Technische Universität Berlin

FAKULTÄT II
MATHEMATIK UND
NATURWISSENSCHAFTEN
Institut für Mathematik

SCHEDULING PARALLEL JOBS TO MINIMIZE MAKESPAN

by

BERIT JOHANNES

No. 723/2001

Scheduling Parallel Jobs to Minimize Makespan

Berit Johannes[*]

November 11, 2001

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 pre-specified, job-dependent number of machines when being processed. Our main result is the following. The makespan of a (non-preemptive) schedule constructed by *any* listscheduling algorithm is within a factor of 2 of the optimal preemptive makespan. This gives the best known approximation algorithms for both the preemptive and the non-preemptive variant of the problem, improving upon previously known performance guarantees of 3. We also show that no listscheduling algorithm can achieve a better performance guarantee than 2 for the non-preemptive problem, no matter which priority list is chosen.

Since listscheduling also works in the online setting in which jobs arrive over time and the length of a job becomes only known when it completes, the main result 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. In this context, no listscheduling algorithm has a constant competitive ratio. We present the first online algorithm for scheduling parallel jobs with a constant competitive ratio. We also prove a new information-theoretic lower bound of 2.25 for the competitive ratio of any deterministic online algorithm for this model.

1 Introduction

Scheduling parallel jobs has recently gained considerable attention. The papers [2, 4, 5, 10, 13, 24, 25, 29, 31, 36] are just a small sample of work in this area. In fact, the study of computer architectures with parallel processors has prompted the importance of the design and the analysis of good algorithms for scheduling parallel jobs. Because of this background, parallel jobs are often alternatively called multiprocessor tasks. While many different scheduling models and underlying computer architectures have been considered, we focus on scheduling non-malleable parallel jobs on identical parallel machines to minimize the makespan. We refer to [12] for a comprehensive introduction to the scheduling aspect of parallel computing, and to [9] for an overview of results on computational complexity and approximation algorithms; [11] contains a good collection of further references.

Model. We discuss the following class of scheduling problems. We are given m identical parallel machines and a set of n independent, parallel jobs $j = 1, \dots, n$. 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 known as the *width* of job j. Note that we assume that m_j is part of the input; in particular, jobs are non-malleable. Moreover, each job j has a non-negative integer release date r_j at which it becomes available for processing. Any machine can process at most one job at a

[*]Institut für Mathematik, Technische Universität Berlin, Germany, e-mail: johannes@math.tu-berlin.de

time. The objective is to find a feasible schedule of minimal completion time; that is, the makespan is to be minimized. We consider both the preemptive and the non-preemptive variant 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. We denote the preemptive problem by $\mathrm{P}\,|\,m_j,\, r_j,\, \mathrm{pmtn}\,|\,C_{\max}$ and the non-preemptive problem by $\mathrm{P}\,|\,m_j,\, r_j\,|\,C_{\max}$, following the three-field-notation introduced in [19]. We note that some authors use "size$_j$" instead of "m_j" as a symbol to refer to parallel jobs of the nature described before, see, e.g., [9]. ¿From time to time, we shall also refer to the "ancestors" of the studied scheduling problems; they arise as the special case in which $m_j = 1$ for all $j = 1, \dots, n$. Their short-form notation is $\mathrm{P}\,|\,r_j,\, \mathrm{pmtn}\,|\,C_{\max}$ and $\mathrm{P}\,|\,r_j\,|\,C_{\max}$, respectively, and, in the absence of non-trivial release dates, $\mathrm{P}\,|\,\mathrm{pmtn}\,|\,C_{\max}$ and $\mathrm{P}\,|\,|\,C_{\max}$.

Since both parallel job scheduling problems 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. Whereas 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.

Related work. As mentioned earlier, there has lately been a considerable amount of work on parallel job scheduling and we shall restrict ourselves to the subset most closely related to the topic of this paper. In particular, we will not discuss problems with malleable jobs or dedicated machines. We start by pointing out remarkable differences in the behavior of classic machine scheduling problems and parallel job scheduling problems.

While preemptive scheduling of parallel jobs is NP-hard, even in the absence of release dates [8], preemptive scheduling of non-parallel jobs (that is, $m_j = 1$) can be solved in polynomial time [30, 23]. Graham [17] showed that every listscheduling algorithm is a $(2 - 1/m)$–approximation algorithm for the strongly NP-hard problem of scheduling non-parallel jobs without release dates, $\mathrm{P}\,|\,|\,C_{\max}$. Gusfield [20] and Hall and Shmoys [21] observed that Graham's result holds for non-parallel jobs with release dates, $\mathrm{P}\,|\,r_j\,|\,C_{\max}$, as well. If non-parallel jobs are scheduled in non-increasing order of their lengths, listscheduling is a $((4m-1)/3m)$–approximation algorithm for $\mathrm{P}\,|\,|\,C_{\max}$, see [18], and a $3/2$–approximation algorithm for $\mathrm{P}\,|\,r_j\,|\,C_{\max}$, see [6]. Hochbaum and Shmoys [22] and Hall and Shmoys [21] 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 performance guarantee better than $3/2$ for $\mathrm{P}\,|\,m_j\,|\,C_{\max}$, unless $\mathrm{P} = \mathrm{NP}$.

It follows from the work of Garey and Graham [15] on project scheduling with resource constraints that listscheduling has performance guarantee 2 for scheduling parallel jobs without release dates, $\mathrm{P}\,|\,m_j\,|\,C_{\max}$. Turek, Wolf and Yu [36] presented a direct, simplified proof of this result. Feldmann, Sgall and Teng [13] observed that the length of a non-preemptive listschedule is actually at most $2 - 1/m$ times the optimal preemptive makespan. This implies that there is a $(3 - 1/m)$–approximation algorithm for both, $\mathrm{P}\,|\,m_j,\, r_j,\, \mathrm{pmtn}\,|\,C_{\max}$ and $\mathrm{P}\,|\,m_j,\, r_j\,|\,C_{\max}$ (see also [31]).

A problem closely related to $\mathrm{P}\,|\,m_j\,|\,C_{\max}$ 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 [36] pointed out that there is indeed advantage to using non-contiguous machine assignments (in terms of the length of an optimal schedule). ¿From a parallel computer architecture perspective, $\mathrm{P}\,|\,m_j\,|\,C_{\max}$ corresponds to scheduling on a PRAM, while strip-packing is equivalent to scheduling on a linear array of processors [29]. The strip packing problem was first posed by Baker, Coffman, and Rivest [3]. Various authors proposed approximation algorithms with performance guarantees 3 [3, 7, 16], 2.7 [7], 2.5 [34], and 2 [35], respectively. Kenyon and Remila [27] 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, $\mathrm{P}m \mid m_j, \text{ pmtn} \mid C_{\max}$ can be solved as a linear programming problem in polynomial time [4]. Jansen and Porkolab [25] presented an algorithm with running time $\mathrm{O}(n) + \mathrm{poly}(m)$ for the same problem, thereby showing that it cannot be strongly NP-hard, unless $\mathrm{P} = \mathrm{NP}$. They also gave a polynomial-time approximation scheme for the non-preemptive problem with a fixed number of machines, $\mathrm{P}m \mid m_j \mid C_{\max}$, see [24]. Du and Leung [10] showed that this problem is strongly NP-hard for $m \geqslant 5$.

