment to the current machine M_{j_0} would not generate an additional full machine. By assumption, at time t_{j_0} and hence also at the current time $l_m > 0.25\frac{S}{m}$. Thus *LL* would prefer to schedule $J_{t_{j+1}}$ on machine M_{j_0}. Since this assignment is not performed, the resulting load would exceed $1.75\frac{S}{m}$, i.e. $l_{j_0}(t_{j+1}-1) + p_{t_{j+1}} > 1.75\frac{S}{m}$ and hence

$$p_{t_{j+1}} > \tfrac{S}{m} - (l_{j_0}(t_{j+1}-1) - 0.75\tfrac{S}{m}).$$

Machine M_{j_0} is a least loaded machine in $\mathcal{H}(t_{j+1}-1)$. At time t_{j+1} the least loaded machine in $\mathcal{M}(t_{j+1}-1)$ is removed from the set. As argued above $\mathcal{M}(t) \subseteq \mathcal{H}(t)$ for any t with $t_{j_0} \le t < t_m$. Hence the least loaded machine in $\mathcal{M}(t_{j+1}-1)$ has a load of at least $l_{j_0}(t_{j+1}-1)$. Thus at time t_{j+1} the potential decreases by at least

$$\Phi(t_{j+1}-1) - \Phi(t_{j+1}) \ge l_{j_0}(t_{j+1}-1) - 0.75\tfrac{S}{m}.$$

We obtain $p_{t_{j+1}} + \Phi(t_{j+1}-1) - \Phi(t_{j+1}) > \frac{S}{m}$ and, as desired,

$$\begin{aligned} L(t_{j+1}) - \Phi(t_{j+1}) &= L(t_{j+1}-1) - \Phi(t_{j+1}-1) + p_{t_j+1} + \Phi(t_{j+1}-1) - \Phi(t_{j+1}) \\ &\ge 0.25S + j_0 \cdot 0.5\tfrac{S}{m} + (j+1-j_0)\tfrac{S}{m}. \end{aligned}$$

The inductive step is complete. □ □

For the further analysis we need a second lemma.

Lemma 2 *At any time t with $t_{j_0} \leq t < t_m$ machine M_{j_0} in LL's schedule has a load of at most $(1.25 - \epsilon)\frac{S}{m}$.*

Proof. Suppose that at some time t, $t_{j_0} \leq t < t_m$, the machine M_{j_0} with the j_0-th highest load had a load greater than $(1.25 - \epsilon)\frac{S}{m}$. Thus at this time t and also at time t_m the j_0 machines with highest load in *LL*'s schedule had a total load greater than $j_0(1.25 - \epsilon)\frac{S}{m}$. At time t_m all machines of *LL* are full and the $m - j_0$ machines with the smallest load have a total load of at least $(m - j_0)(0.75 + \epsilon)\frac{S}{m}$. Thus at time t_m the total load on the m machines is greater than

$$j_0(1.25 - \epsilon)\frac{S}{m} + (m - j_0)(0.75 + \epsilon)\frac{S}{m} \geq \lceil \frac{m}{2} \rceil 0.5\frac{S}{m} + 0.75S - \epsilon\frac{S}{m} \geq S - \epsilon\frac{S}{m}.$$

Including J_n, the total processing time of jobs in σ is greater than $S - \epsilon\frac{S}{m} + p_n > S$ because $p_n > (0.5 + \epsilon)\frac{S}{m}$, This contradicts the fact that the total processing time of jobs in σ is equal to S. □

We are now ready to identify m large jobs in *LL*'s schedule at time t_m. More specifically, we show that at any time t_j, for $j = 1, \ldots, m$, a job of processing time at least $(0.5 + \epsilon)\frac{S}{m}$ is scheduled.

By Lemma 1 at time t_{j_0} we have $l_m \leq 0.25\frac{S}{m}$. Hence at any time with $t \leq t_{j_0}$ we have $l_m \leq 0.25\frac{S}{m}$ and *LL* schedules an incoming job on the least loaded machine. At any time t_j with $j = 1, \ldots, j_0$ a machine becomes full and hence $l_m + p_{t_j} \geq (0.75 + \epsilon)\frac{S}{m}$. This implies $p_{t_j} \geq (0.75 + \epsilon)\frac{S}{m} - 0.25\frac{S}{m} = (0.5 + \epsilon)\frac{S}{m}$.

