J Sched (2006) 9:433–452
DOI 10.1007/s10951-006-8497-6

Scheduling parallel jobs to minimize the makespan

Berit Johannes

Abstract We consider the NP-hard problem of scheduling parallel jobs with release dates on identical parallel machines to minimize the makespan. A parallel job requires simultaneously a prespecified, job-dependent number of machines when being processed. We prove that the makespan of any nonpreemptive list-schedule is within a factor of 2 of the optimal preemptive makespan. This gives the best-known approximation algorithms for both the preemptive and the nonpreemptive variant of the problem. We also show that no list-scheduling algorithm can achieve a better performance guarantee than 2 for the nonpreemptive problem, no matter which priority list is chosen.

List-scheduling also works in the online setting where jobs arrive over time and the length of a job becomes known only when it completes; it therefore yields a deterministic online algorithm with competitive ratio 2 as well. In addition, we consider a different online model in which jobs arrive one by one and need to be scheduled before the next job becomes known. We show that no list-scheduling algorithm has a constant competitive ratio. Still, we present the first online algorithm for scheduling parallel jobs with a constant competitive ratio in this context. We also prove a new information-theoretic lower bound of 2.25 for the competitive ratio of any deterministic online algorithm for this model. Moreover, we show that 6/5 is a lower bound for the competitive ratio of any deterministic online algorithm of the preemptive version of the model jobs arriving over time.

Keywords multiprocessor scheduling, parallel jobs, approximation algorithms, online algorithms, list scheduling, release dates

1. Introduction

Scheduling parallel jobs has recently gained considerable attention. The papers (Amoura et al., 1997; Błażewicz, Drabowski, and Węglarz, 1986; Chen and Miranda, 1999; Du and Leung, 1989; Feldmann, Sgall, and Teng, 1994; Jansen and Porkolab, 1999, 2000; Ludwig and Tiwari, 1994; Mu'alem and Feitelson, 1999; Turek, Wolf, and Yu, 1992) are just a small sample of work in this

B. Johannes (✉)
Institute of Theoretical Computer Science, ETH Zurich, 8092 Zurich, Switzerland
e-mail: beritj@inf.ethz.ch

area. In fact, the study of computer architectures with parallel processors has prompted the design and analysis of good algorithms for scheduling parallel jobs. Parallel jobs are often alternatively called multiprocessor tasks. We refer to Feitelson et al. (1997) for a comprehensive introduction to the scheduling aspect of parallel computing, and to Drozdowski (1996) for an overview of results on computational complexity and approximation algorithms; (Feitelson, 1997) contains a collection of further references.

We discuss the following class of scheduling problems. We are given m identical parallel machines and a set of n independent parallel jobs $j = 1, \ldots, n$. Each job j has a positive integer processing time p_j, which we also call its *length*. Job j simultaneously requires $m_j \leqslant m$ machines at each point in time it is in process. The positive integer m_j is also called the *width* of job j. Note that we assume that m_j is part of the input; in particular, jobs are nonmalleable. Moreover, each job j has a nonnegative integer release date r_j at which it becomes available for processing. Every machine can process at most one job at a time. The objective is to find a feasible schedule of minimal completion time; that is, the makespan is to be minimized. Whenever a machine falls idle or a job is released, a *list-scheduling algorithm* schedules from a given priority list the first job that is already released and that does not require more machines than are available. We consider both the preemptive and the nonpreemptive variants of this problem. If preemptions are allowed, a job may be interrupted at any point in time and continued later, possibly on a different set of machines. Hence, in *preemptive list-scheduling*, jobs with lower priority can be preempted by jobs with higher priority. We denote the preemptive problem by $\mathrm{P}|m_j, r_j, \mathrm{pmtn}|C_{\max}$ and the nonpreemptive problem by $\mathrm{P}|m_j, r_j|C_{\max}$, following the three-field notation introduced in Graham et al. (1979). We note that some authors use "size$_j$" instead of "m_j" to refer to parallel jobs of the nature described before; see, e.g., Drozdowski (1996). Occasionally, we shall also refer to problems featuring nonparallel jobs only; they arise as the special case in which $m_j = 1$ for all $j = 1, \ldots, n$. The associated short-hand notation is $\mathrm{P}|r_j, \mathrm{pmtn}|C_{\max}$ and $\mathrm{P}|r_j|C_{\max}$, respectively, and, in the absence of nontrivial release dates, $\mathrm{P}|\mathrm{pmtn}|C_{\max}$ and $\mathrm{P}||C_{\max}$.

Since both the parallel job-scheduling problems considered here are NP-hard, we are interested in approximation algorithms. An *α-approximation algorithm* for a minimization problem is a polynomial-time algorithm that constructs for any instance a solution of value at most α times the optimal value; α is also called the *performance guarantee* of the algorithm. Motivated by the application context of parallel job scheduling, we also study (deterministic) online algorithms. While we shall consider different online scenarios, we always measure the quality of an online algorithm in terms of its competitive ratio. An online algorithm is *α-competitive* if it produces for any instance a solution of value at most α times the value of an offline optimum.

While preemptive scheduling of parallel jobs is NP-hard, even in the absence of release dates (Drozdowski, 1994), preemptive scheduling of nonparallel jobs (that is, $m_j = 1$) can be solved in polynomial time (McNaughton, 1959; Horn, 1974). For the strongly NP-hard problem of scheduling nonparallel jobs without release dates, $\mathrm{P}\|C_{\max}$, Graham (1966) showed that every list-scheduling algorithm is a $(2 - 1/m)$-approximation algorithm. Gusfield (1984) and Hall and Shmoys (1989) observed that Graham's result holds for nonparallel jobs with release dates, $\mathrm{P}|r_j|C_{\max}$, as well. If nonparallel jobs are scheduled in nonincreasing order of their lengths, list-scheduling is a $((4m - 1)/3m)$-approximation algorithm for $\mathrm{P}\|C_{\max}$, see (Graham, 1969), and a 3/2-approximation algorithm for $\mathrm{P}|r_j|C_{\max}$, see Chen and Vestjens (1997). In contrast thereto, we provide an instance of parallel jobs with release dates that is simultaneously bad for all possible priority lists; that is, no variant of list-scheduling can achieve a better performance guarantee than 2 for $\mathrm{P}|m_j, r_j|C_{\max}$. Hochbaum and Shmoys (1987) and Hall and Shmoys (1989) gave polynomial-time approximation schemes for the problems $\mathrm{P}||C_{\max}$ and $\mathrm{P}|r_j|C_{\max}$, respectively. In contrast, there is no approximation algorithm with a performance guarantee better than 3/2 for $\mathrm{P}|m_j|C_{\max}$, unless P = NP, as we point out in Section 2.

It follows from the work of Garey and Graham (1975) on project scheduling with resource constraints that list-scheduling has performance guarantee 2 for scheduling parallel jobs without release dates, $P|m_j|C_{\max}$. Turek, Wolf, and Yu (1992) presented a direct, simplified proof of this result. Feldmann, Sgall, and Teng (1994) observed that the length of a nonpreemptive list-schedule is actually at most $2 - 1/m$ times the optimal preemptive makespan, if no release dates are present. This implies that there is a $(3 - 1/m)$-approximation algorithm for both, $P|m_j, r_j, \text{pmtn}|C_{\max}$ and $P|m_j, r_j|C_{\max}$, since a delay of the start times of all jobs by the maximal release date increases the performance guarantee by at most 1 (see also Mu'alem and Feitelson, 1999).

In Section 4, we show that any nonpreemptive list-scheduling algorithm produces a schedule with makespan at most twice the makespan of an optimal preemptive schedule. This leads at the same time to a 2-approximation algorithm for both $P|m_j, r_j|C_{\max}$ and $P|m_j, r_j, \text{pmtn}|C_{\max}$. We show that in contrast to scheduling with nonparallel jobs, this approximation ratio cannot be improved upon by applying different priority strategies.

Independently of our work, first presented in Johannes (2001), Naroska and Schwiegelshohn (2002) also showed that list-scheduling is a 2-approximation algorithm for these problems. While Naroska and Schwiegelshohn prove this result by induction over the number of different release dates, we directly compare the structure of a list-schedule with that of an optimal preemptive schedule, which results in a somewhat more perspicuous proof. As a matter of fact, we use our techniques to obtain additional results.

For $P|m_j, r_j, \text{pmtn}|C_{\max}$, we show that preemptive m_j-list-scheduling, that is, preemptive list-scheduling where jobs are in order of nonincreasing widths, generates a schedule that is at most $2 - 1/m$ times as long as an optimal preemptive schedule. Although this result is essentially outperformed by the main result, we will present it anyway in Section 3, because its proof is short, and it provides a first idea of the techniques used in the main proof.

While it is not difficult to see that list-scheduling algorithms that do not depend on job characteristics (e.g., the length) also work in the online settings *unknown running times* and *jobs arriving over time* with corresponding competitive ratios, no list-scheduling algorithm has a constant competitive ratio in the context of *scheduling jobs one by one*. In Section 5, we present the first online algorithm with a constant competitive ratio for scheduling parallel jobs one by one. We also show that no deterministic online algorithm has a competitive ratio smaller than 2.25. For the preemptive version of the online model *jobs arriving over time*, we show that no deterministic online algorithm can achieve a better competitive ratio than 6/5. So far, no lower bound is known for this problem.

A problem closely related to $P|m_j|C_{\max}$ is strip packing, sometimes also called orthogonal packing in two dimensions. In contrast to the model considered here, machines assigned to a job need to be contiguous in a solution to the strip-packing problem. Turek, Wolf, and Yu (1992) pointed out that there is indeed an advantage to using noncontiguous machine assignments. From a parallel computer architecture perspective, $P|m_j|C_{\max}$ corresponds to scheduling on a PRAM, while strip packing is equivalent to scheduling on a linear array of processors (Ludwig and Tiwari, 1994). The strip-packing problem was first posed by Baker, Coffman, and Rivest (1980). Various authors proposed approximation algorithms with performance guarantees 3 (Baker, Coffman, and Rivest, 1980; Coffman et al., 1980; Golan, 1981), 2.7 (Coffman et al., 1980), 2.5 (Sleator, 1980), and 2 (Steinberg, 1997), respectively. Kenyon and Remila (1996) gave an asymptotic fully polynomial-time approximation scheme when m is fixed.