In scheduling, one typically distinguishes between three basic online models, each characterized by a different dynamics of the situation (see, e.g., [32]). 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. In the third 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.

Listscheduling complies with the requirements of both online models “unknown running times” and “jobs arriving over time”. Therefore, listscheduling is a 2–competitive algorithm for $\mathrm{P} \mid m_j \mid C_{\max}$ for the model “unknown running times” without release dates, and its extension to $\mathrm{P} \mid m_j, r_j \mid C_{\max}$ is 3–competitive for both models “unknown running times” and “jobs arriving over time”. Shmoys, Wein and Williamson [33] showed that there is no deterministic online algorithm with a better competitive ratio than $2 - 1/m$ for the online model “unknown running times”, even if every job requires only one machine and arrives at time zero. Also for the non-parallel job case, Chen and Vestjens [6] proved a lower bound of 1.347 for the competitive ratio of any deterministic non-preemptive online algorithm for the online model “jobs arriving over time”. For the online model “scheduling jobs one by one”, the best-known competitive ratio for scheduling non-parallel jobs is achieved by an algorithm of Albers [1], which is 1.923–competitive. She also proved that there is no deterministic online algorithm with a better competitive ratio than 1.852. Fleischer and Wahl [14] presented an algorithm with competitive ratio 1.9201 for $m \to \infty$, which for $m \geqslant 64$ has a better competitive ratio than Albers’ algorithm. No algorithm with constant competitive ratio was known for parallel jobs and the online model “scheduling jobs one by one”. Of course, the lower bound by Albers applies to the setting with parallel jobs as well.

Main results. Whenever a machine falls idle or a job is released, a listscheduling algorithm schedules the first job from a given priority list that is already released and does not require more machines than are available. In preemptive listscheduling, jobs with lower priority can be preempted by jobs with higher priority. We extend the study of this class of greedy-like algorithms, which was started for parallel-job scheduling in [36], to the following directions. For $\mathrm{P} \mid m_j, r_j, \text{ pmtn} \mid C_{\max}$, preemptive m_j–listscheduling, that is, preemptive listscheduling where jobs are in order of non-increasing widths, constructs a schedule that is at most $2 - 1/m$ times as long as an optimal preemptive schedule. The analysis is presented in Section 3. In Section 4, we show that for both $\mathrm{P} \mid m_j, r_j \mid C_{\max}$ and $\mathrm{P} \mid m_j, r_j, \text{ pmtn} \mid C_{\max}$, any non-preemptive listscheduling algorithm produces a schedule with makespan at most twice the makespan of an optimal preemptive schedule. Both results are obtained by a novel way of directly comparing the listschedule with the structure of an optimal preemptive schedule. Not only do they improve upon recent 3–approximation algorithms by Mu’alem and Feitelson [31], but we also provide an instance of $\mathrm{P} \mid m_j, r_j \mid C_{\max}$ that is simultaneously bad for all possible priority lists. That is, no variant of listscheduling can achieve a better performance guarantee than 2 for this problem.

While it is not difficult to see that all listscheduling algorithms with no specific ordering of the list also work in the online settings “unknown running times” and “jobs arriving over time” with corresponding competitive ratios, no listscheduling 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 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. These results are again in sharp contrast to non-parallel job scheduling for which it already follows from Graham’s work [17] that listscheduling is $(2 - 1/m)$–competitive.

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 listscheduling algorithms.

The problem of non-preemptively scheduling parallel jobs of length one to minimize makespan $(\mathrm{P} \,|\, m_j,\, p_j = 1 \,|\, C_{\max})$ is equivalent to the strongly NP-hard Bin Packing problem, as was observed in [4]. It follows that there is no approximation algorithm for scheduling parallel jobs to minimize makespan with 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 *constant, unless P = NP. Here,* $C^*_{\max}$ *denotes the optimal makespan.*

Li [28] gave a polynomial-time algorithm with asymptotic performance guarantee of $31/18$ for $\mathrm{P} \,|\, m_j \,|\, C_{\max}$. Drozdowski [8] observed that if all jobs have length one then the existence of a preemptive schedule of length two implies the existence of a non-preemptive schedule of length two as well. Thus, preemptive scheduling of parallel jobs with length one and therefore $\mathrm{P} \,|\, m_j,\, \mathrm{pmtn} \,|\, C_{\max}$ does not have a better than $3/2$–approximation algorithm either, unless P = NP.

It is well-known that listscheduling algorithms for classic (i.e., non-parallel) 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 we may 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.*

Sketch of 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 $j_1, j_2, \dots, j_{n_t}$ be the jobs that start at time t $(t = t_1, \dots, t_z)$. We assign the jobs $j_1, \dots, j_{n_t}$ to the machines $i_1, \dots, i_{m_t}$ in the following way. Job j_1 is assigned to the first m_{j_1} machines in $\{i_1, \dots, i_{m_t}\}$, job j_2 is assigned to the next m_{j_2} machines in $\{i_1, \dots, i_{m_t}\}$, and so on. Then one can show that for every point $t = t_1, \dots, t_z$ in time there are no more than $n+1$ machine-intervals. Here, a machine-interval is a set of consecutive machines (in the order $1, 2, \dots, m$) 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 $t \in \{t_1, \dots, t_z\}$ the set of machine-intervals with the corresponding jobs. $\square$

In particular, the non-preemptive listscheduling algorithm described in Section 1 computes a feasible schedule (including job-machine assignments) in polynomial time. For preemptive listscheduling, 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. Clearly, the total number of preemptions is bounded from above by $2n$ and the job-machine assignments can again be compactly described. Note also that preemptive listscheduling only preempts at integer points in time, whereas an optimal preemptive schedule may not.

3 Preemptive listscheduling 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–listscheduling 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 non-increasing widths m_j and as many of them are greedily assigned to the machines as feasibly possible.