Next consider the times t_j with $j_0 < j \leq m$. At those times another full machine can only be created if the incoming job is scheduled on the least loaded machine. Let t^* be the first point of time at which the least loaded machine in *LL*'s schedule has a load greater than $0.25\frac{S}{m}$. As above we can show that at any time t_j with $t_{j_0} < t_j \leq t^*$ a job of processing time at least $(0.5+\epsilon)\frac{S}{m}$ is scheduled. Finally consider times t_j with $t^* < t_j \leq t_m$. The least loaded machine in *LL*'s schedule has a load greater than $0.25\frac{S}{m}$. Therefore, *LL* would prefer to place J_{t_j} on the machine with the j_0-th highest load. Since J_{t_j} is placed on the least loaded machine instead we have $l_{j_0}(t_j - 1) + p_{t_j} > 1.75\frac{S}{m}$. By Lemma 2, $l_{j_0}(t_j - 1) \leq (1.25 - \epsilon)\frac{S}{m}$ and hence $p_{t_j} > (0.5 + \epsilon)\frac{S}{m}$. This concludes the proof of Theorem 2.

We next provide a matching lower bound on the performance of *LL*.

Theorem 3 *Algorithm LL does not achieve a competitive ratio smaller than 1.75.*

Proof. For simplicity we assume that m is even. Moreover, let $m \geq 4$. Choose an ϵ with $0 < \epsilon < 1$. We prove that *LL* does not achieve a competitive ratio smaller than $1.75 - \epsilon$. Let $S = m$. Furthermore, let k be an integer satisfying $k \geq 1.75/\epsilon$ and set $p_1 = 1/(4k)$. An adversary first presents km jobs of processing time p_1. These jobs have a total processing time of $m/4$. Thus, while the p_1-jobs arrive, machine M_m in *LL*'s schedule has a load of at most $1/4 = 0.25\frac{S}{m}$ and each p_1-job is assigned to this least loaded machine M_m. Hence, when all the km p_1-jobs are scheduled, each of the m machine has a load of exactly $kp_1 = 0.25$.

Next the adversary presents $m/2$ jobs of processing time $p_2 = 0.5$ and $m/2$ jobs of processing time $p_3 = 1 - 2/m$. While these jobs are scheduled, the least loaded machine M_m in *LL*'s schedule has a load of exactly $0.25\frac{S}{m}$. Thus, again, any of these m jobs is placed on machine M_m. After the assignment of these jobs, each machine in *LL*'s schedule has a load of at least 0.75 because $p_3 = 1 - 2/m \geq 0.5$. The adversary finally reveals a job of processing time $p_4 = 1$ so that *LL*'s final makespan is 1.75.

The total processing time of all jobs is $kmp_1 + m/2 \cdot p_2 + m/2 \cdot p_3 + 1 = m/4 + m/4 + m/2(1 - 2/m) + 1 = m = S$, as desired. The adversary can construct a schedule whose makespan is upper bounded by $1 + p_1$: The $m/2$ jobs of processing time p_3 and the job of processing time p_4 are assigned to different machines. The $m/2$ jobs of processing time $p_2 = 0.5$ are combined to pairs and placed on

machines not yet occupied by the p_3- and p_4-jobs. If $m/2$ is odd, then one p_2-job is scheduled alone on a machine. Finally, each job of processing time p_1 is scheduled on a machine currently having the smallest load.

We conclude that the competitive ratio of *LL* is at least $1.75/(1+p_1)$ and this ratio is at least $1.75-\epsilon$ because $p_1 < \epsilon/1.75$. □

We finally observe that *LL* can be extended easily to the scenario where an online scheduler knows the value $OPT(\sigma)$ of the optimum makespan. In this case we just have to replace $\frac{S}{m}$ by $OPT(\sigma)$ in both the description and the analysis of the algorithm.

Corollary 1 *Algorithm LL achieve a competitive ratio of 1.75 if S/m is replaced by the value of the optimum makespan.*

4 Conclusion and open problems

In this paper we have studied makespan minimization in the setting where an online scheduler knows the sum of the jobs' processing times. An obvious open problem is to determine the exact competitive ratio that can be achieved in this scenario. However, since the gap between our lower bound of 1.585 and the best known upper bound of 1.6 [11] is small, a further improvement in this range might be hard to obtain. A more fruitful working direction is to analyze the best competitiveness attainable if an online scheduler known the value of the optimum makespan. Azar and Regev [7] showed a lower bound of 4/3. The algorithm by Cheng et al. [11] is also 1.6-competitive in this problem setting.