If no network topology is specified (PRAM) and the number m of machines is fixed, $Pm|m_j, \text{pmtn}|C_{\max}$ can be solved as a linear programming problem in polynomial time (Błażewicz, Drabowski, and Węglarz, 1986). Jansen and Porkolab (2000) presented an algorithm with running time $O(n) + \text{poly}(m)$ if m is not fixed, thereby showing that it cannot be

strongly NP-hard, unless P = NP. They also gave a polynomial-time approximation scheme for the nonpreemptive problem with a fixed number of machines, P$m|m_j|C_{\max}$, see (Jansen and Porkolab, 1999). Du and Leung (1989) showed that this problem is strongly NP-hard for any fixed $m \geqslant 5$.

2. Preliminaries

In this section, we briefly discuss the limits of approximability for some parallel job-scheduling problems as well as the running time of list-scheduling algorithms.

The problem of nonpreemptively scheduling parallel jobs of length 1 to minimize the makespan, P$|m_j, p_j = 1|C_{\max}$, is equivalent to the strongly NP-hard BIN PACKING problem, as was observed by Błażewicz, Drabowski, and Węglarz (1986). It follows that there is no approximation algorithm for scheduling parallel jobs to minimize the makespan with a performance guarantee better than 3/2, unless P = NP. Moreover, in contrast to BIN PACKING, item sizes (i.e., lengths of jobs) can be scaled. Therefore, we can state the following theorem.

Theorem 2.1. *There is no polynomial-time algorithm that produces for every instance of $P|m_j|C_{\max}$ a schedule with makespan at most $\alpha C^*_{\max} + \beta$ with $\alpha < 3/2$ and β being a polynomial in n, unless P = NP. Here, $C^*_{\max}$ denotes the optimal makespan.*

Proof: Let us consider the NP-complete decision version of the problem P2$\|C_{\max}$: Is it possible to schedule the n nonparallel jobs of a given instance I_1 on two identical parallel machines such that the makespan is at most T? We transform I_1 into an instance I_2 of the problem P$|m_j, p_j = p|C_{\max}$ by introducing T machines and n jobs. Each job i with length p_i in I_1 will be transformed into a job j in I_2 with width $m_j = p_i$ and length p for some $p > 0$. Hence, a feasible schedule for the instance I_2 has a makespan of at most $2p$ if and only if I_1 is a yes-instance. If there was a polynomial-time algorithm A for the problem P$|m_j, p_j = p|C_{\max}$ with $C^A_{\max} \leqslant \alpha C^*_{\max} + \beta$ with $\alpha < 3/2$ and β being a polynomial in n, we would have the means to recognize any yes-instance of P2$\|C_{\max}$ in polynomial time by choosing $p > \frac{\beta}{3-2\alpha}$. □

Li (1999) gave a polynomial-time algorithm with asymptotic performance guarantee 31/18 for P$|m_j|C_{\max}$. Drozdowski (1994) observed that if all jobs have length 1, then the existence of a preemptive schedule of length 2 implies the existence of a nonpreemptive schedule of length 2 as well. Thus, preemptive scheduling of parallel jobs with length 1 and therefore P$|m_j$, pmtn$|C_{\max}$ do not have a better than 3/2-approximation algorithm either, unless P = NP.

It is well known that list-scheduling algorithms for classic (i.e., nonparallel) job-scheduling problems can easily be implemented in polynomial time. However, one needs to be more careful for parallel job-scheduling problems because we may not assume that the number m of machines is at most the number n of jobs. Therefore, any polynomial-time scheduling algorithm that outputs job-machine assignments has to find a compact way of encoding this output since not m, but $\log m$ is part of the input size (as machines are identical). The following lemma ensures that there is always an assignment of the jobs to the machines such that no job is split over too many machines, and we may therefore safely restrict ourselves to algorithms that specify the starting times of jobs.

Lemma 2.2. *Let S be a feasible schedule for an instance of the problem $P|m_j, r_j|C_{\max}$, given by job starting times. Then, there is a polynomial-time algorithm that computes a feasible assignment*

of jobs to machines (without changing the starting times of jobs). In particular, the job-machine assignment can be represented with polynomial size.

Proof: Let $t_1 < t_2 < \cdots < t_z$ be the different starting times of the jobs in S. Let $i_1 < i_2 < \cdots < i_{m_t}$ be the idle machines in S at time t and let $j_1, j_2, \ldots, j_{n_t}$ be the jobs that start at time $t, t = t_1, \ldots, t_z$. We assign the jobs $j_1, \ldots, j_{n_t}$ to the machines $i_1, \ldots, i_{m_t}$ in the following way. Job j_1 is assigned to the first m_{j_1} machines in $\{i_1, \ldots, i_{m_t}\}$, job j_2 is assigned to the next m_{j_2} machines in $\{i_1, \ldots, i_{m_t}\}$, and so on. Let f_t be the number of machine-intervals that exist or are created at time t, for $t = 1, \ldots, z$. A machine-interval is a set of consecutive machines (in the order $1, 2, \ldots, m$) of maximal cardinality such that all machines in this set are processing the same job. In particular, a machine-interval is completely specified by (the index of) its first and its last machine. Hence, we will output for every $t \in \{t_1, \ldots, t_z\}$ the set of machine-intervals with the corresponding jobs.

At time $t_1 = 0$, n_{t_1} jobs start. Therefore, $f_{t_1} \leqslant n_{t_1} + 1$. For every additional split of a machine-interval at time t_{k+1}, we need a new job that starts at time t_{k+1}. Hence, for all $1 \leqslant k \leqslant z$ we have $f_{t_z} \leqslant 1 + \sum_{t=t_1}^{t_z} n_t = n + 1$. We conclude that for every point in time $t = t_1, \ldots, t_z$ there are no more than $n + 1$ machine-intervals and thus the size of the output is polynomial in the number of jobs. □

In particular, the nonpreemptive list-scheduling algorithm described in Section 1 computes a feasible schedule (including job-machine assignments) in polynomial time. For preemptive list-scheduling, it is not necessary to invoke Lemma 2.2. At any event (release date or completion time of a job,) one can simply interrupt the processing of all jobs and then newly assign each job (highest priorities first) to consecutive machines. Hence, the total number of preemptions is bounded from above by $2n$. Thus, the job-machine assignments can again be compactly described. Note also that preemptive list-scheduling only preempts at integer points in time, whereas an optimal preemptive schedule may preempt at any time.

3. Preemptive list-scheduling of parallel jobs with release dates

We now present a 2-approximation algorithm for scheduling parallel jobs with release dates when preemptions are allowed. More specifically, we prove that preemptive m_j-list-scheduling delivers for all instances of $\mathrm{P}|m_j, r_j, \mathrm{pmtn}|C_{\max}$ a schedule that is at most $2 - 1/m$ times as long as an optimal preemptive schedule. The algorithm works as follows. At every decision point (i.e., release date or completion time) all currently running jobs are preempted. Then, the already released, but not yet completed jobs are considered in order of nonincreasing widths m_j, and as many of them are greedily assigned to the machines as feasibly possible.

Theorem 3.1. *Consider an instance of $P|m_j, r_j, pmtn|C_{\max}$. The length of the schedule constructed by the preemptive m_j-list-scheduling algorithm is at most $2 - 1/m$ times the optimal makespan $C^*_{\max}$.*

Proof: Let e be the job that determines the makespan in the preemptive list-schedule, and let C_e be its completion time. We divide the time horizon $(0, C_e]$ into intervals $(t, t + 1]$ of length one ($t = 0, 1, \ldots, C_e - 1$). We call such an interval *time-slot*. We distinguish two cases.

Case 1. $m_e > m/2$.
Let r_z be the minimal point in time after which every time-slot contains a wide job (i.e., a job j with $m_j > m/2$) in the m_j-list-schedule. It follows from the definition of m_j-list-scheduling that r_z is the minimal release date of all jobs with $m_j > m/2$ that are scheduled in this last contiguous block of time-slots with wide jobs. Hence, r_z plus the total processing time of wide jobs in this block is a lower bound for $C^*_{\max}$. Thus, $C_e = r_z + (C_e - r_z) \leqslant C^*_{\max}$, and the list-schedule is optimal in this case.

Case 2. $m_e \leqslant m/2$.
Suppose $C_e > (2 - 1/m)C^*_{\max}$. Let r_z be the minimal point in time after which every time-slot contains a job j with $m_j \geqslant m_e$ in the list-schedule. By definition, r_z is the minimal release date of all jobs with $m_j \geqslant m_e$ that are scheduled in this last contiguous block of time-slots containing jobs at least as wide as e. Consequently, all these jobs have to be scheduled after r_z in the optimal schedule as well. Thus, the total load of jobs to be processed in a space of $(C^*_{\max} - r_z)m$ is strictly more than $(r_e - r_z)m_e + (C^*_{\max}(2 - 1/m) - r_e - p_e)(m - m_e + 1) + p_e m_e$. The first term in the summation results from jobs j with $m_j \geqslant m_e$ scheduled prior to the time at which e is released, and from the definition of r_z. The second term accounts for those time-slots after the release date of job e in which e is not scheduled. Finally, the third term is the load produced by e itself. A simple calculation shows that this load is too much to fit into the space provided by an optimal schedule between r_z and $C^*_{\max}$.

□

It is not hard to show that this approximation bound of $2 - 1/m$ for (preemptive) list-scheduling of (parallel) jobs to minimize makespan is tight. The following instance has already been proposed by Graham (1966) to show that list-scheduling for nonparallel jobs without release dates has no better performance guarantee than $2 - 1/m$.

Example 3.2. Consider an instance with $m^2 - m + 1$ unit-width jobs, each one with length 1, except for the last job in the list, which has length m. The resulting schedule has no preemptions and is of length $m - 1 + m = 2m - 1$, while the optimal schedule has makespan m.

4. Nonpreemptive list-scheduling of parallel jobs with release dates

The following result gives a universal performance guarantee of 2 for all nonpreemptive list-scheduling algorithms, regardless of which priority list is used. It holds for both the preemptive and the nonpreemptive version of the problem.