Theorem 3.1. *For every instance of $P \,|\, m_j,\, r_j,\, pmtn \,|\, C_{\max}$, the length of the schedule constructed by the preemptive m_j–listscheduling 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 listschedule, and let C_e be its completion time. We divide the time horizon $(0, C_e]$ into intervals $(t, t+1]$ of length one ($t = 0, 1, \dots, C_e - 1$). 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–listschedule. It follows from the definition of m_j–listscheduling 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 listschedule 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 from which on every time-slot contains a job j with $m_j \geqslant m_e$ in the listschedule. 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 as 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 the 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. The following calculation shows that this is too much load for so little space.

$$\begin{aligned}
(C^*_{\max} - r_z)m &> (r_e - r_z)m_e + (2C^*_{\max} - \frac{C^*_{\max}}{m} - r_e - p_e)(m - m_e + 1) + p_e m_e \\
\Leftrightarrow \qquad -r_z m &> m(C^*_{\max} - r_e - p_e) + 2m_e(-C^*_{\max} + r_e + p_e) + (C^*_{\max} - r_e - p_e) + \frac{C^*_{\max}}{m}(m_e - 1) - r_z m_e \\
\Leftrightarrow r_z(m - m_e) &< (C^*_{\max} - r_e - p_e)(2m_e - m) - (C^*_{\max} - r_e - p_e) - \frac{C^*_{\max}}{m}(m_e - 1)
\end{aligned}$$

While the term on the left-hand side is non-negative because $m_e \leqslant m/2$, the right-hand side is non-positive because $C^*_{\max} - r_e - p_e \geqslant 0$. Therefore, we have a contradiction and $C_e \leqslant (2 - 1/m)C^*_{\max}$. □

It is not hard to show that this approximation bound of $(2 - 1/m)$ for (preemptive) listscheduling of (parallel) jobs to minimize makespan is tight. The following instance was already proposed by Graham [17] to show that listscheduling for non-parallel 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, m^2-m of length one, and the last job in the list with length m. The resulting schedule will have no preemptions and has length $m-1+m = 2m-1$, while the optimal (preemptive) schedule has makespan m.

4 Non-preemptive listscheduling of parallel jobs with release dates

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

Theorem 4.1. *For every instance of the problems $P \mid m_j, r_j$, (pmtn) $\mid C_{\max}$, the length of the schedule constructed by any non-preemptive listscheduling-algorithm is at most twice the optimal preemptive makespan.*

Proof. Let OPT be an optimal preemptive schedule for a given instance. Let $C^{OPT}_{\max}$ be the makespan of OPT. We partition the time horizon into periods of same length, such that all jobs in OPT are started, preempted, re-started and completed only at the beginning or the end of a period. By scaling, we may assume that such a period starts at time $s - 1$ and ends at time s (for some non-negative integer s), and is of unit length. We call it *time-slot* s. The part of job j in one time-slot is called *slice* of j. The release date of a slice is the release date of its job. A slice of job j is wide if $m_j > m/2$, and it is called small, otherwise. The proof will refer to slices of jobs instead of jobs since this is the level of granularity needed to prove the result. In particular, we will also identify the slices of jobs in the listschedule. Let LS be the schedule constructed by the listscheduling-algorithm, and let $C^{LS}_{\max}$ be its makespan.

Two disjoint sets of time-slots E_1 and E_2 of equal size ($|E_1| = |E_2|$) can be *matched* if the total load of all slices in $E_1 \cup E_2$ exceeds $E_1 m$.

Among others, we make use of the following trivial lower bounds for the optimal makespan:

$$C^{OPT}_{\max} \geqslant \frac{1}{m}\sum_{j=1}^{n} p_j m_j \ , \tag{1}$$

$$C_{\max}^{OPT} \geqslant r_j + p_j, \text{for all } j = 1, \ldots, n \ . \tag{2}$$

Let $0 = r_0 < r_1 < r_2 < \cdots < r_z$ be the different release dates of the given instance. For $k = 0, \ldots, z$, 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 following three properties.

Observation 1. $D(r_h) \cap \{r_k + 1, \ldots, C_{\max}^{LS}\} \subseteq D(r_k)$ 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 $k = 0, \ldots, z$.

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 $S(C_{\max}^{LS})$ be the set of all time-slots in LS, thus $S(C_{\max}^{LS}) = \{1, \ldots, C_{\max}^{LS}\}$. We define the load of a set of time-slots as the sum of the loads of all slices in these time-slots.

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

Observation 5. If there exists a $k \leqslant z$ with $S(C_{\max}^{LS}) \setminus S(r_k) = \emptyset$, then the claim of Theorem 4.1 is true. Reason: If $S(C_{\max}^{LS}) \setminus S(r_k) = \emptyset$ for a $k \leqslant z$, then all time-slots between the time r_k and $C_{\max}^{LS}$ are part of $D(r_k)$, therefore $C_{\max}^{LS} = r_k + |D(r_k)|$. On the other hand, $C_{\max}^{OPT} > r_k$ and $C_{\max}^{OPT} \geqslant |D(r_k)|$. This implies $C_{\max}^{OPT} > \frac{r_k + |D(r_k)|}{2} = \frac{C_{\max}^{LS}}{2}$.

Therefore, we assume henceforth that $S(C_{\max}^{LS}) \setminus S(r_k) \neq \emptyset$ 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, since 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 $C_{\max}^{LS}$: Either there is a release date r_h, $h \in \{0, \ldots, z\}$, such that there is more load than $\frac{|S(C_{\max}^{LS}) \setminus S(r_h)|m}{2}$ in $S(C_{\max}^{LS}) \setminus S(r_h)$, 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 $S(C_{\max}^{LS}) \setminus S(r_h)$ is more than $\frac{|S(C_{\max}^{LS}) \setminus S(r_h)|m}{2}$.

It follows that the load in $S(C_{\max}^{LS}) \setminus S(r_z)$ cannot be more than $\frac{|S(C_{\max}^{LS}) \setminus S(r_z)|m}{2}$. Together with Observation 5, this implies that there is a time-slot in $S(C_{\max}^{LS}) \setminus S(r_z) \subseteq \{r_z + 1, \ldots, C_{\max}^{LS}\}$ containing a small job. Let e be a small job in $S(C_{\max}^{LS}) \setminus S(r_z)$ 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 $S(C_{\max}^{LS}) \setminus S(r_e)$ is at most $\frac{|S(C_{\max}^{LS}) \setminus S(r_e)|m}{2}$.

We partition the set $S(C_{\max}^{LS}) \setminus S(r_e)$ into the disjoint sets $\bar{E}$, $E := E_1 \cup E_2$, and $\tilde{E}$. Let E be the set of time-slots in $S(C_{\max}^{LS}) \setminus S(r_e)$ 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 $S(C_{\max}^{LS}) \setminus S(r_e)$ containing no slice of e and which are before r_z. Let $\tilde{E}$ be the set of time-slots in $S(C_{\max}^{LS}) \setminus S(r_e)$, which do not contain a slice of e and which come after r_z. Finally, $D_{r_e}^e$ is the set of all time-slots between r_e and r_z that contain a slice of e, but which are not element of the set $S(C_{\max}^{LS}) \setminus S(r_e)$, 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 toal number of busy machines in both time-slots exceeds m.

