Online Makespan Minimization with Parallel Schedules

Susanne Albers¹ · Matthias Hellwig²

Received: 5 October 2015 / Accepted: 30 May 2016 / Published online: 8 June 2016
© Springer Science+Business Media New York 2016

Abstract  Online makespan minimization is a classical problem in which a sequence of jobs $\sigma = J_1, \ldots, J_n$ has to be scheduled on $m$ identical parallel machines so as to minimize the maximum completion time of any job. In this paper we investigate the problem with an essentially new model of resource augmentation. More specifically, an online algorithm is allowed to build several schedules in parallel while processing $\sigma$. At the end of the scheduling process the best schedule is selected. This model can be viewed as providing an online algorithm with extra space, which is invested to maintain multiple solutions. The setting is of particular interest in parallel processing environments where each processor can maintain a single or a small set of solutions. As a main result we develop a $(4/3 + \varepsilon)$-competitive algorithm, for any $0 < \varepsilon \leq 1$, that uses a constant number of schedules. The constant is equal to $1/\varepsilon^{O(\log(1/\varepsilon))}$. We also give a $(1 + \varepsilon)$-competitive algorithm, for any $0 < \varepsilon \leq 1$, that builds a polynomial number of $(m/\varepsilon)^{O(\log(1/\varepsilon)/\varepsilon)}$ schedules. This value depends on $m$ but is independent of the input $\sigma$. The performance guarantees are nearly best possible. We show that

---

A preliminary version of this paper has appeared in *Proc. 14th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT), 2014.*

S. Albers: Work supported by the German Research Foundation, project Al 464/7-1.

Susanne Albers
albers@in.tum.de
Matthias Hellwig
mhellwig@informatik.hu-berlin.de

1 Department of Computer Science, Technische Universität München, Boltzmannstr. 3, 85748 Garching, Germany
2 Department of Computer Science, Humboldt-Universität zu Berlin, Unter den Linden 6, 10099 Berlin, Germany
any algorithm that achieves a competitiveness smaller than $4/3$ must construct $\Omega(m)$ schedules. Our algorithms make use of novel guessing schemes that (1) predict the optimum makespan of a job sequence $\sigma$ to within a factor of $1 + \varepsilon$ and (2) guess the job processing times and their frequencies in $\sigma$. In (2) we have to sparsify the universe of all guesses so as to reduce the number of schedules to a constant. We remark that the competitive ratios achieved using parallel schedules are considerably smaller than those in the standard problem without resource augmentation. Furthermore they are at least as good and in most cases better than the ratios obtained with other means of resource augmentation for makespan minimization.

**Keywords** Scheduling · Makespan minimization · Online algorithms · Competitive analysis

### 1 Introduction

Makespan minimization is a fundamental and extensively studied problem in scheduling theory. Consider a sequence of jobs $\sigma = J_1, \ldots, J_n$ that has to be scheduled on $m$ identical parallel machines. Each job $J_t$ is specified by a processing time $p_t > 0$, $1 \leq t \leq n$. Preemption of jobs is not allowed. The goal is to minimize the makespan, i.e. the maximum completion time of any job in the constructed schedule. We focus on the online version of the problem where the jobs of $\sigma$ arrive one by one. Each incoming job $J_t$ has to be assigned immediately to one of the machines without knowledge of any future jobs $J_{t'}$, $t' > t$.

Online algorithms for makespan minimization have been studied since the 1960s. In an early paper Graham [25] showed that the famous List scheduling algorithm is $(2 - 1/m)$-competitive. The best online strategy currently known achieves a competitiveness of about 1.92. Makespan minimization has also been studied with various types of resource augmentation, giving an online algorithm additional information or power while processing $\sigma$. The following scenarios were considered. (1) An online algorithm knows the optimum makespan or the sum of the processing times of $\sigma$. (2) An online strategy has a buffer that can be used to reorder $\sigma$. Whenever a job arrives, it is inserted into the buffer; then one job of the buffer is removed and placed in the current schedule. (3) An online algorithm may migrate a certain number or volume of jobs.

In this paper we investigate makespan minimization assuming that an online algorithm is allowed to build several schedules in parallel while processing a job sequence $\sigma$. Each incoming job is sequenced in each of the schedules. At the end of the scheduling process the best schedule is selected. We believe that this is a sensible form of resource augmentation: In the classical online makespan minimization problem, studied in the literature so far, an algorithm constructs a schedule while jobs arrive one by one. Only when all jobs have arrived, the schedule is executed. Hence there is a priori no reason why an algorithm should not be able to construct several solutions, the best of which is finally chosen.

The investigated setting can be viewed as providing an online algorithm with extra space, which is used to maintain several solutions. Very little is known about the value of extra space in the design of online algorithms. Makespan minimization with paral-
lel schedules is of particular interest in parallel processing environments where each processor can take care of a single or a small set of schedules. We develop algorithms that require hardly any coordination or communication among the schedules. Moreover, the proposed setting is interesting w.r.t. the foundations of scheduling theory, giving insight into the value of multiple candidate solutions. Last not least the model represents a possible approach to design good randomized online algorithms, because any randomized algorithm is a probability distribution over a set of candidate schedules. At present no randomized online algorithm is known whose competitiveness is provably smaller than that of deterministic online algorithms. Still an algorithm for parallel schedules has to be augmented by a suitable distribution, which is non-trivial. Our study complements work along another line of research, investigating online algorithms with advice, see e.g. [13,17,36]. In that scenario an online algorithm, at any time, can query some information about future input and thereby achieve an improved solution.