Theorem 4.1. *For every instance of the problems* $P|m_j, r_j, (pmtn)/C_{\max}$, *the length of the schedule constructed by any nonpreemptive list-scheduling algorithm is less than twice the optimal preemptive makespan.*

Proof: To prove the result, we directly compare the nonpreemptive list-schedule *LS* of a given instance with a preemptive optimal schedule *OPT*. Let $C^{LS}_{\max}$ be the makespan of *LS* and let $C^{OPT}_{\max}$ be the makespan of *OPT*. We partition the time horizon into periods of the same length, such that all jobs are started, preempted, restarted, and completed at the beginning or the end of a period. By scaling, we may assume that such a period, which we call *time-slot s*, starts at time $s - 1$ and ends at time s (for some nonnegative integer s). The load of a time-slot is the number of machines that are busy during this period. We define the *load* of a set of time-slots as the sum

of the loads in the individual time-slots. We refer to the part of a job that is scheduled in one time-slot as a *slice*. Let $r_0 < r_1 < \cdots < r_z$ be the different release dates of the given instance. The release date of a slice is the release date of its job. A slice of job j is *wide* if $m_j > m/2$, otherwise, it is called *small*. The proof will refer to slices of jobs instead of jobs since this is the level of granularity needed to prove the result. We will combine time-slots in *LS* to sets and then match sets to add up their contained load in a way that enables us to compare the load in *LS* to the load in *OPT*. Two disjoint sets of time-slots E_1 and E_2 in *LS* of equal size ($|E_1| = |E_2|$) can be *matched* if the load of any time-slot in E_1 plus the load of any time-slot in E_2 exceeds m, and hence the sum of the load of all slices in $E_1 \cup E_2$ exceeds $|E_1|m$. For instance, if a job j is released at time r_j but started only at time $s_j > r_j$ in our list-schedule, then the sum of the load in any time-slot between r_j and s_j and the load in any time-slot between s_j and the completion time C_j of j, is greater than m. Hence, any such two time-slots can be matched and therefore two sets of same size, consisting of time-slots between r_j and s_j, and time-slots between s_j and C_j, respectively, can also be matched. Also, if we know that every time-slot in two disjoint sets of the same size contains a wide job, then these two sets can be matched as well. Due to the fact that our sets of time-slots defined during the proof are not necessarily of the same size, we have occasionally to distinguish several cases. For instance, if we have two disjoint sets of time-slots E_1 and E_2, and any time-slot in E_1 can be matched with any time-slot in E_2, then depending on which set is greater, we might have a total load of more than $|E_1|m$ plus some remaining unmatched time-slots in E_2, or we have a total load of more than $|E_2|m$ and some remaining unmatched time-slots in E_1. For $k = 0, 1, \ldots, z$, let $D(r_k)$ be the set of time-slots in *LS* after r_k that contain wide slices, which are completed in *OPT* before r_k. In particular, $D(r_0) = \emptyset$. These sets have the properties described in Observations 1, 2, and 3. We denote by $S(r_k)$ the set of time-slots $\{1, \ldots, r_k\} \cup D(r_k)$, for each $k = 1, \ldots, z$. Let $S(r_0) = \emptyset$ and let $S(C^{LS}_{\max})$ be the set of all time-slots in *LS*; thus, $S(C^{LS}_{\max}) = \{1, \ldots, C^{LS}_{\max}\}$.

Observation 1. $D(r_h) \cap \{r_k + 1, \ldots, C^{LS}_{\max}\} \subseteq D(r_k)$ for $0 \leqslant h < k \leqslant z$.

Observation 2. For every set $D \subseteq D(r_k)$ we define the bipartite graph $B(D)$ in the following way. For every time-slot $h \in \{(r_k - |D| + 1), \ldots, r_k\}$ we introduce one node on the left side of the graph $B(D)$. For every time-slot $d \in D$ we introduce one node on the right side of the graph $B(D)$. A node h on the left side of $B(D)$ and a node d on the right side of $B(D)$ are adjacent if and only if the wide slice in the time-slot d is released before h, that is, $r_d \leqslant h - 1$. Then $B(D)$ contains a perfect matching.

Observation 3. The inequality $|D(r_k)| \leqslant r_k$ holds for all $k = 0, \ldots, z$.

Observation 4. For all $0 \leqslant h < k \leqslant z$ we have $(S(C^{LS}_{\max})\backslash S(r_k)) \cap (S(r_k)\backslash S(r_h)) = \emptyset$. Moreover, $(S(C^{LS}_{\max})\backslash S(r_k)) \cup (S(r_k)\backslash S(r_h)) = S(C^{LS}_{\max})\backslash S(r_h)$.

Observation 5. If there exists a $k \leqslant z$ with $S(C^{LS}_{\max})\backslash S(r_k) = \emptyset$, then the claim of Theorem 4.1 is true.

Reason. If $S(C^{SL}_{\max})\backslash S(r_k) = \emptyset$ for some $k \leqslant z$, then all time-slots between the time r_k and $C^{LS}_{\max}$ are part of $D(r_k)$, therefore $C^{LS}_{\max} = r_k + |D(r_k)|$. On the other hand, $C^{OPT}_{\max} > r_k$ and $C^{OPT}_{\max} \geqslant |D(r_k)|$. This implies $C^{OPT}_{\max} > \frac{r_k + |D(r_k)|}{2} = \frac{C^{LS}_{\max}}{2}$.

Therefore, we assume henceforth that $S(C^{LS}_{\max})\backslash S(r_k) \neq \emptyset$ for all $k \leqslant z$.

We can also exclude for every release date $r_k, k = 1, \ldots, z$, the case that the time-slot r_k is empty in *LS*; in this case all subsequent jobs would not be released before time r_k, and we could consider the remaining part of *LS* separately.

We consider the following two cases for $C^{LS}_{\max}$: Either there is a release date $r_h, h \in \{0, \ldots, z\}$, such that there is more load than $\frac{|S(C^{LS}_{\max})\backslash S(r_h)|m}{2}$ in $S(C^{LS}_{\max})\backslash S(r_h)$, or there is no such release date.

Case 1. There is no release date $r_h, h \in \{0, \ldots, z\}$, such that the load in $S(C^{LS}_{\max})\backslash S(r_h)$ is more than $\frac{|S(C^{LS}_{\max})\backslash S(r_h)|m}{2}$.

It follows that the load in $S(C^{LS}_{\max})\backslash S(r_z)$ cannot be more than $\frac{|S(C^{LS}_{\max})\backslash S(r_z)|m}{2}$. Together with Observation 5, this implies that there is a time-slot in $S(C^{LS}_{\max})\backslash S(r_z) \subseteq \{r_z + 1, \ldots, C^{LS}_{\max}\}$ containing a small job. Let e be a small job in $S(C^{LS}_{\max})\backslash S(r_z)$ that completes last. Let r_e be the release date, s_e the start time, and C_e the completion time of e. The load in $S(C^{LS}_{\max})\backslash S(r_e)$ is at most $\frac{|S(C^{LS}_{\max})\backslash S(r_e)|m}{2}$.

We partition the set $S(C^{LS}_{\max})\backslash S(r_e)$ into the disjoint sets $\bar{E}$, $E := E_1 \cup E_2$, and $\tilde{E}$. Let E be the set of time-slots in $S(C^{LS}_{\max})\backslash S(r_e)$ containing a slice of e. Let E_1 be the set of time-slots in E before the date r_z, and let E_2 be the set of time-slots in E after r_z. Notice that $E_2 \neq \emptyset$. Let $\bar{E}$ be the set of time-slots in $S(C^{LS}_{\max})\backslash S(r_e)$ containing no slice of e and which are before r_z. Let $\tilde{E}$ be the set of time-slots in $S(C^{LS}_{\max})\backslash S(r_e)$, which do not contain a slice of e and which come after r_z. Finally, D^e_{re} is the set of all time-slots between r_e and r_z that contain a slice of e, but which are not elements of the set $S(C^{LS}_{\max})\backslash S(r_e)$, and hence not part of the set E.

Observation 6. More than $\frac{m}{2}$ machines are busy in every time-slot in $\bar{E}$. Moreover, any two time-slots $s_1 \in E$ and $s_2 \in \bar{E}$ can be matched.

Reason. Job e with $m_e \leqslant \frac{m}{2}$ is released at date r_e, but is started at time s_e only. Therefore, at least $m - m_e + 1$ machines are busy at any time between r_e and s_e.

Observation 7. In every time-slot $s_1 \in \tilde{E}$ more than $\frac{m}{2}$ machines are busy. Any two time-slots $s_1 \in \tilde{E}$ and $s_2 \in E_2$ can be matched.

Reason. Since job e with $m_e \leqslant \frac{m}{2}$ is released at time r_e, but started only at time s_e, the statement is clearly true for all time-slots s_1 before s_e. Each time-slot $s_1 \in \tilde{E}$ after C_e contains a wide slice. We consider now a time-slot $s_2 \in E_2$. Either s_2 contains a slice of the wide job from s_1 or the wide job could not be processed in time-slot s_2 although it was already released, because too many machines are busy in s_2. In both cases the total number of busy machines in both time-slots exceeds m.

Let us consider the set $S(C^{LS}_{\max})\backslash S(r_z)$. We partition the time-slots between date r_z and $C^{LS}_{\max}$ that are part of the set $S(C^{LS}_{\max})\backslash S(r_e)$, but not part of the set $S(C^{LS}_{\max})\backslash S(r_z)$, and which are therefore part of the set $D(r_z)$, into two disjoint sets. The ones that contain a slice of e form the set $D^E_{r_z}$, and those without a slice of e form the set $D^{\tilde{E}}_{r_z}$. Thus, we have $D^E_{r_z} = E_2 \cap D(r_z)$ and $D^{\tilde{E}}_{r_z} = \tilde{E} \cap D(r_z)$. Each time-slot in $D^E_{r_z} \cup D^{\tilde{E}}_{r_z}$ contains a wide slice. We have therefore partitioned the set $S(C^{LS}_{\max})\backslash S(r_z)$ into the disjoint sets $E_2\backslash D^E_{r_z}$ and $\tilde{E}\backslash D^{\tilde{E}}_{r_z}$. All jobs in $S(C^{LS}_{\max})\backslash S(r_z)$ are released not later than r_z. Observation 7 implies the following insight.

Observation 8. $|E_2\backslash D^E_{r_z}| > |\tilde{E}\backslash D^{\tilde{E}}_{r_z}|$.

We distinguish four further cases with respect to the relationship between the size of the sets $\bar{E}$ and E_1 and the relationship between the size of the sets E_2 and $\tilde{E}$.

Case 1.A. $|\tilde{E}| < |E_2|$.