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

Observation 8. $|E_2 \setminus D_{r_z}^{E}| > |\tilde{E} \setminus D_{r_z}^{\tilde{E}}|$.

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 $S(C_{\max}^{LS}) \setminus S(r_e)$ contains in total more load than $(|\tilde{E}| + |E| - |\tilde{E}|)m + \frac{(|\bar{E}| - |E| + |\tilde{E}|)m}{2} = \frac{(|\bar{E}| + |E| + |\tilde{E}|)m}{2} = \frac{|S(C_{\max}^{LS}) \setminus S(r_e)|m}{2}$, which contradicts the assumption of Case 1.

Case 1.A.2: $|\bar{E}| < |E| - |\tilde{E}|$.
In this case, we have $|E| > \frac{|S(C_{\max}^{LS}) \setminus S(r_e)|}{2}$. With the lower bound (2) and Observation 3 follows $C_{\max}^{OPT} \geqslant r_e + |E| > r_e + \frac{|S(C_{\max}^{LS}) \setminus S(r_e)|}{2} = r_e + \frac{C_{\max}^{LS} - r_e - |D(r_e)|}{2} = \frac{C_{\max}^{LS}}{2} + \frac{r_e - |D(r_e)|}{2} \geqslant \frac{C_{\max}^{LS}}{2}$, 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 $S(C_{\max}^{LS}) \setminus S(r_e)$ exceeds $(|E_1| + |E_2|)m + \frac{(|\bar{E}|-|E_1|)m}{2} + \frac{(|\tilde{E}|-|E_2|)m}{2} = \frac{(|\bar{E}|+|E|+|\tilde{E}|)m}{2} = \frac{|S(C_{\max}^{LS}) \setminus S(r_e)|m}{2}$. 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 $\bar{E}$. With Observations 7 and 8 follows that any set of $|\tilde{E} \setminus D_{r_z}^{\tilde{E}}|$ many time-slots in $E_2 \setminus D_{r_z}^{E}$ can be matched with the time-slots in $\tilde{E} \setminus D_{r_z}^{\tilde{E}}$. Furthermore, $|D_{r_z}^{E}| < |D_{r_z}^{\tilde{E}}|$ because of $|\tilde{E}| \geqslant |E_2|$ and Observation 8. Therefore, the set $D_{r_z}^{E}$ can be matched with any set of $|D_{r_z}^{E}|$ many time-slots in $D_{r_z}^{\tilde{E}}$. Since $|E_2| \leqslant |\tilde{E}|$, we obtain the following inequality: $|D_{r_z}^{\tilde{E}}| - |D_{r_z}^{E}| \geqslant |E_2 \setminus D_{r_z}^{E}| - |\tilde{E} \setminus D_{r_z}^{\tilde{E}}|$. Hence, there are $|E_2 \setminus D_{r_z}^{E}| - |\tilde{E} \setminus D_{r_z}^{\tilde{E}}|$ many time-slots in the remaining $|D_{r_z}^{\tilde{E}}| - |D_{r_z}^{E}|$ many time-slots in $D_{r_z}^{\tilde{E}}$ that can be matched with the set of the remaining $|E_2 \setminus D_{r_z}^{E}| - |\tilde{E} \setminus D_{r_z}^{\tilde{E}}|$ many time-slots in $E_2 \setminus D_{r_z}^{E}$. Let D_{rest} be the set of the $|D_{r_z}^{\tilde{E}}| - |D_{r_z}^{E}| - (|E_2 \setminus D_{r_z}^{E}| - |\tilde{E} \setminus D_{r_z}^{\tilde{E}}|)$ many presently unmatched time-slots in $D_{r_z}^{\tilde{E}}$. By Observation 2, there is a set D_{rest}^{match} of time-slots in $\{r_z - (|D_{r_z}^{\tilde{E}}| - |D_{r_z}^{E}| - (|E_2 \setminus D_{r_z}^{E}| - |\tilde{E} \setminus D_{r_z}^{\tilde{E}}|)) + 1, \dots, r_z\}$ that can be matched with D_{rest}. We combine the time-slots in $D_{r_z}^{\tilde{E}}$ that are matched with a time-slot in $D(r_e) \cap D_{rest}^{match}$, thus with a time-slot in $D_{r_e}^{e}$, which is therefore not part of the set $S(C_{\max}^{LS}) \setminus S(r_e)$, in the set D_{spare}. The load in $S(C_{\max}^{LS}) \setminus S(r_e)$ is at most $\frac{|S(C_{\max}^{LS}) \setminus S(r_e)|m}{2}$. Since every time-slot in D_{spare} contains more load than $\frac{m}{2}$, it follows that the load in $(S(C_{\max}^{LS}) \setminus S(r_e)) \setminus D_{spare}$ is bounded from above by $\frac{|(S(C_{\max}^{LS}) \setminus S(r_e)) \setminus D_{spare}|m}{2}$.

If every time-slot in $E_1 \setminus E_1^1$ has been matched with a time-slot in D_{rest}, then every time-slot in E has been matched with another time-slot in $((S(C_{\max}^{LS}) \setminus S(r_e)) \setminus D_{spare}) \setminus E$. Each of the remaining time-slots in $((S(C_{\max}^{LS}) \setminus S(r_e)) \setminus D_{spare}) \setminus E$ contains more than $\frac{m}{2}$ load. Thus, $(S(C_{\max}^{LS}) \setminus S(r_e)) \setminus D_{spare}$ contains more load than $\frac{|(S(C_{\max}^{LS}) \setminus S(r_e)) \setminus D_{spare}|m}{2}$, 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, $|D_{spare}| \leqslant |D_{r_e}^{e}|$. Furthermore, $|E| > |\bar{E}| + |\tilde{E} \setminus D_{r_z}^{\tilde{E}}| + |D_{r_z}^{E}| + |E_2 \setminus D_{r_z}^{E}| - |\tilde{E} \setminus D_{r_z}^{\tilde{E}}| + (|D_{r_z}^{\tilde{E}}| - |D_{r_z}^{E}| - (|E_2 \setminus D_{r_z}^{E}| - |\tilde{E} \setminus D_{r_z}^{\tilde{E}}|)) - |D_{spare}| = |\bar{E}| + |\tilde{E}| - |D_{r_z}^{\tilde{E}}| + |D_{r_z}^{E}| + |E_2| - |D_{r_z}^{E}| - |\tilde{E}| + |D_{r_z}^{\tilde{E}}| + |D_{r_z}^{\tilde{E}}| - |D_{r_z}^{E}| - |E_2| + |D_{r_z}^{E}| + |\tilde{E}| - |D_{r_z}^{\tilde{E}}| - |D_{spare}| = |\bar{E}| + |\tilde{E}| - |D_{spare}|$. It follows that $|E| > \frac{|(S(C_{\max}^{LS}) \setminus S(r_e)) \setminus D_{spare}|}{2}$, and with (2) we conclude that $C_{\max}^{OPT} \geqslant r_e + |E| + |D_{r_e}^{e}| > r_e + |D_{r_e}^{e}| + \frac{|(S(C_{\max}^{LS}) \setminus S(r_e)) \setminus D_{spare}|}{2} = r_e + |D_{r_e}^{e}| + \frac{C_{\max}^{LS}}{2} - \frac{r_e}{2} - \frac{|D(r_e)|}{2} - \frac{|D_{spare}|}{2} \geqslant \frac{C_{\max}^{LS}}{2} + \frac{r_e - |D(r_e)|}{2} + \frac{|D_{r_e}^{e}| - |D_{spare}|}{2} \geqslant \frac{C_{\max}^{LS}}{2}$, and herewith the correctness of Theorem 4.1 in this case.