Makespan minimization with parallel schedules was also addressed by Kellerer et al. [32]. However, the paper focused on the restricted setting in which $m = 2$ machines are available. In this paper we explore the problem for a general number $m$ of machines. As a main result we show that a constant number of schedules suffices to achieve a significantly improved competitiveness, compared to the standard setting without resource augmentation. Moreover, the competitive ratios obtained are at least as good and in most cases better than those attained in the other models of resource augmentation mentioned above.

The approach to grant an online algorithm extra space, invested to maintain multiple solutions, could be interesting in other problems as well. The approach is viable in applications where an online algorithm constructs a solution that is used when the entire input has arrived. This is the case, for instance, in basic online graph coloring and matching problems [28,30,34]. The approach is also promising in problems that can be solved by a set of independent agents, each of which constructs a separate solution. Good examples are online navigation and exploration problems in robotics [11,12,16]. Some results are known for graph search and exploration, see e.g. [10,22,33], but the approach has not been studied for geometric environments.

Problem definition: We investigate the problem Makespan Minimization with Parallel Schedules (MPS). As always, the jobs of a sequence $\sigma = J_1, \ldots, J_n$ arrive one by one and must be scheduled non-preemptively on $m$ identical parallel machines. Each job $J_t$ has a processing time $p_t > 0$. In MPS, an online algorithm $A$ may maintain a set $S = \{S_1, \ldots, S_l\}$ of schedules during the scheduling process while jobs of $\sigma$ arrive. Each job $J_t$ is sequenced in each schedule $S_k$, $1 \leq k \leq l$. At the end of $\sigma$, algorithm $A$ selects a schedule $S_k \in S$ having the smallest makespan and outputs this solution. The other schedules of $S$ are deleted.

As we shall show MPS can be reduced to the problem variant where the optimum makespan of the job sequence to be processed is known in advance. Hence let MPS$_{\text{opt}}$ denote the variant of MPS where, prior to the arrival of the first job, an algorithm $A$ is given the value of the optimum makespan $\text{OPT}(\sigma)$ for the incoming job sequence $\sigma$. An algorithm $A$ for MPS or MPS$_{\text{opt}}$ is $\rho$-competitive if, for every job sequence $\sigma$, it outputs a schedule whose makespan is at most $\rho$ times $\text{OPT}(\sigma)$. 

© Springer
Our contribution: We present a comprehensive study of MPS. We develop a \((4/3 + \varepsilon)\)-competitive algorithm, for any \(0 < \varepsilon \leq 1\), that uses a constant number of \(1/\varepsilon O(\log(1/\varepsilon))\) schedules. Furthermore, we give a \((1 + \varepsilon)\)-competitive algorithm, for any \(0 < \varepsilon \leq 1\), that uses a polynomial number of schedules. The number is \((m/\varepsilon) O(\log(1/\varepsilon)/\varepsilon)\), which depends on \(m\) but is independent of the job sequence \(\sigma\). These performance guarantees are nearly best possible. The algorithms are obtained via some intermediate results that may be of independent interest.

First, in Sect. 2 we show that the original problem MPS can be reduced to the variant MPSopt in which the optimum makespan is known. More specifically, given any \(\rho\)-competitive algorithm \(A\) for MPSopt we construct a \((\rho + \varepsilon)\)-competitive algorithm \(A^*(\varepsilon)\), for any \(0 < \varepsilon \leq 1\). If \(A\) uses \(l\) schedules, then \(A^*(\varepsilon)\) uses \(l \cdot \lceil \log(1 + 6\rho/\varepsilon)/\log(1 + 6\rho) \rceil\) schedules. The construction works for any algorithm \(A\) for MPSopt. In particular we could use a 1.5-competitive algorithm by Böhm et al. [14] that assumes that the optimum makespan is known and builds a single schedule. We would obtain a \((1.5 + \varepsilon)\)-competitive algorithm that builds at most \(\lceil \log(1 + 9/\varepsilon)/\log(1 + 2\varepsilon/9) \rceil\) schedules.

We proceed and develop algorithms for MPSopt. In Sect. 3 we give a \((1 + \varepsilon)\)-competitive algorithm, for any \(0 < \varepsilon \leq 1\), that uses \(\lceil 2m/\varepsilon \rceil + 1\lceil \log(2/\varepsilon)/\log(1+\varepsilon/2) \rceil\) schedules. In Sect. 4 we devise a \((4/3 + \varepsilon)\)-competitive algorithm, for any \(0 < \varepsilon \leq 1\), that uses \(1/\varepsilon O(\log(1/\varepsilon))\) schedules. Combining these algorithms with our transformation technique represented by \(A^*(\varepsilon)\), we derive the two algorithms for MPS mentioned in the above paragraph; see also Sect. 5. The number of schedules used by our strategies depends on \(1/\varepsilon\) and exponentially on \(\log(1/\varepsilon)\) or \(1/\varepsilon\). Such a dependence seems inherent if we wish to explore the full power of parallel schedules. The trade-offs resemble those exhibited by PTASs in offline approximation. Recall that the PTAS by Hochbaum and Shmoys [27] for makespan minimization achieves a \((1 + \varepsilon)\)-approximation with a running time of \(O((n/\varepsilon)^{1/\varepsilon^2})\).