References

[1] S. Albers. Better bounds for online scheduling. *SIAM Journal on Computing*, 29:459-473, 1999.

[2] S. Albers. On randomized online scheduling. *Proc. 34th Annual ACM Symposium on Theory of Computing*, 134–143, 2002.

[3] E. Angelelli, A.B. Nagy, M.G. Speranza and Z. Tuza. The on-line multiprocessor scheduling problem with known sum of the tasks. *Journal of Scheduling*, 7:421–428, 2004.

[4] E. Angelelli, M.G. Speranza and Z. Tuza. Semi-on-line scheduling on two parallel processors with an upper bound on the items. *Algorithmica*, 37:243–262, 2003.

[5] E. Angelelli, M.G. Speranza and Z. Tuza. New bounds and algorithms for on-line scheduling: two identical processors, known sum and upper bound on the tasks. *Discrete Mathematics & Theoretical Computer Science*, 8:1–16, 2006.

[6] E. Angelelli, M.G. Speranza and Z. Tuza. Semi-online scheduling on two uniform processors. *Theoretical Computer Science*, 393:211–219, 2008.

[7] Y. Azar and O. Regev. On-line bin-stretching. *Theoretical Computer Science*, 268:17–41, 2001.

[8] Y. Bartal, A. Fiat, H. Karloff and R. Vohra. New algorithms for an ancient scheduling problem. *Journal of Computer and System Sciences*, 51:359–366, 1995.

[9] Y. Bartal, H. Karloff and Y. Rabani. A better lower bound for on-line scheduling. *Infomation Processing Letters*, 50:113–116, 1994.

[10] B. Chen, A. van Vliet and G.J. Woeginger. A lower bound for randomized on-line scheduling algorithms. *Information Processing Letters*, 51:219–222, 1994.

[11] T.C.E. Cheng, H. Kellerer and V. Kotov. Semi-on-line multiprocessor scheduling with given total processing time. *Theoretical Computer Science*, 337:134–146, 2005.

[12] M. Englert, D. Özmen and M. Westermann. The power of reordering for online minimum makespan scheduling. *Proc. 49th Annual IEEE Symposium on Foundations of Computer Science*, 603-612, 2008.

[13] L. Epstein. Bin stretching revisited. *Acta Informtica*, 39:97–117, 2003.

[14] U. Faigle, W. Kern and G. Turan. On the performance of on-line algorithms for partition problems. *Acta Cybernetica*, 9:107–119, 1989.

[15] R. Fleischer and M. Wahl. Online scheduling revisited. *Journal of Scheduling*, 3:343–353, 2000.

[16] G. Galambos and G. Woeginger. An on-line scheduling heuristic with better worst case ratio than Graham's list scheduling. *SIAM Journal on Computing*, 22:349–355, 1993.

[17] M.R. Garay and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.

[18] T. Gormley, N. Reingold, E. Torng and J. Westbrook. Generating adversaries for request-answer games. *Proc. 11th ACM-SIAM Symposium on Discrete Algorithms*, 564–565, 2000.

[19] R.L. Graham. Bounds for certain multi-processing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.

[20] D.S. Hochbaum and D.B. Shmoys. Using dual approximation algorithms for scheduling problems theoretical and practical results. *Journal of the ACM*, 34:144–162, 1987.

[21] D.R. Karger, S.J. Phillips and E. Torng. A better algorithm for an ancient scheduling problem. *Journal of Algorithms*, 20:400–430, 1996.

[22] H. Kellerer, V. Kotov, M.G. Speranza and Z. Tuza. Semi on-line algorithms for the partition problem. *Operations Research Letters*, 21:235–242, 1997.

[23] J.F. Rudin III. Improved bounds for the on-line scheduling problem. Ph.D. Thesis. The University of Texas at Dallas, May 2001.

[24] J.F. Rudin III and R. Chandrasekaran. Improved bounds for the online scheduling problem. *SIAM Journal on Computing*, 32:717–735, 2003.

[25] J. Sgall. A lower bound for randomized on-line multiprocessor scheduling. *Information Processing Letters*, 63:51–55, 1997.

[26] D.D. Sleator and R.E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28:202–208, 1985.