A schematic illustration of the relevant part of the listschedule LS for this case is given in Figure 1. The red bar underneath the picture indicates the time-slots from the set $S(r_e)$ (that are visible in this part of LS). The blue bar marks the time-slots in $S(C_{\max}^{LS})$ and the green bars mark the time-slots from the set $(S(C_{\max}^{LS}) \setminus S(r_e)) \setminus D_{spare}$. The red bar within LS corresponds to the job e. Sets of hatched time-slots with same color have been matched.

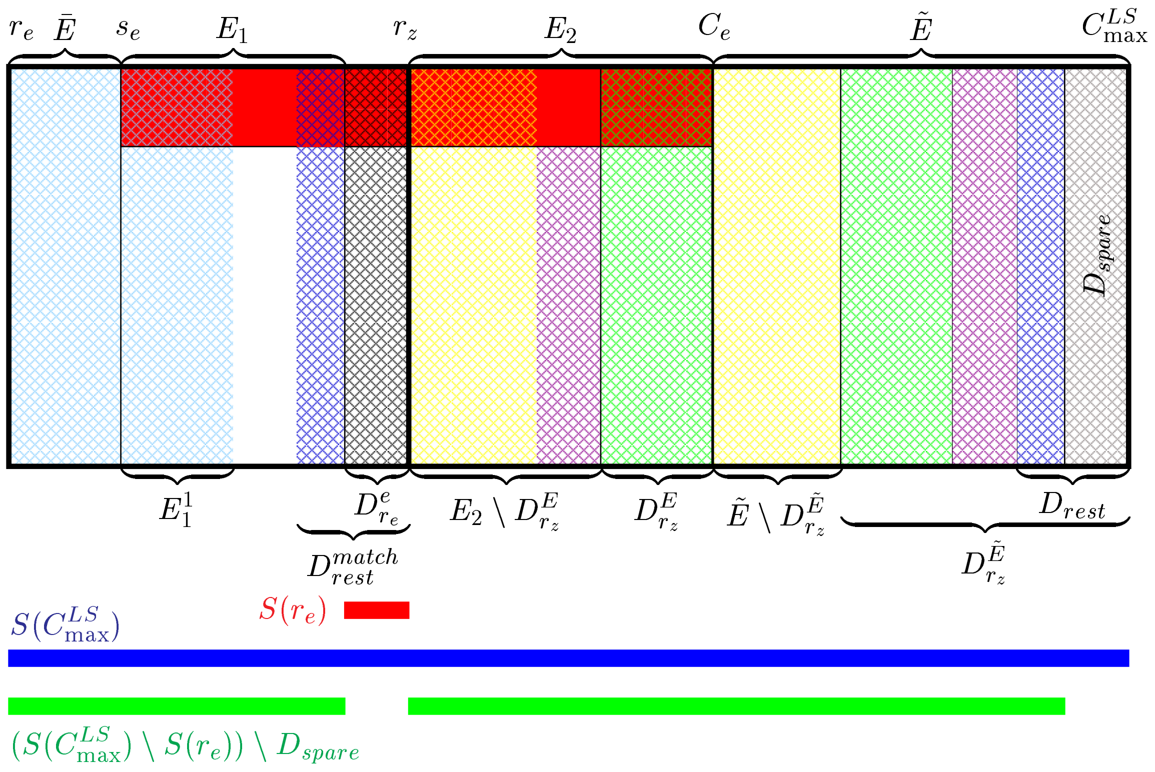

Figure 1: Illustration of Case 1.B.2.

Case 2. There is a release date $r_h < C_{\max}^{LS}$ such that $S(C_{\max}^{LS})\setminus S(r_h)$ contains more load than $\frac{|S(C_{\max}^{LS})\setminus S(r_h)|m}{2}$. Let r_k be the smallest release date of this kind. If $r_k = 0$, the theorem follows from (1). 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 $S(C_{\max}^{LS}) \setminus S(r_h) = (S(C_{\max}^{LS}) \setminus S(r_k)) \cup (S(r_k) \setminus S(r_h)) = S(C_{\max}^{LS}) \setminus S(r_k)$. Since $S(C_{\max}^{LS}) \setminus S(r_k)$ contains more load than $\frac{|S(C_{\max}^{LS})\setminus S(r_k)|m}{2}$, it follows that the set $S(C_{\max}^{LS}) \setminus S(r_h)$ contains more load than $\frac{|S(C_{\max}^{LS})\setminus S(r_h)|m}{2}$, a contradiction to the definition of r_k.

Observation 10. The load of the set $S(r_k) \setminus S(r_h)$ is at most $\frac{|S(r_k)\setminus S(r_h)|m}{2}$, for every release date $r_h < r_k$.
Reason: If the load in $S(r_k) \setminus S(r_h)$ was more than $\frac{|S(r_k)\setminus S(r_h)|m}{2}$, we would be able, with the help of Observation 4, to merge the loads in $S(r_k) \setminus S(r_h)$ and $S(C_{\max}^{LS}) \setminus S(r_k)$. The cumulated load in $S(C_{\max}^{LS}) \setminus S(r_h)$ would then exceed $\frac{|S(C_{\max}^{LS})\setminus S(r_h)|m}{2}$, a contradiction to the definition of r_k.

Observation 11. Every wide slice in $S(C_{\max}^{LS}) \setminus S(r_k)$ 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 $S(C_{\max}^{LS}) \setminus S(r_k)$.