In Sect. 6 we present lower bounds. We show that any online algorithm for MPS that achieves a competitive ratio smaller than 4/3 must construct more than \([m/3]\) schedules. Hence the competitive ratio of 4/3 is best possible using a constant number of schedules. We show a second lower bound that implies that the number of schedules of our \((1 + \varepsilon)\)-competitive algorithm is nearly optimal, up to a polynomial factor.

Our algorithms make use of novel guessing schemes. \(A^*(\varepsilon)\) works with guesses on the optimum makespan. Guessing and doubling the value of the optimal solution is a technique that has been applied in other load balancing problems, see e.g. [6]. However here we design a refined scheme that carefully sets and readjusts guesses so that the resulting competitive ratio increases by a factor of \(1 + \varepsilon\) only, for any \(\varepsilon > 0\). Moreover, the readjustment and job assignment rules have to ensure that scheduling errors, made when guesses were to small, are not critical. Our \((4/3 + \varepsilon)\)-competitive algorithm works with guesses on the job processing times and their frequencies in \(\sigma\). In order to achieve a constant number of schedules, we have to sparsify the set of all possible guesses. As far as we know such an approach has not been used in the literature before.

All our algorithms have the property that the parallel schedules are constructed basically independently. The algorithms for MPSopt require no coordination at all
among the schedules. In \( A^*(\varepsilon) \) a schedule only has to report when it fails, i.e. when a guess on the optimum makespan is too small.

The competitive ratios achieved with parallel schedules are considerably smaller than the best ratios of about 1.92 known for the scenario without resource augmentation. Our ratio of \((4/3 + \varepsilon)\), for small \( \varepsilon \), is lower than the competitiveness of about 1.46 obtained in the settings where a reordering buffer of size \( O(m) \) is available or \( O(m) \) jobs may be reassigned. Sanders et al. [39] gave an online algorithm that is \((1 + \varepsilon)\)-competitive if, before the assignment of any job \( J_t \), jobs of processing volume \( 2^{O((1/\varepsilon) \log^2(1/\varepsilon))} p_t \) may be migrated. Hence the total amount of extra resources used while scheduling \( \sigma \) depends on the input sequence. As for online computation with advice, Renault et al. [36] devised an algorithm that is \((1 + \varepsilon)\)-competitive and, per incoming job, queries \( O(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon}) \) bits about future input, for any \( 0 < \varepsilon < 1/2 \).

We remark that our results give a related, improved overall bound. As stated above, our \((1 + \varepsilon)\)-competitive algorithm uses \( \lceil \frac{2m}{\varepsilon} + 1 \rceil \left\lfloor \log(2/e) / \log(1+\varepsilon/2) \right\rfloor \) schedules. The number of bits to encode this schedule number is \( O(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon} \log m) \). Hence a total number of \( O(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon} \log m) \) advice bits are needed to guide an online algorithm in processing a job sequence. The latter number is independent of the number \( n \) of jobs.

**Related work:** Makespan minimization with parallel schedules was first studied by Kellerer et al. [32]. They assume that \( m = 2 \) machines are available and two schedules may be constructed. They show that in this case the optimal competitive ratio is equal to 4/3.

We summarize results known for online makespan minimization without resource augmentation. As mentioned before, \( List \) is \((2 - 1/m)\)-competitive. Deterministic online algorithms with a smaller competitive ratio were presented in [1,9,21,24,29]. The best algorithm currently known is 1.9201-competitive [21]. Lower bounds on the performance of deterministic strategies were given in [1,8,19,26,37,38]. The best bound currently known is 1.88, see [37]. No randomized online algorithm whose competitive ratio is provably below the deterministic lower bound is currently known for general \( m \).

We next review the results for the various models of resource augmentation. Articles [3–5,7,14,15,23,31,32] study makespan minimization assuming that an online algorithm knows the optimum makespan or the sum of the processing times of \( \sigma \). Böhm et al. [14] developed a 1.5-competitive algorithm. Azar and Regev [7] showed that no online algorithm can attain a competitive ratio smaller than 4/3. The setting in which an online algorithm is given a reordering buffer was explored in [18,32]. Englert et al. [18] presented an algorithm that, using a buffer of size \( O(m) \), achieves a competitive ratio of \( W_{-1}(-1/e^2)/(1 + W_{-1}(-1/e^2)) \approx 1.46 \), where \( W_{-1} \) is the Lambert \( W \) function. No algorithm using a buffer of size \( o(n) \) can beat this ratio.

Makespan minimization with job migration was addressed in [2,39]. An algorithm that achieves again a competitiveness of \( W_{-1}(-1/e^2)/(1 + W_{-1}(-1/e^2)) \approx 1.46 \) and uses \( O(m) \) job reassignments was devised in [2]. No algorithm using \( o(n) \) reassignments can obtain a smaller competitiveness. Sanders et al. [39] study a scenario in which before the assignment of each job \( J_t \), jobs up to a total processing volume of \( \beta p_t \) may be migrated, for some constant \( \beta \). For \( \beta = 4/3 \), they present a 1.5-competitive
algorithm. They also show a \((1 + \varepsilon)\)-competitive algorithm, for any \(\varepsilon > 0\), where 
\[ \beta = 2^{O((1/\varepsilon) \log^2 (1/\varepsilon))}. \]

As for memory in online algorithms, Sleator and Tarjan [40] studied the paging problem assuming that an online algorithm has a larger fast memory than an offline strategy. Raghavan and Snir [35] traded memory for randomness in online caching algorithms.