Case 1.A.1. $|\bar{E}| \geqslant |E| - |\tilde{E}|$.
Observation 7 implies that the set $\tilde{E}$ and any set of $|\tilde{E}|$ many time-slots in E_2 can be matched. Because of Observation 6, the set of the remaining $|E_2| - |\tilde{E}|$ many unmatched time-slots in E_2, together with the time-slots in E_1, can be matched with any set of $|E_2| - |\tilde{E}| + |E_1| = |E| - |\tilde{E}|$ many time-slots in $\bar{E}$. The number of busy machines in any remaining unmatched time-slot in $\bar{E}$ exceeds $\frac{m}{2}$. Thus, the set $S(C^{LS}_{\max})\backslash S(r_e)$ contains in total more load than $(|\tilde{E}| + |E| - |\tilde{E}|)m + \frac{(|\bar{E}|-|\tilde{E}|+\tilde{E})m}{2} = \frac{(|\bar{E}|+|E|+\tilde{E})m}{2} = \frac{|S(C^{LS}_{\max})\backslash S(r_e)|m}{2}$, which contradicts the assumption of Case 1.

Case 1.A.2. $|\bar{E}| < |E| - |\tilde{E}|$.
In this case, we have $|E| > \frac{|S(C^{LS}_{\max})\backslash S(r_e)|}{2}$. From $C^{OPT}_{\max} \geqslant r_e + p_e$ and Observation 3 follows $C^{OPT}_{\max} \geqslant r_e + |E| > r_e + \frac{|S(C^{LS}_{\max})\backslash S(r_e)|}{2} = r_e + \frac{C^{LS}_{\max} - r_e - |D(r_e)|}{2} = \frac{C^{LS}_{\max}}{2} + \frac{r_e - |D(r_e)|}{2} \geqslant \frac{C^{LS}_{\max}}{2}$, and Theorem 4.1 is proved in this case.

Case 1.B. $|\tilde{E}| \geqslant |E_2|$.

Case 1.B.1. $|\bar{E}| \geqslant |E_1|$.
Observation 7 shows that we can match any set of $|E_2|$ many time-slots in $\tilde{E}$ with the set E_2. Using Observation 6, we can match any set of $|E_1|$ many time-slots in $\bar{E}$ with the time-slots in E_1. In each of the remaining unmatched time-slots in $\tilde{E}$ and $\bar{E}$, more than $\frac{m}{2}$ machines are busy. Thus, the load in $S(C^{LS}_{\max})\backslash S(r_e)$ exceeds $(|E_1| + |E_2|)m + \frac{(|\bar{E}|-|E_1|)m}{2} + \frac{(|\tilde{E}|-|E_2|)m}{2} = \frac{(|\bar{E}|+|E|+|\tilde{E}|)m}{2} = \frac{|S(C^{LS}_{\max})\backslash S(r_e)|m}{2}$. This is in contradiction to the assumption of Case 1.

Case 1.B.2. $|\bar{E}| < |E_1|$.
We denote the set of the first $|\bar{E}|$ time-slots in E_1 by E^1_1. Observation 6 implies that E^1_1 can be matched with $\bar{E}$. From Observations 7 and 8 follows that any set of $|\tilde{E}\backslash D^{\tilde{E}}_{r_z}|$ many time-slots in $E_2\backslash D^E_{r_z}$ can be matched with the time-slots in $\tilde{E}\backslash D^{\tilde{E}}_{r_z}$. Furthermore, $|D^E_{r_z}| < |D^{\tilde{E}}_{r_z}|$ because of $|\tilde{E}| \geqslant |E_2|$ and Observation 8. Therefore, the set $D^{\tilde{E}}_{r_z}$ can be matched with any set of $|D^E_{r_z}|$ many time-slots in $D^{\tilde{E}}_{r_z}$. Since $|E_2| \leqslant |\tilde{E}|$, we obtain the following inequality: $|D^{\tilde{E}}_{r_z}| - |D^E_{r_z}| \geqslant |E_2\backslash D^E_{r_z}| - |\tilde{E}\backslash D^{\tilde{E}}_{r_z}|$. Hence, there are $|E_2\backslash D^E_{r_z}| - |\tilde{E}\backslash D^{\tilde{E}}_{r_z}|$ many time-slots in the remaining $|D^{\tilde{E}}_{r_z}| - |D^E_{r_z}|$ many time-slots in $D^{\tilde{E}}_{r_z}$ that can be matched with the set of the remaining $|E_2\backslash D^E_{r_z}| - |\tilde{E}\backslash D^{\tilde{E}}_{r_z}|$ many time-slots in $E_2\backslash D^E_{r_z}$. Let D_{rest} be the set of the $|D^{\tilde{E}}_{r_z}| - |D^E_{r_z}| - (|E_2\backslash D^E_{r_z}| - |\tilde{E}\backslash D^{\tilde{E}}_{r_z}|)$ many presently unmatched time-slots in $D^{\tilde{E}}_{r_z}$. By Observation 2, there is a set $D^{\text{match}}_{\text{rest}}$ of time-slots in $\{r_z - (|D^{\tilde{E}}_{r_z}| - |D^E_{r_z}| - (|E_2\backslash D^E_{r_z}| - |\tilde{E}\backslash D^{\tilde{E}}_{r_z}|)) + 1, \ldots, r_z\}$ that can be matched with D_{rest}. We combine the time-slots in $D^{\tilde{E}}_{r_z}$ that are matched with a time-slot in $D(r_e) \cap D^{\text{match}}_{\text{rest}}$, thus with a time-slot in $D^e_{r_e}$, which is therefore not part of the set $S(C^{LS}_{\max})\backslash S(r_e)$, in the set D_{spare}. The load in $S(C^{LS}_{\max})\backslash S(r_e)$ is at most $\frac{|S(C^{LS}_{\max})\backslash S(r_e)|m}{2}$. Since every time-slot in D_{spare} contains more load than $\frac{m}{2}$, it follows that the load in $(S(C^{LS}_{\max})\backslash S(r_e))\backslash D_{\text{spare}}$ is bounded from above by $\frac{|(S(C^{LS}_{\max})\backslash S(r_e))\backslash D_{\text{spare}}|m}{2}$.

If every time-slot in $E_1\backslash E^1_1$ has been matched with a time-slot in D_{rest}, then every time-slot in E has been matched with another time-slot in $(S(C^{LS}_{\max})\backslash S(r_e))\backslash D_{\text{spare}})\backslash E$. Each of the remaining time-slots in $(S(C^{LS}_{\max})\backslash S(r_e))\backslash D_{\text{spare}})\backslash E$ contains more than $\frac{m}{2}$ load. Thus, $(S(C^{LS}_{\max})\backslash S(r_e))\backslash D_{\text{spare}}$ contains more load than $\frac{|(S(C^{LS}_{\max})\backslash S(r_e))\backslash D_{\text{spare}}|m}{2}$, which is a contradiction.

We may therefore assume that there is at least one time-slot in $E_1 \setminus E_1^1$ that has not been matched with a time-slot in D_{rest}. Consequently, $|D_{\text{spare}}| \leqslant |D^e_{r_e}|$. Furthermore, $|E| > |\bar{E}| + |\tilde{E} \setminus D^{\tilde{E}}_{r_z}| + |D^E_{r_z}| + |E_2 \setminus D^E_{r_z}| - |\tilde{E} \setminus D^{\tilde{E}}_{r_z}| + (|D^{\tilde{E}}_{r_z}| - |D^E_{r_z}| - (|E_2 \setminus D^E_{r_z}| - |\tilde{E} \setminus D^{\tilde{E}}_{r_z}|)) - |D_{\text{spare}}| = |\bar{E}| + |\tilde{E}| - |D^{\tilde{E}}_{r_z}| + |D^E_{r_z}| + |E_2| - |D^E_{r_z}| - |\tilde{E}| + |D^{\tilde{E}}_{r_z}| + |D^{\tilde{E}}_{r_z}| - |D^E_{r_z}| - |E_2| + |D^E_{r_z}| + |\tilde{E}| - |D^{\tilde{E}}_{r_z}| - |D_{\text{spare}}| = |\bar{E}| + |\tilde{E}| - |D_{\text{spare}}|$. It follows that $|E| > \frac{|(S(C^{LS}_{\max}) \setminus S(r_e)) \setminus D_{\text{spare}}|}{2}$, and because $C^{OPT}_{\max} \geqslant r_e + pe$ we conclude that $C^{OPT}_{\max} \geqslant r_e + |E| + |D^e_{r_e}| > r_e + |D^e_{r_e}| + \frac{|(S(C^{LS}_{\max}) \setminus S(r_e)) \setminus D_{\text{spare}}|}{2} = r_e + |D^e_{r_e}| + \frac{C^{LS}_{\max}}{2} - \frac{r_e}{2} - \frac{|D(r_e)|}{2} - \frac{|D_{\text{spare}}|}{2} \geqslant \frac{C^{LS}_{\max}}{2} + \frac{r_e - |D(r_e)|}{2} + \frac{|D^e_{r_e}| - |D_{\text{spare}}|}{2} \geqslant \frac{C^{LS}_{\max}}{2}$, and herewith the correctness of Theorem 4.1 in this case.

A schematic illustration of the relevant part of the list-schedule *LS* for this case is given in Fig. 1.

Case 2. There is a release date $r_h < C^{LS}_{\max}$ such that $S(C^{LS}_{\max}) \setminus S(r_h)$ contains more load than $\frac{|S(C^{LS}_{\max}) \setminus S(r_h)|m}{2}$.

Let r_k be the smallest release date of this kind. If $r_k = 0$, the theorem follows immediately. Let us therefore assume that $r_k > 0$.

Observation 9. $S(r_k) \setminus S(r_h) \neq \emptyset$ for every release date $r_h < r_k$.