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

We henceforth assume that there is at least one small job in $S(C_{\max}^{LS}) \setminus S(r_k)$ which has been released before r_k. We call such small jobs, which are released before r_k but completed by the listscheduling-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 $\tilde{D}_{r_u}^{before}$ be the set of time-slots in $D(r_u) \cap \{r_u + 1, \dots, r_k\}$ which contain no slice of u. Let $D_{r_u}^{before}$ be the set of time-slots in $D(r_u) \cap \{r_u + 1, \dots, r_k\}$ that contain a slice of u. Let $D_{r_u}^{after} := D(r_u) \cap \{r_k + 1, \dots, C_{\max}^{LS}\}$. We partition the time-slots between date r_u and r_k into the disjoint sets U, $\bar{U}$, $\tilde{D}_{r_u}^{before}$, and $D_{r_u}^{before}$. Let U be the set of time-slots in $\{r_u + 1, \dots, r_k\} \setminus D_{r_u}^{before}$ that contain a slice of the job u. Let $\bar{U}$ be the set of time-slots in $\{r_u + 1, \dots, r_k\} \setminus \tilde{D}_{r_u}^{before}$ that do not contain a slice of the job u. Hence, the set $\{r_u + 1, \dots, r_k\} \cup (D(r_k) \setminus D_{r_u}^{after})$ is composed of the disjoint sets of time-slots U, $\bar{U}$, $\tilde{D}_{r_u}^{before}$, $D_{r_u}^{before}$, and $D(r_k) \setminus D_{r_u}^{after}$.

Observation 12. The term $|\bar{U}| + |\tilde{D}_{r_u}^{before}|$ 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 $S(r_k) \setminus S(r_u)$ contains at most $\frac{|S(r_k) \setminus S(r_u)| m}{2}$ 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 $\tilde{D}_{r_u}^{before}$. Since each time-slot in $D(r_k) \setminus D_{r_u}^{after}$ contains a wide slice, more than $\frac{m}{2}$ machines are busy in each of them. Because of Observation 2, for every time-slot $s_1 \in D(r_k) \setminus D_{r_u}^{after}$ there exists another time-slot $s_2 \in \{r_k - |D(r_k) \setminus D_{r_u}^{after}| + 1, \dots, r_k\}$ such that s_1 and s_2 can be matched. This way at most $|D_{r_u}^{before}|$ many time-slots from $D(r_k) \setminus D_{r_u}^{after}$ are matched with a time-slot in $D_{r_u}^{before}$. We combine those time-slots to the set D_{spare}. We have $|D_{spare}| \leqslant |D_{r_u}^{before}|$. Since $S(r_k) \setminus S(r_u)$ contains at most $\frac{|S(r_k) \setminus S(r_u)| m}{2}$ load, it follows that $(S(r_k) \setminus S(r_u)) \setminus D_{spare}$ contains at most $\frac{|(S(r_k) \setminus S(r_u)) \setminus D_{spare}| m}{2}$ load, because every time-slot in D_{spare} is loaded by more than $\frac{m}{2}$. The set $(S(r_k) \setminus S(r_u)) \setminus D_{spare}$ decomposes into the disjoint sets $\bar{U}$, U, and $(D(r_k) \setminus D_{r_u}^{after}) \setminus D_{spare}$. If $|U| \leqslant |\bar{U}| + |(D(r_k) \setminus D_{r_u}^{after}) \setminus D_{spare}|$, we could match every time-slot in U with another time-slot in $((S(r_k) \setminus S(r_u)) \setminus D_{spare}) \setminus U$. The remaining time-slots in $((S(r_k) \setminus S(r_u)) \setminus D_{spare}) \setminus U$ contain each more than $\frac{m}{2}$ load. Thus the load in $(S(r_k) \setminus S(r_u)) \setminus D_{spare}$ would exceed $\frac{|(S(r_k) \setminus S(r_u)) \setminus D_{spare}| m}{2}$, in contradiction to our earlier observations regarding this set. We conclude that $|U| > |\bar{U}| + |D(r_k) \setminus D_{r_u}^{after}| - |D_{r_u}^{before}|$. Combining this inequality with Observation 12 and 3 results in $\tilde{u} + |D(r_k)| < |U| + |\tilde{D}_{r_u}^{before}| + |D_{r_u}^{before}| + |D_{r_u}^{after}| \leqslant |U| + |D(r_u)| \leqslant |U| + r_u \leqslant r_k$.

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 $S(r_k) \setminus S(r_{k-1})$ exceeds $\frac{|S(r_k) \setminus S(r_{k-1})| m}{2}$.

Let us consider the set $S(C_{\max}^{LS}) \setminus S(r_k)$. The load between r_k and $C_{\max}^{LS}$, excluding the load in the time-slots in $D(r_k)$, is more than $\frac{(C_{\max}^{LS} - r_k - |D(r_k)|) m}{2}$. 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 $\frac{(C_{\max}^{LS} - r_k - |D(r_k)|) m}{2} - \frac{\tilde{u} m}{2}$. Consequently, $C_{\max}^{OPT} > r_k + \frac{C_{\max}^{LS} - r_k - |D(r_k)|}{2} - \frac{\tilde{u}}{2} > \frac{C_{\max}^{LS}}{2}$.

A schematic illustration of the relevant part of the listschedule LS for this case is given in Figure 2. The red bar underneath the picture indicates the time-slots from the set $S(r_u)$, the blue bars mark the visible time-slots in $S(r_k)$, and the green bars mark the time-slots from the set $(S(r_k) \setminus S(r_u)) \setminus D_{spare}$. The sets of hatched time-slots with same color have been matched.