**Notation:** Throughout this paper it will be convenient to associate schedules with algorithms, i.e. a schedule \(S_k\) is maintained by an algorithm \(A_k\) that specifies how to assign jobs to machines in \(S_k\). Thus an algorithm \(\mathcal{A}\) for MPS or MPS\(_\text{opt}\) can be viewed as a family \(\{A_k\}_{k \in K}\) of algorithms that maintain the various schedules. We will write \(\mathcal{A} = \{A_k\}_{k \in K}\). If \(\mathcal{A}\) is an algorithm for MPS\(_\text{opt}\), then the value \(\text{OPT}(\sigma)\) is of course given to all algorithms of \(\{A_k\}_{k \in K}\). Furthermore, throughout this paper, let the **load** of a machine denote the sum of the processing times of the jobs already assigned to that machine.

### 2 Reducing MPS to MPS\(_\text{opt}\)

In this section we will show that any \(\rho\)-competitive algorithm \(\mathcal{A}\) for MPS\(_\text{opt}\) can be used to construct a \((\rho + \varepsilon)\)-competitive algorithm \(\mathcal{A}^*(\varepsilon)\) for MPS, for any \(0 < \varepsilon \leq 1\). The main idea is to repeatedly execute \(\mathcal{A}\) for a set of guesses on the optimum makespan. The initial guesses are small and are increased whenever a guess turns out to be smaller than \(\text{OPT}(\sigma)\). The increments are done in small steps so that, among the final guesses, there exists one that is upper bounded by approximately \((1 + \varepsilon)\text{OPT}(\sigma)\). In the analysis of this scheme we will have to bound machine loads caused by scheduling “errors” made when guesses were too small. Unfortunately the execution of \(\mathcal{A}\), given a guess \(\gamma \neq \text{OPT}(\sigma)\), can lead to undefined algorithmic behavior. As we shall show, guesses \(\gamma \geq \text{OPT}(\sigma)\) are not critical. However, guesses \(\gamma < \text{OPT}(\sigma)\) have to be handled carefully.

So let \(\mathcal{A} = \{A_k\}_{k \in K}\) be a \(\rho\)-competitive algorithm for MPS\(_\text{opt}\) that, given guess \(\gamma\), is executed on a job sequence \(\sigma\). Upon the arrival of a job \(J_t\), an algorithm \(A_k \in \mathcal{A}\) may fail because the scheduling rules of \(A_k\) do not specify a machine where to place \(J_t\) in the current schedule \(S_k\). We define two further conditions when an algorithm \(A_k\) fails. The first one identifies situations where a makespan of \(\rho \gamma\) is not preserved and hence \(\rho\)-competitiveness may not be guaranteed. More precisely, \(A_k\) would assign \(J_t\) to a machine \(M_j\) such that \(\ell(j) + p_t > \rho \gamma\), where \(\ell(j)\) denotes \(M_j\’s\) machine load before the assignment. The second condition identifies situations where \(\gamma\) is not consistent with lower bounds on the optimum makespan, i.e. \(\gamma\) is smaller than the average machine load or the processing time of \(J_t\). Formally, an algorithm \(A_k\) fails if a job \(J_t\), \(1 \leq t \leq n\), has to be scheduled and one of the following conditions holds.

(i) \(A_k\) does not specify a machine where to place \(J_t\) in the current schedule \(S_k\).
(ii) There holds \(\ell(j) + p_t > \rho \gamma\), for the machine \(M_j\) to which \(A_k\) would assign \(J_t\) in \(S_k\).
(iii) There holds \(\gamma < \sum_{t' \leq t} p_{t'}/m\) or \(\gamma < p_t\).

The next lemma shows that guesses \(\gamma \geq \text{OPT}(\sigma)\) are not problematic.
Lemma 1 Let \( A = \{A_k\}_{k \in K} \) be a \( \rho \)-competitive algorithm for MPS\(_{\text{opt}}\) that, given guess \( \gamma \), is executed on a job sequence \( \sigma \) with \( \gamma \geq \text{OPT}(\sigma) \). Then there exists an algorithm \( A_k \in A \) that does not fail during the processing of \( \sigma \) and generates a schedule whose makespan is at most \( \rho \gamma \).