Reason. If $S(r_k) \setminus S(r_h) = \emptyset$ for $r_h < r_k$, Observation 4 implies $S(C^{LS}_{\max}) \setminus S(r_h) = (S(C^{LS}_{\max}) \setminus S(r_k)) \cup (S(r_k) \setminus S(r_h)) = S(C^{LS}_{\max}) \setminus S(r_k)$. Since $S(C^{LS}_{\max}) \setminus S(r_k)$ contains more load

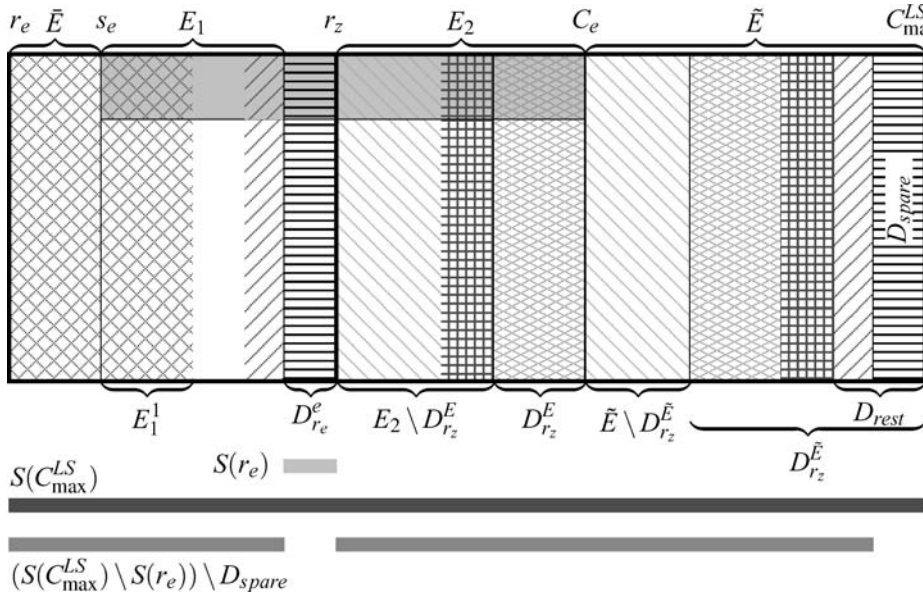

Fig. 1 Illustration of Case 1.B.2. The *bars* underneath the picture underline the time-slots that belong to the sets $S(r_e)$, $S(C^{LS}_{\max})$, or $(S(C^{LS}_{\max}) \setminus S(r_e)) \setminus D_{\text{spare}}$, respectively. The *horizontal bar* within *LS*, going from s_e to C_e, corresponds to the job e. Sets of time-slots with the same pattern have been matched with each other.

than $\frac{|S(C^{LS}_{\max})\backslash S(r_k)|m}{2}$, it follows that the set $S(C^{LS}_{\max})\backslash S(r_h)$ contains more load than $\frac{|S(C^{LS}_{\max})\backslash S(r_h)|m}{2}$, a contradiction to the definition of r_k.

Observation 10. The load of the set $S(r_k)\backslash S(r_h)$ is at most $\frac{|S(r_k)\backslash S(r_h)|m}{2}$, for every release date $r_h < r_k$.

Reason. If the load in $S(r_k)\backslash S(r_h)$ was more than $\frac{|S(r_k)\backslash S(r_h)|m}{2}$, we would be able to merge the loads in $S(r_k)\backslash S(r_h)$ and $S(C^{LS}_{\max})\backslash S(r_k)$, with the help of Observation 4. The cumulated load in $S(C^{LS}_{\max})\backslash S(r_h)$ would then exceed $\frac{S(C^{LS}_{\max})\backslash S(r_h)|m}{2}$, a contradiction to the definition of r_k.

Observation 11. Every wide slice in $S(C^{LS}_{\max})\backslash S(r_k)$ is processed in *OPT* after r_k.

Reason. The wide slices, which are processed in *OPT* before r_k, but in *LS* after r_k, are included in $D(r_k)$ and therefore not part of the set $S(C^{LS}_{\max})\backslash S(r_k)$.

If all small slices in $S(C^{LS}_{\max})\backslash S(r_k)$ are not released before r_k, Observation 11 and the assumption of Case 2 imply $C^{OPT}_{\max} > r_k + \frac{|S(C^{LS}_{\max})\backslash S(r_k)|}{2} \geqslant \frac{C^{LS}_{\max}}{2}$, and we are done.

We henceforth assume that there is at least one small job in $S(C^{LS}_{\max})\backslash S(r_k)$ which has been released before r_k. We call such small jobs, which are released before r_k but completed by the list-scheduling algorithm after r_k *out-jutting* jobs.

Let $\tilde{j}$ be the minimum of the number of slices after r_k of an out-jutting job j and the number of time-slots between r_j and r_k that contain no slice of j. Thus, $\tilde{j}$ is an upper bound of the number of slices of job j, which are processed in *LS* after r_k, but in *OPT* possibly before r_k. Let u be an out-jutting job for which this bound is maximal among all out-jutting jobs. Let $\tilde{D}^{\text{before}}_{r_u}$ be the set of time-slots in $D(r_u) \cap \{r_u + 1, \ldots, r_k\}$ that contain no slice of u. Let $D^{\text{before}}_{r_u}$ be the set of time-slots in $D(r_u) \cap \{r_u + 1, \ldots, r_k\}$ that contain a slice of u. Let $D^{\text{after}}_{r_u} := D(r_u) \cap \{r_k + 1, \ldots, C^{LS}_{\max}\}$. We partition the time-slots between date r_u and r_k into the disjoint sets U, $\bar{U}$, $\tilde{D}^{\text{before}}_{r_u}$, and $D^{\text{before}}_{r_u}$. Let U be the set of time-slots in $\{r_u + 1, \ldots, r_k\}\backslash D^{\text{before}}_{r_u}$ that contain a slice of the job u. Let $\bar{U}$ be the set of time-slots in $\{r_u + 1, \ldots, r_k\}\backslash \tilde{D}^{\text{before}}_{r_u}$ that do not contain a slice of the job u. Hence, the set $\{r_u + 1, \ldots r_k\} \cup (D(r_k)\backslash D^{\text{after}}_{r_u})$ is composed of the disjoint sets of time-slots U, $\bar{U}$, $\tilde{D}^{\text{before}}_{r_u}$, $D^{\text{before}}_{r_u}$, and $D(r_k)\backslash D^{\text{after}}_{r_u}$.

Observation 12. The term $|\bar{U}| + |\tilde{D}^{\text{before}}_{r_u}|$ is an upper bound for $\tilde{u}$.

Observation 13. We have $\tilde{u} + |D(r_k)| < r_k$.

Reason. Because of Observation 10, the set $S(r_k)\backslash S(r_u)$ contains at most $\frac{|S(r_k)\backslash S(r_u)|m}{2}$ load. Together, more than m machines are busy in each pair of time-slots $s_1 \in \bar{U}$ and $s_2 \in U$ since there is no slice of job u in s_1 although job u was already released. Moreover, more than $m - m_u \geqslant \frac{m}{2}$ machines are busy in each time-slot in $\bar{U}$ and $\tilde{D}^{\text{before}}_{r_u}$. Since each time-slot in $D(r_k)\backslash D^{\text{after}}_{r_u}$ contains a wide slice, more than $\frac{m}{2}$ machines are busy in each of them. Because of Observation 2, for every time-slot $s_1 \in D(r_k)\backslash D^{\text{after}}_{r_u}$ there exists another time-slot $s_2 \in \{r_k - |D(r_k)\backslash D^{\text{after}}_{r_u}| + 1, \ldots, r_k\}$ such that s_1 and s_2 can be matched. This way at most $|D^{\text{before}}_{r_u}|$ many time-slots from $D(r_k)\backslash D^{\text{after}}_{r_u}$ are matched with a time-slot in $D^{\text{before}}_{r_u}$. We combine those time-slots to the set D_{spare}. We have $|D_{\text{spare}}| \leqslant |D^{\text{before}}_{r_u}|$. Since $S(r_k)\backslash S(r_u)$ contains at most $\frac{|S(r_k)\backslash S(r_u)|m}{2}$ load, it follows that $(S(r_k)\backslash S(r_u))\backslash D_{\text{spare}}$ contains at most $\frac{|(S(r_k)\backslash S(r_u))\backslash D_{\text{spare}}|m}{2}$ load, because every time-slot in D_{spare} is loaded by more than $\frac{m}{2}$. The set $(S(r_k)\backslash S(r_u))\backslash D_{\text{spare}}$ decomposes

into the disjoint sets $\bar{U}$, U and $(D(r_k)\backslash D^{\text{after}}_{r_u})\backslash D_{\text{spare}}$. If $|U| \leqslant |\bar{U}| + |(D(r_k)\backslash D^{\text{after}}_{r_u})\backslash D_{\text{spare}}|$, we could match every time-slot in U with another time-slot in $((S(r_k)\backslash S(r_u))\backslash D_{\text{spare}})\backslash U$. The remaining time-slots in $((S(r_k)\backslash S(r_u))\backslash D_{\text{spare}})\backslash U$ each contain more than $\frac{m}{2}$ load. Thus, the load in $(S(r_k)\backslash S(r_u))\backslash D_{\text{spare}}$ would exceed $\frac{|(S(r_k)\backslash S(r_u))\backslash D_{\text{spare}}|m}{2}$, in contradiction to our earlier observations regarding this set. We conclude that $|U| > |\bar{U}| + |D(r_k)\backslash D^{\text{after}}_{r_u}| - |D^{\text{before}}_{r_u}|$. Combining this inequality with Observations 12 and 3 results in $\tilde{u} + |D(r_k)| < |U| + |\tilde{D}^{\text{before}}_{r_u}| + |D^{\text{before}}_{r_u}| + |D^{\text{after}}_{r_u}| \leqslant |U| + |D(r_u)| \leqslant |U| + r_u \leqslant r_k$.

Observation 14. The sum of the widths of all out-jutting jobs is at most $\frac{m}{2}$.

Reason. If this would not be the case, the sum of the widths of all small jobs between r_{k-1} and r_k would exceed $\frac{m}{2}$. But then there would be no time-slot between the release dates r_{k-1} and r_k that is filled with at most $\frac{m}{2}$ load. Using Observation 9, this is in contradiction to Observation 10 since it would follow that the load in $S(r_k)\backslash S(r_{k-1})$ exceeds $\frac{|S(r_k)\backslash S(r_{k-1})|m}{2}$.

Let us consider the set $S(C^{LS}_{\max})\backslash S(r_k)$. The load between r_k and $C^{LS}_{\max}$, excluding the load in the time-slots in $D(r_k)$, is more than $\frac{(C^{LS}_{\max} - r_k - |D(r_k)|)m}{2}$. It follows from Observations 11 and 14 that at most an amount of $\frac{\tilde{u}m}{2}$ of this load can be processed in *OPT* before r_k. Therefore, the load in *LS* that has to be processed in OPT after r_k exceeds $\frac{(C^{LS}_{\max} - r_k - |D(r_k)|)m}{2} - \frac{\tilde{u}m}{2}$. Consequently, by using the inequality from Observation 13, we obtain $C^{OPT}_{\max} > r_k + \frac{C^{LS}_{\max} - r_k - |D(r_k)|}{2} - \frac{\tilde{u}}{2} > \frac{C^{LS}_{\max}}{2}$.