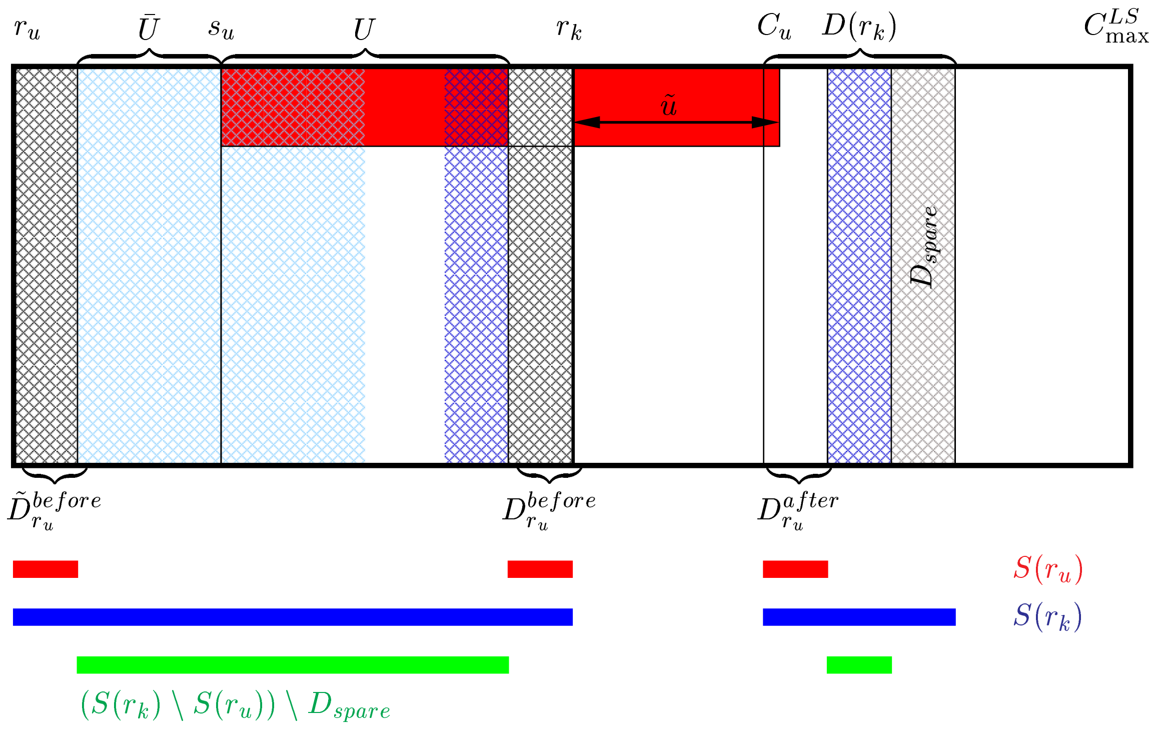

Figure 2: Illustration of Case 2.

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.

Corollary 4.2. *For all instances of* $P \,|\, m_j,\, r_j,\, pmtn \,|\, C_{\max}$*, an optimal non-preemptive schedule is at most twice as long as an optimal preemptive schedule. This bound is tight.*

Chen and Vestjens [6] proved the existence of a priority order that shows that listscheduling according to this job order achieves a performance guarantee better than 2 for the problem with non-parallel jobs $P \,|\, r_j \,|\, C_{\max}$. If longer jobs are considered first, the resulting p_j–listscheduling algorithm (also called LPT rule) is a $3/2$–approximation algorithm for $P \,|\, r_j \,|\, C_{\max}$. In contrast, the following example shows that for non-preemptive scheduling with parallel jobs and release dates, no listscheduling-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 filling the time-slots 2 to $m-2$

completely with big jobs and the last two time-slots with the small jobs. The first time-slot remains empty. Each listscheduling-algorithm receives at time 0 only one job, namely the first small job to schedule. This job thus starts at time 0. At time 1 the listscheduling algorithm receives the second small job and the first big job. Since there are not enough idle machines left to schedule the big job, it assigns the second small job to start at time 1. The same happens at the following point in times, 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 listscheduling algorithm and the optimal makespan is $2 - 2/m$.

Note that the variant of listscheduling in which jobs are assigned to machines in order of their priorities (sometimes called serial or job-based listscheduling) does not lead to improved performance guarantees either, even if jobs are in order of non-increasing widths (Example 3.2) or non-increasing processing times (Example 4.3). Job-based listscheduling for parallel job scheduling is further discussed in [26, 36].

5 Online scheduling of parallel jobs

In this section, we consider online scheduling of parallel jobs. In an online environment, parts of the input data are not known in advance. We consider the following three models. In the model "scheduling jobs one by one", all jobs are given in a list and only the first job in the list is known to the scheduler. The next job from the list becomes visible only after the pevious job was scheduled. Once a job is scheduled, its assignment cannot be changed. The jobs do not have release dates. In the model "unknown running times", the processing time of a job becomes only known at its completion time. This model has two sub-variants, with and without release dates. In the latter case, no information on jobs is known before their release dates. In the model "jobs arriving over time", jobs become fully known at their release date, but no information on them is known beforehand.

Since listscheduling (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 [15] that listscheduling 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 implies that listscheduling is 2–competitive. Theorem 3.1 yields the $(2 - 1/m)$–competitiveness of preemptive m_j–listscheduling for the preemptive versions of the online models "unknown running times" and "jobs arriving over time". Shmoys, Wein and Williamson [33] showed for all 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 non-parallel.

For the online model "scheduling jobs one by one", listscheduling achieves a competitive ratio of $2 - 1/m$ for non-parallel jobs [17]. In contrast, listscheduling of parallel jobs does not have a constant competitive ratio, which follows, for example, from the instance given in the proof of Theorem 5.1 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 in this section a deterministic 12–competitive algorithm. We show first, that 2.25 is a lower bound for the competitive ratio of any deterministic online algorithm for the model "scheduling jobs one by one" with parallel jobs. This result shows again that scheduling parallel jobs is significantly harder than scheduling non-parallel jobs. For non-parallel job scheduling, deterministic online algorithms are known with a competitive ratio smaller than 2 [1, 14].

Theorem 5.1. *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 $\mathrm{P}\,|\,m_j\,|\,C_{\max}$ 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 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 i–th job of width 1 in the list, and let d_i be the i–th job of width m, $i = 1, \dots, m$. 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 $p_{k_i} + \sum_{j=1}^{i-1} p_{d_j}$. If the instance ends with the i–th 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}, $i = 1, \dots, 10$, 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). A computer program showed 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. □

Note that due to limited computational power (and/or poor design of the complete enumeration algorithm) we could not improve this bound. We are certain that it can be strengthened. The following algorithm is the first algorithm with a constant competitive ratio for scheduling parallel jobs one by one.

Algorithm 5.2.

1. Partition the time axis into the intervals $I_i := [2^i, 2^{i+1}]$, $i = 0, 1, \dots$.
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 5.3. *The competitive ratio of Algorithm* 5.2 *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 5.2. 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 non-empty interval of S. Hence, $C^S_{\max} \leqslant 2^{\ell+1} - 1$.

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 5.2 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 5.2 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 $C^S_{\max}/C^*_{\max} \leqslant \frac{3(2^{\ell+1}-1)}{2^{\ell-1}} < 12$. □

The following table gives an overview of the best known results for scheduling parallel jobs to minimize makespan, offline and online. 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 are under the assumption that P ≠ NP.

model	release dates	preemptive	non-preemptive
offline	without r_j	$\leqslant 2 - 1/m$ [13] NP-hard [8]	$\leqslant 2 - 1/m$ [13] $\geqslant 3/2$
	with r_j	$\leqslant 2 - 1/m$ NP-hard [8]	< 2 $\geqslant 3/2$
online model “scheduling jobs one by one”		— —	< 12 > 2.25
online model “unknown running times”	without r_j	$\leqslant 2 - 1/m$ [13] $\geqslant 2 - 1/m$ [33]	$\leqslant 2 - 1/m$ [13] $\geqslant 2 - 1/m$ [33]
	with r_j	$\leqslant 2 - 1/m$ $\geqslant 2 - 1/m$ [33]	< 2 $\geqslant 2 - 1/m$ [33]
online model “jobs arriving over time”	with r_j	$\leqslant 2 - 1/m$ open	< 2 $\geqslant 1.347$ [6]

Acknowledgments. I am grateful to Martin Skutella for directing me to this interesting research area, and for several useful comments and discussions. In particular, he was the advisor of my Diploma thesis from which the results presented here originated [26].