Proof Let \( S_{\text{opt}} \) be an optimal schedule for the job sequence \( \sigma = J_1, \ldots, J_n \). Moreover, let \( \ell(j) \) denote the load of machine \( M_j \) in \( S_{\text{opt}} \), \( 1 \leq j \leq m \). For any \( j \) with \( \ell(j) < \gamma \), define a job \( J'_j \) of processing time \( p'_{j} = \gamma - \ell(j) \). Let \( \sigma' \) be the job sequence consisting of \( \sigma \) followed by the new jobs \( J'_j \). These up to \( m \) jobs may be appended to \( \sigma \) in any order. Obviously \( \text{OPT}(\sigma') = \gamma \). Hence when \( A \) using guess \( \gamma \) is executed on \( \sigma' \), there must exist an algorithm \( A_{k^*} \in A \) that generates a schedule with a makespan of at most \( \rho \gamma \). Since \( \sigma \) is a prefix of \( \sigma' \), this algorithm \( A_{k^*} \) does not fail and generates a schedule with a makespan of at most \( \rho \gamma \), when \( A \) given guess \( \gamma \) is executed on \( \sigma \). \( \Box \)

Algorithm for MPS: We describe our algorithm \( A^*(\epsilon, h) \) for MPS, where \( 0 < \epsilon \leq 1 \) and \( h \in \mathbb{N} \) may be chosen arbitrarily. The construction takes as input any algorithm \( A = \{A_k\}_{k \in K} \) for MPS\(_{\text{opt}}\). For a proper choice of \( h \), \( A^*(\epsilon, h) \) will be \((\rho + \epsilon)\)-competitive, provided that \( A \) is \( \rho \)-competitive.

General scheme: At any time \( A^*(\epsilon, h) \) works with \( h \) guesses \( \gamma_1 < \ldots < \gamma_h \) on the optimum makespan for the incoming job sequence \( \sigma \). These guesses may be adjusted during the processing of \( \sigma \); the update procedure will be described in detail below. For each guess \( \gamma_i \), \( 1 \leq i \leq h \), \( A^*(\epsilon, h) \) executes \( A \). Hence \( A^*(\epsilon, h) \) maintains a total of \( h|K| \) schedules, which can be partitioned into subsets \( S_1, \ldots, S_h \). Subset \( S_i \) contains those schedules generated by \( A \) using \( \gamma_i \), \( 1 \leq i \leq h \). Let \( S_{ik} \in S_i \) denote the schedule generated by \( A_k \) using \( \gamma_i \).

Processing of \( \sigma \): A job sequence \( \sigma \) is processed as follows. Initially, upon the arrival of the first job \( J_1 \), the guesses are initialized as \( \gamma_1 = p_1 \) and \( \gamma_i = (1 + \epsilon)\gamma_{i-1} \), for \( i = 2, \ldots, h \). Each job \( J_t \), \( 1 \leq t \leq n \), is handled in the following way. Of course each such job is placed in every schedule \( S_{ik} \), \( 1 \leq i \leq h \) and \( 1 \leq k \leq |K| \). Algorithm \( A^*(\epsilon, h) \) checks if \( A_k \) using \( \gamma_i \) fails when having to sequence \( J_t \) in \( S_{ik} \). We remark that this check can be performed easily by just verifying if one of the conditions (i–iii) holds. If \( A_k \) using \( \gamma_i \) does not fail and has not failed since the last adjustment of \( \gamma_i \), then in \( S_{ik} \) job \( J_t \) is assigned to the machine specified by \( A_k \) using \( \gamma_i \). Here the initialization of a guess is also regarded as an adjustment. If \( A_k \) using \( \gamma_i \) does fail, then \( J_t \) and all future jobs are always assigned to a least loaded machine in \( S_{ik} \) until \( \gamma_i \) is adjusted the next time.

Updating guesses: Suppose that after the sequencing of \( J_t \) all algorithms of \( A = \{A_k\}_{k \in K} \) using a particular guess \( \gamma_i \) have failed since the last adjustment of this guess. Let \( i^* \) be the largest index \( i \) with this property. Then the guesses \( \gamma_1, \ldots, \gamma_{i^*} \) are adjusted. Set \( \gamma_1 = (1 + \epsilon)\max\{\gamma_h, p_t, \sum_{1 \leq i' \leq t} p_{i'}/m\} \) and \( \gamma_i = (1 + \epsilon)\gamma_{i-1} \), for \( i = 2, \ldots, i^* \). For any readjusted guess \( \gamma_i \), \( 1 \leq i \leq i^* \), algorithm \( A \) using \( \gamma_i \) ignores all jobs \( J_{t'} \) with \( t' < t \) when processing future jobs of \( \sigma \). Specifically, when making scheduling decisions and determining machine loads, algorithm \( A_k \) using \( \gamma_i \) ignores all job \( J_{t'} \) with \( t' < t \) in its schedule \( S_{ik} \). These jobs are also ignored when \( A^*(\epsilon, h) \) checks if \( A_k \) using guess \( \gamma_i \) fails on the arrival of a job. Furthermore, after the assignment
Algorithm \( A^*(\varepsilon, h) \)

1. Set \( \gamma_i = p_1(1 + \varepsilon)^{i-1} \), for \( i = 1, \ldots, h \).
2. At time \( t \) execute the following steps.
   (a) \( J_t \) is sequenced as follows in each \( S_{i,k} \). If \( A_k \) using \( \gamma_i \) fails or has failed since the last adjustment of \( \gamma_i \), then assign \( J_t \) to a least loaded machine. Otherwise assign it to the machine specified by \( A_k \), ignoring jobs that arrived before the last adjustment of \( \gamma_i \). If all algorithms \( \{A_k\}_{k \in K} \) for some \( \gamma_i \) have failed since the last readjustment of \( \gamma_i \), then let \( i^* \) be the largest index with this property. Set \( \gamma_i = (1 + \varepsilon)^i \max \{\gamma_k, p_1, \sum_{t \leq 1} P_t / m\} \), for \( i = 1, \ldots, i^* \). Renumber the guesses such that \( \gamma_1 < \ldots < \gamma_h \).