A schematic illustration of the relevant part of the list-schedule *LS* for this case is given in Fig. 2.

Therefore, the length of *LS* is less than two times the length of the optimal preemptive schedule *OPT*. □

An immediate implication on the power of preemption is stated in the following corollary.

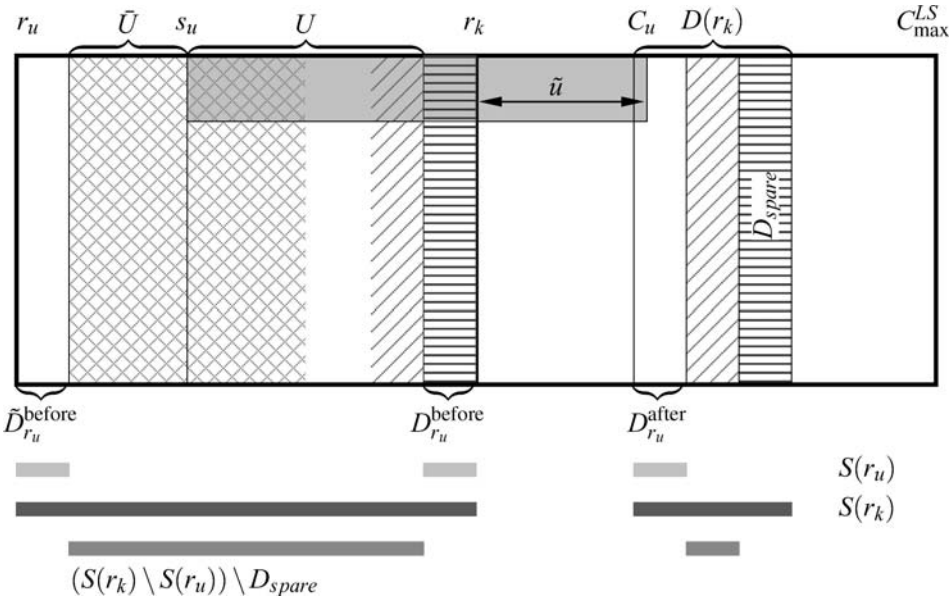

Fig. 2 Illustration of Case 2. The bars *underneath* the picture underline the time-slots that belong to the sets $S(r_u)$, $S(r_k)$, or $(S(r_k)\backslash S(r_u))\backslash D_{\text{spare}}$, respectively. Sets of time-slots with the same pattern have been matched with each other.

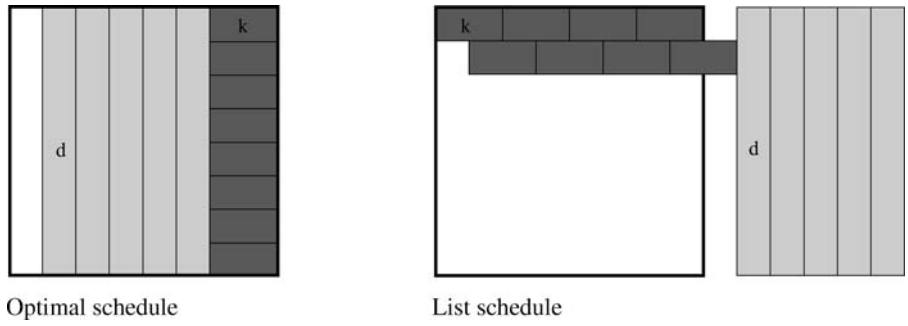

Fig. 3 Worst-case example for any list-scheduling with release dates

Corollary 4.2. *An optimal nonpreemptive schedule is less than twice as long as an optimal preemptive schedule for all instances of* $P|m_j, r_j, pmtn|C_{\max}$. *This bound is tight.*

Chen and Vestjens (1997) proved that the p_j-list-scheduling algorithm (also called the LPT rule) is a 3/2-approximation algorithm for $P|r_j|C_{\max}$. In contrast, the following example shows that for nonpreemptive scheduling with parallel jobs and release dates, no list-scheduling algorithm has a performance guarantee smaller than 2, no matter which priority list is chosen.

Example 4.3. Let m be the number of machines. Let $m-3$ jobs $d = 1, \ldots, m-3$ of length $p_d = 1$ and width $m_d = m$ be given. We call these jobs big jobs. The release date of each big job $d = 1, \ldots, m-3$ is $r_d := d$. Let there also be m small jobs $k = 1, \ldots, m$ with length 2 and width 1. The release date of a small job is $r_k := k-1, k = 1, \ldots, m$. An optimal schedule has length m, by scheduling the big jobs from time 1 to time $m-2$ and using the last two time-slots for the small jobs. The first time-slot remains empty. Each list-scheduling algorithm receives at time 0 only one job, namely the first small job to be scheduled. This job therefore starts at time 0. At time 1 the list-scheduling algorithm receives the second small job and the first big job. Since there are not enough idle machines to schedule the big job, the list-scheduling algorithm assigns the second small job to start at time 1. This pattern reoccurs at each point in time thereafter; one small job is still running, thus preventing the start of big jobs and causing the next small job to be started. At time $m+1$ all small jobs are completed and the big jobs can be started. The resulting schedule has length $m+1+m-3 = 2m-2$. Therefore, the ratio between the makespan of any list-scheduling algorithm and the optimal makespan is $2-2/m$.

Figure 3 illustrates Example 4.3 with eight machines.
Note that the variant of list-scheduling in which jobs are assigned to machines in order of their priorities (sometimes called serial or job-based list-scheduling) does not lead to improved performance guarantees either, even if jobs are in order of nonincreasing widths (Example 3.2) or nonincreasing processing times (Example 4.3).

5. Online-scheduling

In this section, we discuss online-scheduling of parallel jobs. In an online environment, parts of the input data are not known in advance. In scheduling, one typically distinguishes between three basic online models, each characterized by a different dynamics of the situation (see, e.g., Sgall, 1998). In the first online model, *scheduling jobs one by one*, jobs arrive one after the

other, and the current job needs to be (irrevocably) scheduled before the next job and all its characteristics become known. In the model *jobs arriving over time*, the characteristics of a job become known when the job becomes known, which happens at its release date. In contrast, in the model *unknown running times*, the processing time of a job remains unknown until it is completed.

Since list-scheduling (without special ordering of the list) complies with the requirements of the online model *unknown running times*, it follows from the work of Garey and Graham (1975) that list-scheduling is 2-competitive in the context of the online model *unknown running times* for parallel jobs without release dates. For the same model with release dates and for the online model *jobs arriving over time*, Theorem 4.1 or Theorem 4 in Naroska and Schwiegelshohn (2002) implies that list-scheduling is 2-competitive for both the preemptive and the nonpreemptive version. Shmoys, Wein, and Williamson (1995) showed for both versions of the online model *unknown running times* that there is no deterministic online algorithm with a better competitive ratio than $2 - 1/m$, even if all jobs are nonparallel. Therefore, list-scheduling algorithms achieve the best possible competitive ratio within the class of deterministic online algorithms for this model.

Chen and Vestjens (1997) showed that 1.347 is a lower bound for any deterministic nonpreemptive online algorithm for the model *jobs arriving over time*, even if every job needs only one machine for its processing. This bound does not hold if preemptions are allowed. In fact, Gonzalez and Johnson (1980) presented for the preemptive case of the online model *jobs arriving over time* with nonparallel jobs an exact algorithm. In comparison, Example 5.1 shows that there is no deterministic online algorithm for the preemptive version of online model *jobs arriving over time* with a competitive ratio smaller than 6/5, when dealing with parallel jobs.

Example 5.1. There are four identical parallel machines. At time 0 we release three jobs, job a and job b, each with length 1 and width 2, and job ℓ with length 2 and width 1. If the online algorithm decides to dedicate at most 3/5 of the first time-slot to the processing of job ℓ, we do not release any more jobs. In this case, the makespan of the schedule of the online algorithm is at least $3 - 3/5$, whereas the length of the optimal schedule is 2. If the online algorithm dedicates more than 3/5 of the first time-slot to job ℓ, we release at time 1 a job d with length 2 and width 3. In this case the online algorithm produces a schedule of length at least $3 + 3/5$, whereas the optimal makespan is 3. Both cases give us a ratio between the length of the online schedule and the optimal makespan of 6/5.

For the online model *scheduling jobs one by one*, list-scheduling achieves a competitive ratio of $2 - 1/m$ for nonparallel jobs (Graham, 1966). In contrast, list-scheduling of parallel jobs does not have a constant competitive ratio, which is revealed by Example 5.2 below. In fact, for parallel jobs and the online model *scheduling jobs one by one* no algorithm with constant competitive ratio was known heretofore, to the best of the author's knowledge. We present the first such algorithm below (Algorithm 5.4) and show that it is 12-competitive (Theorem 5.5). Theorem 5.3 provides a lower bound of 2.25 for the competitive ratio of any deterministic online algorithm for the model *scheduling jobs one by one* with parallel jobs. The latter result shows again that scheduling parallel jobs is significantly harder than scheduling nonparallel jobs. For nonparallel job scheduling, deterministic online algorithms are known with competitive ratio smaller than 2 (Albers, 1999; Fleischer and Wahl, 2000).

Example 5.2. Let I be an instance of the online-scheduling model scheduling jobs one by one with m machines and $2m$ jobs. Half of the jobs in I are wide; they have length 1 and width m. The other jobs $j = 1, \ldots, m$ are small with width 1 and length $p_j = j$. The list alternates between

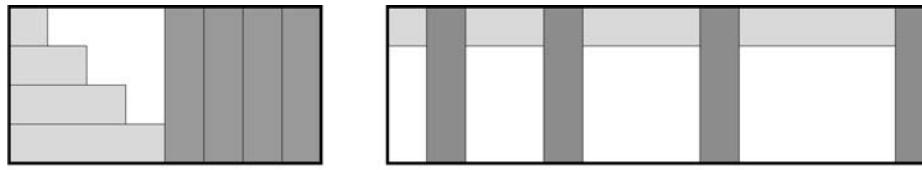

Fig. 4 Worst-case example for list-scheduling in the online model *scheduling jobs one by one*

small and wide jobs, and shorter small jobs precede longer small jobs. The optimal schedule has length $2m$, whereas the list-schedule has a makepan of $m + \sum_{j=1}^{m} j = \frac{m^2+3m}{2}$. The two schedules of an instance with four machines are depicted in Fig. 4.