References

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

[2] A. K. Amoura, 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-Verlag, Berlin, 1997, 1–12.

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

[4] J. Błażewicz, M. Drabowski, and J. Węglarz, Scheduling multiprocessor tasks to minimize schedule length, IEEE Transactions on Computers 35 (1986), 389–393.

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

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

[7] E. G. Coffman, 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 (1980), 808–826.

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

[9] M. Drozdowski, Scheduling multiprocessor tasks — an overview, European Journal of Operational Research 64 (1996), 215–230.

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

[11] D. G. Feitelson, A survey of scheduling in multiprogrammed parallel systems, Research Report RC 19790 (87657), IBM T. J. Watson Research Center, October 1994, revised August 1997.

[12] D. G. Feitelson, 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, 1–34.

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

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

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

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

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

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

[19] R. L. Graham, 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 (1979), 287–326.

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

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

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

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

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

[25] K. Jansen and L. Porkolab, Preemptive parallel task scheduling in $\mathcal{O}(n) + poly(m)$ time, in D. T. Lee and S.-H. Teng (eds.), Algorithms and Computation, Lecture Notes in Computer Science 1969, Springer-Verlag, Berlin, 2000, 398–409.

[26] B. Johannes, Obere und untere Schranken für die Güte von Heuristiken und Relaxierungen im Maschinen Scheduling, Diplomarbeit, Institut für Mathematik, Technische Universität Berlin, Germany, June 2001.

[27] C. Kenyon and E. Remila, Approximative strip packing, Proceedings of the 37th IEEE Symposium on Foundations of Computer Science, 1996, 31–36.

[28] K. Li, Analysis of an approximation algorithm for scheduling independent parallel tasks, Discrete Mathematics and Theoretical Computer Science 3 (1999), 155–166.

[29] W. Ludwig and P. Tiwari, Scheduling malleable and nonmalleable tasks, Proceedings of the 5th ACM-SIAM Symposium on Discrete Algorithms, 1994, 167–176.

[30] R. McNaughton, Scheduling with deadlines and loss functions, Management Science 6 (1959), 1–12.

[31] A. W. Mu'alem 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, December 1999.

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

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

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

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

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

Reports from the group

“Combinatorial Optimization and Graph Algorithms”

of the Department of Mathematics, TU Berlin

716/2001 *Christian Liebchen:* The Periodic Assignment Problem (PAP) May Be Solved Greedily

711/2001 *Esther M. Arkin, Michael A. Bender, Sándor P. Fekete, Joseph S. B. Mitchell, and Martin Skutella:* The Freeze-Tag Problem: How to Wake Up a Swarm of Robots

710/2001 *Esther M. Arkin, Sándor P. Fekete, and Joseph S. B. Mitchell:* Algorithms for Manufacturing Paperclips and Sheet Metal Structures

705/2000 *Ekkehard Köhler:* Recognizing Graphs without Asteroidal Triples

704/2000 *Ekkehard Köhler:* AT-free, coAT-free Graphs and AT-free Posets

702/2000 *Frederik Stork:* Branch-and-Bound Algorithms for Stochastic Resource-Constrained Project Scheduling

700/2000 *Rolf H. Möhring:* Scheduling under uncertainty: Bounding the makespan distribution

698/2000 *Sándor P. Fekete, Ekkehard Köhler, and Jürgen Teich:* More-dimensional packing with order constraints

697/2000 *Sándor P. Fekete, Ekkehard Köhler, and Jürgen Teich:* Extending partial suborders and implication classes

696/2000 *Sándor P. Fekete, Ekkehard Köhler, and Jürgen Teich:* Optimal FPGA module placement with temporal precedence constraints

695/2000 *Sándor P. Fekete, Henk Meijer, André Rohe, and Walter Tietze:* Solving a “hard” problem to approximate an “easy” one: heuristics for maximum matchings and maximum Traveling Salesman Problems

694/2000 *Esther M. Arkin, Sándor P. Fekete, Ferran Hurtado, Joseph S. B. Mitchell, Marc Noy, Vera Sacristán and Saurabh Sethia:* On the reflexivity of point sets

693/2000 *Frederik Stork and Marc Uetz:* On the representation of resource constraints in project scheduling

691/2000 *Martin Skutella and Marc Uetz:* Scheduling precedence constrained jobs with stochastic processing times on parallel machines

689/2000 *Rolf H. Möhring, Martin Skutella, and Frederik Stork:* Scheduling with AND/OR precedence constraints

685/2000 *Martin Skutella:* Approximating the single source unsplittable min-cost flow problem

684/2000 *Han Hoogeveen, Martin Skutella, and Gerhard J. Woeginger:* Preemptive scheduling with rejection

683/2000 *Martin Skutella:* Convex quadratic and semidefinite programming relaxations in Scheduling

682/2000 *Rolf H. Möhring and Marc Uetz:* Scheduling scarce resources in chemical engineering

681/2000 *Rolf H. Möhring:* Scheduling under uncertainty: optimizing against a randomizing adversary

680/2000 *Rolf H. Möhring, Andreas S. Schulz, Frederik Stork, and Marc Uetz:* Solving project scheduling problems by minimum cut computations (Journal version for the previous Reports 620 and 661)

674/2000 *Esther M. Arkin, Michael A. Bender, Erik D. Demaine, Sándor P. Fekete, Joseph S. B. Mitchell, and Saurabh Sethia:* Optimal covering tours with turn costs

669/2000 *Michael Naatz:* A note on a question of C. D. Savage

667/2000 *Sándor P. Fekete and Henk Meijer:* On geometric maximum weight cliques

666/2000 *Sándor P. Fekete, Joseph S. B. Mitchell, and Karin Weinbrecht:* On the continuous Weber and k-median problems

664/2000 *Rolf H. Möhring, Andreas S. Schulz, Frederik Stork, and Marc Uetz:* On project scheduling with irregular starting time costs

661/2000 *Frederik Stork and Marc Uetz:* Resource-constrained project scheduling: from a Lagrangian relaxation to competitive solutions

Reports may be requested from: Hannelore Vogt-Möller
Fachbereich Mathematik, MA 6–1
TU Berlin
Straße des 17. Juni 136
D-10623 Berlin – Germany

e-mail: moeller@math.TU-Berlin.DE

Reports are also available in various formats from

`http://www.math.tu-berlin.de/coga/publications/techreports/`

and via anonymous ftp as

`ftp://ftp.math.tu-berlin.de/pub/Preprints/combi/Report-`*number*`-`*year*`.ps`