Fig. 1 The algorithm \( A^*(\varepsilon, h) \)

of \( J_t \), machines in \( S_{i,k} \) machines are renumbered so that \( J_t \) is located on a machine it would occupy if it were the first job of an input sequence.

When guesses have been adjusted, they are renumbered, together with the corresponding schedule sets \( S_i \), such that again \( \gamma_1 < \ldots < \gamma_h \). Hence at any time \( \gamma_1 = \min_{1 \leq i \leq h} \gamma_i \) and \( \gamma_i \geq (1 + \varepsilon) \gamma_{i-1} \), for \( i = 2, \ldots, h \). We also observe that whenever a guess is adjusted, its value increases by a factor of at least \( (1 + \varepsilon)^h \). A summary of \( A^*(\varepsilon, h) \) is given in Fig. 1.

The next lemma upper bounds the value of the smallest guess when \( A^*(\varepsilon, h) \) has finished processing a job sequence \( \sigma \).

**Lemma 2** After \( A^*(\varepsilon, h) \) has processed a job sequence \( \sigma \), there holds \( \gamma_1 \leq (1 + \varepsilon) \text{OPT}(\sigma) \).

**Proof** At any time \( A^*(\varepsilon, h) \) maintains \( h \) guesses. We can view these guesses as being stored in \( h \) variables. A variable is updated whenever its current guess is increased. Hence during the processing of \( \sigma \) a variable may take any position in the sorted sequence of guesses. We analyze the steps in which \( A^*(\varepsilon, h) \) adjusts guesses. We first show that when \( A^*(\varepsilon, h) \) adjusts a guess \( \gamma \), then \( \gamma < \text{OPT}(\sigma) \). We then analyze the value of the smallest guess when \( A^*(\varepsilon, h) \) has finished processing \( \sigma \).

1. Suppose that \( A^*(\varepsilon, h) \) adjusts a guess. More specifically, assume that after the arrival of a job \( J_t \), \( A^*(\varepsilon, h) \) adjust guesses \( \gamma_1, \ldots, \gamma_{i^*} \), where \( i^* \) is the largest index \( i \) such that all algorithms \( \{A_k\}_{k \in K} \) using \( \gamma_i \) have failed. We prove \( \gamma_{i^*} < \text{OPT}(\sigma) \), which implies that every updated guess is smaller than \( \text{OPT}(\sigma) \) because guesses are numbered in order of increasing value. Let \( t^* \), with \( t^* < t \), be the most recent time when the variable storing \( \gamma_{i^*} \) was updated last. If the variable has never been updated since its initialization, then let \( t^* = 1 \). All the algorithms \( \{A_k\}_{k \in K} \) using \( \gamma_{i^*} \) ignore the jobs having arrived before \( J_{t^*} \) when making scheduling decisions for \( J_{t^*}, \ldots, J_t \). Let \( \sigma^* = J_{t^*}, \ldots, J_t \). There holds, \( \text{OPT}(\sigma^*) \leq \text{OPT}(\sigma) \). If \( \gamma_{i^*} \geq \text{OPT}(\sigma) \) held true, then by Lemma 1 there would be an algorithm \( A_{k^*} \in \{A_k\}_{k \in K} \) that, using guess \( \gamma_{i^*} \), does not fail when handling \( \sigma^* \). This contradicts the fact that at time \( t \) all algorithms \( \{A_k\}_{k \in K} \) using \( \gamma_{i^*} \) fail or have failed since the arrival of \( J_{t^*} \).

2. Let \( \gamma_{i^*}^f \) denote the value of the smallest guess when \( A^*(\varepsilon, h) \) has finished processing \( \sigma \). We distinguish two cases depending on whether or not the variable storing \( \gamma_{i^*}^f \) has ever been updated since its initialization. If the variable has never been updated, then \( \gamma_{i^*}^f = p_1(1 + \varepsilon)^{i^*-1} \), for some \( i \in \{1, \ldots, h\} \). If \( i = 1 \), there is nothing to show because \( p_1 \leq \text{OPT}(\sigma) \). If \( i > 1 \), then the initial guess of value \( \gamma_{i-1} = p_1(1 + \varepsilon)^{i-2} \).
must have been adjusted. This implies, as shown above, \( \gamma_{t-1} < \text{OPT}(\sigma) \) and the lemma follows because \( \gamma_1^c = (1 + \varepsilon)\gamma_{t-1} \).

In the remainder of the proof we assume that the variable \( g \) storing \( \gamma_1^c \) has been updated. Consider the last update of \( g \) before the end of \( \sigma \) and suppose that it took place on the arrival of job \( J_{t^*} \). First assume that \( g \) stores the smallest guess, among the \( h \) guesses, before the update. Then \( \gamma_1^c = (1 + \varepsilon) \max \{ \gamma^*, p_{t^*}, \sum_{1 \leq j < t^*} p_j / m \} \), where \( \gamma^* \) is the largest guess before the update. If \( \gamma^* \) is also adjusted on the arrival of \( J_{t^*} \), then we are done because, as shown above, \( \gamma^* < \text{OPT}(\sigma) \) and thus \( \max \{ \gamma^*, p_{t^*}, \sum_{1 \leq j < t^*} p_j / m \} \leq \text{OPT}(\sigma) \). If \( \gamma^* \) is not adjusted on the arrival of \( J_{t^*} \), then \( \gamma_1^c \) is the smallest guess greater than \( \gamma^* \) after the update. By the end of \( \sigma \) guess \( \gamma^* \) must be adjusted since otherwise \( \gamma_1^c \) cannot become the smallest guess. Again \( \gamma^* < \text{OPT}(\sigma) \) and we are done.