Theorem 5.3. *No deterministic online algorithm for the online model scheduling jobs one by one of the scheduling problem $P|m_j|C_{\max}$ has a competitive ratio smaller than or equal to 2.25.*

Proof: Let A be any deterministic online algorithm for the model *scheduling jobs one by one* of $P|m_j|C_{\max}$ with competitive ratio 2.25. We construct an instance with a list in which jobs of width 1 and jobs of width m alternate. The length of each job is set in a way such that it can start only after the completion time of the previous job in the list. Let the number of machines be $m := 10$. The total number of jobs is at most 20.

More specifically, let k_i be the ith job of width 1 in the list, and let d_i be the ith job of width m, $i = 1, \ldots, m$. Let z_{k_i} be the delay introduced by A prior to starting job k_i. Similarly, let z_{d_i} be the delay before d_i. Then, the job lengths are defined as follows: $p_{k_1} := 1$, $p_{k_{i+1}} := z_{k_i} + p_{k_i} + z_{d_i} + 1$, and $p_{d_i} := \max_{j<i}\{z_{k_j}, z_{d_j}, z_{k_i}\} + 1$.

The length of the schedule produced by Algorithm A after the completion of job k_i is therefore $z_{k_i} + p_{k_i} + \sum_{j=1}^{i-1}(z_{k_j} + p_{k_j} + z_{d_j} + p_{d_j})$, and it is $\sum_{j=1}^{i}(z_{k_j} + p_{k_j} + z_{d_j} + p_{d_j})$ after the completion of job d_i.

The length of an optimal schedule including all jobs from the list up to (and including) job k_i is at most $p_{k_j} + \sum_{j=1}^{i-1} p_{d_j}$. If the instance ends with the ith d-job, the length of an optimal schedule is at most $p_{k_i} + \sum_{j=1}^{i} p_{d_j}$.

We prove the lower bound of 2.25 for the competitive ratio by complete enumeration over the possible delays z_{k_i} and z_{d_i}, $i = 1, \ldots, 10$, introduced by Algorithm A. Note that no delay can be too large because the competitive ratio 2.25 must be satisfied at any time during the run of the Algorithm (i.e., for every sub-instance). Using a computer program, it turns out that there is no way for A to create delays in such a manner that its competitive ratio is 2.25 for all sub-instances.

□

The following algorithm is the first algorithm with a constant competitive ratio for scheduling parallel jobs one by one.

Algorithm 5.4.

1. *Partition the time axis into the intervals $I_i := [2^i, 2^{i+1}]$, $i = 0, 1, \ldots$.*
2. *Schedule the arriving job j of length p_j and width m_j in the earliest interval I_i that is more than twice as long as p_j and in which job j can feasibly be scheduled, as follows:*

2.1. If $m_j > m/2$, schedule job j as late as possible within I_i.
2.2. If $m_j \leqslant m/2$, schedule job j as early as possible within I_i.

Theorem 5.5. *The competitive ratio of Algorithm 5.4 for the online version scheduling jobs one by one of the scheduling problem $P|m_j|C_{\max}$ is smaller than 12.*

Proof: Let S be the schedule constructed by Algorithm 5.4. We denote by $C^S_{\max}$ the length of S and by $C^*_{\max}$ the optimal offline makespan. Let $I_\ell = [2^\ell, 2^{\ell+1}]$ be the last and therefore the longest nonempty interval of S. Hence, $C^S_{\max} \leq 2^{\ell+1} - 1$.

We distinguish two cases depending on whether I_ℓ contains a job j with $p_j \geqslant 2^\ell/4$ or not, in which case I_ℓ contains only jobs that are shorter, but did not fit into the preceding interval.

Case 1. There is $j \in I_\ell$ with $p_j \geqslant 2^{\ell-2}$.
An optimal schedule cannot be shorter than the longest job of the instance. Thus, the optimal makespan is at least $C^*_{\max} \geqslant 2^{\ell-2}$. Consequently $C^S_{\max}/C^*_{\max} \leqslant (2^{\ell+1} - 1)/2^{\ell-2} < 8$.

Case 2. For all jobs $j \in I_\ell$ we have $p_j < 2^{\ell-2}$.
In this case, every job $j \in I_\ell$ did not fit anymore into the interval $I_{\ell-1}$. We consider the interval $I_{\ell-1}$. Its length is $2^{\ell-1}$. We partition the set of time-slots of the interval $I_{\ell-1}$ into the disjoint sets K, H, L, and D. Let K be the set of time-slots that are filled to more than $m/2$ with small jobs, i.e., jobs with $m_j \leqslant m/2$. These time-slots had been filled during Step 2.2. of Algorithm 5.4 and are located at the beginning of the interval. Let H be the set of time-slots that contain only small jobs, but in which at most $m/2$ machines are busy. These time-slots are located right after the time-slots in K. All jobs in H start no later than in the first time-slot of H. Let L be the set of empty time-slots. They are located between the time-slots of H and the time-slots belonging to D. Let D be the set of time-slots that contain a wide job, i.e., a job with $m_j > m/2$. These time-slots were filled during Step 2.1. of Algorithm 5.4 and are the last time-slots of the interval $I_{\ell-1}$.

The sets K, H, L, and D are disjoint and we have $|K| + |H| + |L| + |D| = 2^{\ell-1}$. The time-slots in K and D are filled to more than half. Note that there cannot be more time-slots in H than in the rest of the interval since all jobs in H start no later than in the first time-slot of H and all jobs are no longer than half of the length of the interval they belong to.

We consider a job j that could have been processed within the interval $I_{\ell-1}$ (since $I_{\ell-1}$ is more than twice as long as job j), but was forced into the next interval I_ℓ because there was insufficient space. We distinguish two further cases depending on whether job j is wide or small.

Case 2.1. $m_j > m/2$.
Obviously, $p_j > |L|$ since otherwise j would have fitted into the empty time-slots in $I_{\ell-1}$. The total load of job j and of jobs scheduled in the interval $I_{\ell-1}$ is more than $(|K| + |D|)\frac{m}{2} + |L|\frac{m}{2} = \frac{m}{2}(|K| + |D| + |L|) = \frac{m}{2}(2^{\ell-1} - |H|)$. It follows that $C^*_{\max} \geqslant (2^{\ell-1} - |H|)/2$. We also have $C^*_{\max} \geqslant |H|$. These facts combined lead to $C^*_{\max} \geqslant 2^{\ell-1}/3$. It follows that $C^S_{\max}/C^*_{\max} \leqslant \frac{3(2^{\ell+1}-1)}{2^{\ell-1}} < 12$.

Case 2.2. $m_j \leqslant m/2$.
This time $p_j > |H| + |L|$ since otherwise j could have been assigned to time-slots in H and L. Thus, $C^*_{\max} \geqslant p_j > |H| + |L|$. Moreover, the time-slots in K and D together contain more load than $(|K| + |D|)\frac{m}{2} = (2^{\ell-1} - (|H| + |L|))\frac{m}{2}$, leading us to $C^*_{\max} > (2^{\ell-1} - (|H| + |L|))/2$. We combine both lower bounds for the optimum to obtain $C^*_{\max} \geqslant 2^{\ell-1}/3$. This results in $C^S_{\max}/C^*_{\max} \leqslant \frac{3(2^{\ell+1}-1)}{2^{\ell-1}} < 12$. □

Note that 8 is an asymptotic lower bound on the competitive ratio of Algorithm 5.4, as the first case in the proof of Theorem 5.5 shows.

6. Conclusion

List-scheduling is the most expedient approach for solving scheduling problems on identical parallel machines to minimize the makespan. List-scheduling algorithms are easy to understand, simple to implement, fast, and in most cases they achieve a good performance guarantee; they even work in many online environments. In this paper, we extended the study of this class of greedy-like algorithms, which was started for parallel job scheduling by Turek, Wolf, and Yu (1992). We provided several new upper and lower bounds for the performance guarantee of list-scheduling algorithms for scheduling parallel jobs to minimize the makespan, offline and online. We showed that list-scheduling with an arbitrary priority list achieves a performance guarantee of 2. We also showed that no list-scheduling algorithm can achieve a better performance guarantee than 2 for the nonpreemptive problem with parallel jobs and release dates, no matter which priority list is chosen. This result contrasts with traditional scheduling of nonparallel jobs, where some list-scheduling algorithms have a performance guarantee of 3/2. If no release dates are present, p_j-list-scheduling of nonparallel jobs has a performance guarantee of 4/3, whereas we showed that 3/2 is a lower bound for parallel job scheduling, unless P = NP. More examples of this kind can be taken from the following Tables 1 and 2, which give an overview of the best-known offline and online results for scheduling nonparallel jobs (Table 1) and parallel jobs (Table 2) to minimize

Table 1 Summary of results for scheduling nonparallel jobs to minimize the makespan on identical parallel machines

Nonparallel jobs			
Model	Release dates	Preemptive	Nonpreemptive
Offline	Without r_j	P, McNaughton (1959)	PTAS, Hochbaum and Shmoys (1987) Strongly NP-hard, Garey and Johnson (1979)
	With r_j	P, Horn (1974)	PTAS, Hall and Shmoys (1989) Strongly NP-hard, Garey and Johnson (1979)
Online model		–	1.923, Albers (1999)
Scheduling jobs one by one		–	1.852, Albers (1999)
Online model	Without r_j	$2-1/m$, Graham (1966)	$2-1/m$, Graham (1966)
Unknown running times		$2-1/m$, Shmoys, Wein, and Williamson (1995)	$2-1/m$, Shmoys, Wein, and Williamson (1995)
	With r_j	$2-1/m$, Gusfield (1984); Hall and Shmoys (1989)	$2-1/m$, Gusfield (1984); Hall and Shmoys (1989)
		$2-1/m$, Shmoys, Wein, and Williamson (1995)	$2-1/m$, Shmoys, Wein, and Williamson (1995)
Online model	With r_j		3/2, Chen and Vestjens (1997)
Jobs arriving over time		*P*, Gonzalez and Johnson (1980)	1,347, Chen and Vestjens (1997)