Finally assume that before the update \( g \) does not store the smallest guess. Let \( g' \) be the variable that stores the largest guess smaller than that in \( g \). After the update there holds \( \gamma_1^c = (1 + \varepsilon)\gamma \), where \( \gamma \) is the guess stored in \( g' \) after the update. Until the end of \( \sigma \), \( \gamma \) must be adjusted again since otherwise \( \gamma_1^c \) cannot become the smallest guess. Again \( \gamma < \text{OPT}(\sigma) \) and hence \( \gamma_1^c < (1 + \varepsilon)\text{OPT}(\sigma) \).

\[ \square \]

Theorem 1 Let \( A = \{A_k\}_{k \in K} \) be a \( \rho \)-competitive algorithm for MPS_{opt}. Then for any \( 0 < \varepsilon \leq 1 \) and \( h = \lceil \log(1 + \frac{6\rho}{\varepsilon})/\log(1 + \frac{\varepsilon}{3\rho}) \rceil \), algorithm \( A^*(\varepsilon) = A^*(\varepsilon/(3\rho), h) \) for MPS is \( (\rho + \varepsilon) \)-competitive and uses \( h|K| \) schedules.

Proof Throughout the proof let \( h = \lceil \log(1 + \frac{6\rho}{\varepsilon})/\log(1 + \frac{\varepsilon}{3\rho}) \rceil \) and \( A^*(\varepsilon) = A^*(\varepsilon/(3\rho), h) \). Consider an arbitrary job sequence and let \( \gamma_1 \) be the smallest of the \( h \) guesses maintained by \( A^*(\varepsilon) \) at the end of \( \sigma \). Let \( S_1 \) be the set of schedules associated with \( \gamma_1 \), i.e. \( S_1 \) was generated by \( A = \{A_k\}_{k \in K} \) using a series of guesses ending with \( \gamma_1 \). Let \( \gamma(0) < \ldots < \gamma(s) \), with \( s \geq 0 \), be this series and \( g \) be the variable that stored these guesses. Here \( \gamma(0) \) is one of the initial guesses and \( \gamma(s) = \gamma_1 \). The following proof consists of two parts. (1) We first observe that at the end of \( \sigma \) there exists an algorithm \( A_{k^*} \in \{A_k\}_{k \in K} \) that using \( \gamma_1 \) has not failed. (2) We prove that the load of every machine in \( S_{1k^*} \) is upper bounded by \( (\rho + \varepsilon)\text{OPT}(\sigma) \). This establishes the theorem.

(1) The desired statement holds true if \( g \) was set to \( \gamma_1 = \gamma(s) \) upon the arrival of a job \( J_t \), with \( t < n \) because the failure of all algorithms \( \{A_k\}_{k \in K} \) using \( \gamma_1 \) would have caused an adjustment of \( \gamma_1 \). The desired statement also holds true if \( g \) was set to \( \gamma_1 \) upon the arrival of \( J_n \) because in this case none of the algorithms \( \{A_k\}_{k \in K} \) using \( \gamma_1 \) has failed at the end of \( \sigma \).

In the following let \( A_{k^*} \in \{A_k\}_{k \in K} \) be an algorithm that using \( \gamma_1 \) has not failed and let \( S_{1k^*} \) be the associated schedule.

(2) We verify that the load of every machine in \( S_{1k^*} \) is upper bounded by \( (\rho + \varepsilon)\text{OPT}(\sigma) \). Let \( t_0 = 1 \). If the variable \( g \) was updated during the processing of \( \sigma \), then let \( t_1, \ldots, t_s \) be these points in time, i.e. the arrival of \( J_{t_i} \) caused an update of \( g \) and the variable was set to \( \gamma(i), 1 \leq i \leq s \). For any machine \( M_j, 1 \leq j \leq m, \) in \( S_{1k^*} \) let \( \ell(j) \) denote its final load at the end of \( \sigma \). Moreover, let \( \ell_{t_i}(j) \) denote its load due to jobs \( J_t \) with \( t \geq t_i \), for \( i = 0, \ldots, s \). Obviously
\[ \ell(j) = \ell_{t_s}(j) + \sum_{i=0}^{s-1} \left( \ell_{t_i}(j) - \ell_{t_{i+1}}(j) \right). \]  

(1)

We first show that \( \ell_{t_s}(j) \leq \rho \gamma_1 \). Immediately after \( J_{t_s} \) has been scheduled, \( M_j \)'s load consisting of jobs \( J_t \) with \( t' \geq t_s \) is at most \( p_{t_s} \). Since \( g \) was set to \( \gamma(s) = \gamma_1 \) on the arrival of \( J_{t_s} \), the guess adjustment rule ensures \( p_{t_s} \leq \gamma_1 \). Until the end of \( \sigma \) algorithm \( A_{k^*} \) using \( \gamma_1 \) does not fail and hence condition (ii) specifying the failure of algorithms implies that the assignment of each further job does not create a machine load greater than \( \rho \gamma_1 \) in \( S_{1k^*} \).