Table 2 Summary of results for scheduling parallel jobs to minimize the makespan on identical parallel machines

		Parallel jobs	
Model	Release dates	Preemptive	Nonpreemptive
Offline	Without r_j	2, Feldmann, Sgall, and Teng (1994) NP-hard Drozdowski (1994)	2, Feldmann, Sgall, and Teng (1994) 3/2*
	With r_j	2*, see also Naroska and Schwiegelshohn (2002) NP-hard Drozdowski (1994)	2*, see also Naroska and Schwiegelshohn (2002) 3/2*
Online model		–	12*
Scheduling jobs one by one		–	2.25*
Online model *Unknown running times*	Without r_j	2, Feldmann, Sgall, and Teng (1994) $2 - 1/m$, Shmoys, Wein, and Williamson (1995)	2, Feldmann, Sgall, and Teng (1994) $2 - 1/m$, Shmoys, Wein, and Williamson (1995)
	With r_j	2*, see also Naroska and Schwiegelshohn (2002) $2 - 1/m$, Shmoys, Wein, and Williamson (1995)	2*, see also Naroska and Schwiegelshohn (2002) $2 - 1/m$, Shmoys, Wein, and Williamson (1995)
Online model *Jobs arriving over time*	With r_j	2*, see also Naroska and Schwiegelshohn (2002) 6/5*	2*, see also Naroska and Schwiegelshohn (2002) 1.347, Chen and Vestjens (1997)

the makespan. The first row contains for each model the best-known performance guarantee or competitive ratio, respectively. The second row contains for each model the complexity or the best-known lower bound for the performance guarantee. The lower bounds for the performance guarantee of approximation algorithms for offline problems assume that P $\neq$ NP. If a problem is solvable in polynomial time, we put a "P" in the corresponding field. We use "PTAS" to indicate that a polynomial-time approximation scheme exists for an offline problem. References are attached to each result; results that are marked with an asterisk are proved in this paper.

As we see, for nonparallel as well as parallel jobs, the lower and upper bounds for the competitive ratio coincide for the online model *unknown running times*. The instance for the lower bound, provided by Shmoys, Wein, and Williamson (1995), consists of nonparallel jobs only. In contrast, for the online model *jobs arriving over time* we were able to make use of the size of jobs to provide the first lower bound for the competitive ratio of any deterministic preemptive online scheduling algorithm to minimize the makespan. It remains to improve the lower bound for the nonpreemptive case. The most interesting problem, however, is to find an algorithm that reduces the gap between the known performance guarantee 2 and the lower bound of 3/2. Because of Example 4.3, we know that such an algorithm cannot be a list-scheduling procedure. Another open problem is to decrease the gap for the online model *scheduling jobs one by one*, where we proved a first lower bound and a first online algorithm with constant competitive ratio, which appear to leave room for improvement.

Acknowledgments I am grateful to Martin Skutella for directing me to this research area. I also thank him and Rolf Möhring for various helpful comments. In particular, they were the advisors of my Diplomarbeit (Johannes, 2001) where the results presented here were first derived.

References

Albers, S. "Better bounds for online scheduling," *SIAM Journal on Computing*, **29**, 459–473 (1999).

Amoura, A. K., E. Bampis, C. Kenyon, and Y. Manoussakis, "Scheduling independent multiprocessor tasks," in R. Burkard and G. Woeginger (eds.), *Algorithms–ESA'97, Lecture Notes in Computer Science 1284*, Springer, Berlin, 1997, pp. 1–12 .

Baker, B. S., E. G. Coffman, Jr., and R. L. Rivest, " Orthogonal packings in two dimensions," *SIAM Journal on Computing*, **9**, 846–855 (1980).

Bażewicz, J., M. Drabowski, and J. Węglarz, "Scheduling multiprocessor tasks to minimize schedule length," *IEEE Transactions on Computers*, **35**, 389–393 (1986).

Chen, B. and A. P. A. Vestjens, "Scheduling on identical machines: How good is LPT in an on-line setting?,"*Operations Research Letters*, **21**, 165–169 (1997).

Chen, J. and A. Miranda, "A polynomial time approximation scheme for general multiprocessor job scheduling," in *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, 1999, pp. 418–427 .

Coffman, E. G., Jr. M. R. Garey., D. S. Johnson, and R. E. Tarjan, "Performance bounds for level-oriented two-dimensional packing algorithms," *SIAM Journal on Computing*, **9**, 808–826 (1980).

Drozdowski, M., "On complexity of multiprocessor task scheduling," *Bulletin of the Polish Academy of Sciences, Technical Sciences*, **43**, 437–445 (1994).

Drozdowski, M., "Scheduling multiprocessor tasks–an overview," *European Journal of Operational Research*, **64**, 215–230 (1996).

Du, J. and J. Y.-T. Leung, "Complexity of scheduling parallel task systems," *SIAM Journal of Discrete Mathematics*, **2**, 473–487 (1989).

Feitelson, D. G., "A survey of scheduling in multiprogrammed parallel systems," *Research Report RC 19790 (87657),* IBM T. J. Watson Research Center, Oct. 1994, revised Aug. 1997.

Feitelson, D. G., L. Rudolph, U. Schwiegelshohn, K. C. Sevcik, and P. Wong, "Theory and practice in parallel job scheduling," in D. G. Feitelson and L. Rudolph (eds.), *Job Scheduling Strategies for Parallel Processing*, Lecture Notes in Computer Science 1291, Springer, Berlin, 1997, pp. 1–34,.

Feldmann, A., J. Sgall, and S.-H. Teng," Dynamic scheduling on parallel machines," *Theoretical Computer Science*, **130**, 49–72 (1994).

Fleischer, R. and M. Wahl, "On-line scheduling revisited," *Journal of Scheduling*, **3**, 343–353 (2000).

Garey, M. R. and R. L. Graham, "Bounds for multiprocessor scheduling with resource constraints," *SIAM Journal on Computing*, **4**, 187–200 (1975).

Garey, M. R. and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, CA. 1979.

Golan, I., "Performance bounds for orthogonal oriented two-dimensional packing algorithms," *SIAM Journal on Computing*, **10**, 571–582 (1981).

Gonzalez, T. F. and D. B. Johnson, "A new algorithm for preemptive scheduling of trees," *Journal of the Association for Computing Machinery*, **27**, 287–312 (1980).

Graham, R. L., "Bounds for certain multiprocessing anomalies," *Bell System Technical Journal*, **45**, 1563–1581 (1966).

Graham, R. L., "Bounds on multiprocessing timing anomalies," *SIAM Journal of Applied Mathematics*, **17**, 416–426 (1969).

Graham, R. L., E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan, "Optimization and approximation in deterministic sequencing and scheduling: A survey," *Annals of Discrete Mathematics*, **5**, 287–326 (1979).

Gusfield, D., "Bounds for naive multiple machine scheduling with release times and deadlines," *Journal of Algorithms*, **5**, 1–6 (1984).

Hall, L. A. and D. B. Shmoys, "Approximation schemes for constrained scheduling problems," in *Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science*, 1989 pp. 134–139.

Hochbaum, D. S., and Shmoys, D. B., "Using dual approximation algorithms for scheduling problems: Theoretical and practical results," *Journal of the Association for Computing Machinery*, **34** , 144–162 (1987).

Horn, W. A., "Some simple machine scheduling algorithms," *Naval Research Logistics Quarterly*, **21**, 177–185 (1974).

Jansen, K. and L. Porkolab, "Linear-time approximation schemes for scheduling malleable parallel tasks," in *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1999, pp. 490–498.

Jansen, K. and L. Porkolab, "Preemptive parallel task scheduling in $O(n) + poly(m)$ time," in D. T. Lee and S.-H. Teng (eds.), *Algorithms and Computation*, Lecture Notes in Computer Science 1969, Springer, Berlin, 2000, pp. 398–409.

Johannes, B., Obere und untere Schranken für die Güte von Heuristiken und Relaxierungen im Maschinen Scheduling, Diplomarbeit, Institut für Mathematik, Technische Universität Berlin, Germany, June 2001; partly published in B. Johannes, Scheduling parallel jobs to minimize makespan, *Technical Report* 723/2001, Institut für Mathematik, Technische Universität Berlin, Germany.

Kenyon, C. and E. Remila, "Approximative strip packing," in *Proceedings of the 37th IEEE Symposium on Foundations of Computer Science*, 1996, pp. 31–36.

Li, K., "Analysis of an approximation algorithm for scheduling independent parallel tasks," *Discrete Mathematics and Theoretical Computer Science*, **3**, 1999, 155–166.

Ludwig, W., and P. Tiwari, "Scheduling malleable and nonmalleable tasks," in *Proceedings of the 5th ACM-SIAM Symposium on Discrete Algorithms*, 1994, pp. 167–176.

McNaughton, R., "Scheduling with deadlines and loss functions," *Management Science*, **6**, 1–12 (1959).

Mu'alem, A. W. and D. G. Feitelson, "Preemptive bicriteria scheduling for parallel jobs: Off-line and on-line algorithms. Technical Report 99-36," Leibniz Center for Research in Computer Science, Hebrew University, Jerusalem, Israel, Dec. 1999.

Naroska, E. and U. Schwiegelshohn, "On an online scheduling problem for parallel jobs," I*nformation Processing Letters*, **81**, 297–304 (2002).

Sgall, J., "On-line scheduling," in A. Fiat and G. J. Woeginger (eds.), *Online Algorithms, Lecture Notes in Computer Science 1442*, Springer, Berlin, 1998, pp. 197–231.

Shmoys, D. B., J. Wein, and D. P. Williamson, "Scheduling parallel machines on-line," *SIAM Journal on Computing*, **24**, 1313–1331 (1995).

Sleator, D., "A 2.5 times optimal algorithm for packing in two dimensions," *Information Processing Letters*, **10**, 37–40 (1980).

Steinberg, A., "A strip-packing algorithm with absolute performance bound 2,"*SIAM Journal on Computing*, **26**, 401–409 (1997).

Turek, J., J. L. Wolf, and P. S. Yu, "Approximate algorithms for scheduling parallelizable tasks," in *Proceedings of the 4*th *ACM Symposium on Parallel Algorithms and Architectures*, 1992, pp. 323–332.