We next show \( \ell_{t_i}(j) - \ell_{t_{i+1}}(j) \leq \max(\rho, 2) \gamma(i) \), for each \( i = 0, \ldots, s - 1 \). The latter difference is the load on machine \( M_j \) caused by jobs of the subsequence \( J_{t_i}, \ldots, J_{t_{i+1}-1} \). Hence it suffices to show that after the assignment of any \( J_t \), with \( t_i \leq t < t_{i+1} \), \( M_j \)'s load due to jobs \( J_{t'} \), with \( t' \geq t_i \), is at most \( \max(\rho, 2) \gamma(i) \). After the assignment of \( J_{t_i} \), \( M_j \)'s respective load \( \ell_{t_i}(j) \) is at most \( p_{t_i} \) and this value is upper bounded by \( \gamma(i) \) as ensured by the guess adjustment rule. At times \( t > t_i \), while \( A_{k^*} \) using \( \gamma(i) \) has not failed, \( M_j \)'s load due to jobs \( J_{t'} \) with \( t' \geq t_i \) does not exceed \( \rho \gamma(i) \) as ensured by condition (ii) specifying the failure of algorithms. Finally consider a time \( t \), \( t_i < t < t_{i+1} \), at which \( A_{k^*} \) fails or has failed. The incoming job \( J_t \) is assigned to a least loaded machine. Hence if \( J_t \) is placed on \( M_j \), then the resulting machine load due to jobs \( J_{t'} \) with \( t' \geq t_i \) is upper bounded by \( \sum_{t_i \leq t' < t} p_{t'}/m + p_t \leq \sum_{1 \leq t' \leq t} p_{t'}/m + p_t \). Observe that after the arrival of \( J_t \) there exists an algorithm \( A_k \in \mathcal{A} \) that using \( \gamma(i) \) has not yet failed, since otherwise \( \gamma(i) \) would be adjusted before time \( t_{i+1} \). Condition (iii) defining the failure of algorithms ensures that \( \sum_{1 \leq t' \leq t} p_{t'}/m \leq \gamma(i) \) and \( p_t \leq \gamma(i) \). We obtain that \( M_j \)'s machine load is at most \( 2\gamma(i) \).

We conclude that \( \ell(j) \), given in (1), is upper bounded by

\[ \ell(j) \leq \rho \gamma_1 + \sum_{i=0}^{s-1} \max(\rho, 2) \gamma(i). \]  

(2)

By Lemma 2, \( \gamma_1 = \gamma(s) \leq (1 + \varepsilon/(3\rho)) \text{OPT}(\sigma) \). At the end of the description of \( \mathcal{A}^*(\varepsilon, h) \) we observed that whenever a guess is adjusted it increases by a factor of at least \( (1 + \varepsilon)^h \). Hence \( \gamma(i) \geq (1 + \varepsilon/(3\rho))^h \gamma(i - 1) \). It follows that \( \gamma(i) \leq (1 + \varepsilon/(3\rho))^{i(s-i)+\gamma(i)} \), for every \( 0 \leq i \leq s \). Thus \( \ell(j) \), as estimated in (2), is upper bounded by

\[ \ell(j) \leq \rho \left( 1 + \frac{\varepsilon}{3\rho} \right) \text{OPT}(\sigma) + \sum_{i=0}^{s-1} \max(\rho, 2) \gamma(s) \frac{2}{(1 + \varepsilon/(3\rho))^{h(s-i)}} \]  

\[ \leq \rho \left( 1 + \frac{\varepsilon}{3\rho} \right) \text{OPT}(\sigma) + \rho \left( 1 + \frac{\varepsilon}{3\rho} \right) \text{OPT}(\sigma) \sum_{i=0}^{s-1} \frac{2}{(1 + \varepsilon/(3\rho))^{h(s-i)}} \]  

\[ \leq \rho \left( 1 + \frac{\varepsilon}{3\rho} \right) \text{OPT}(\sigma) \left( 1 + \sum_{i=1}^{\infty} \frac{2}{(1 + \varepsilon/(3\rho))^{h(i)}} \right) \]  

(3)
\[
\begin{align*}
&= \rho \left( 1 + \frac{\varepsilon}{3\rho} \right) \text{OPT}(\sigma) \left( 1 + \frac{2}{(1 + \varepsilon/(3\rho))^h - 1} \right) \quad (4) \\
&\leq \rho \left( 1 + \frac{\varepsilon}{3\rho} \right)^2 \text{OPT}(\sigma) \leq \rho \left( 1 + \frac{\varepsilon}{\rho} \right) \text{OPT}(\sigma) = (\rho + \varepsilon)\text{OPT}(\sigma). \quad (5)
\end{align*}
\]

Here (3) uses the fact that \( \max\{\rho, 2\} \leq 2\rho \) and, as mentioned above, is a consequence of Lemma 2. Line (4) follows from the geometric series and, finally, (5) is by the choice of \( h \) and the assumption \( 0 < \varepsilon \leq 1 \).

3 A \((1 + \varepsilon)\)-Competitive Algorithm for MPS\textsubscript{opt}

We present an algorithm \( A_1(\varepsilon) \) for MPS\textsubscript{opt} that attains a competitive ratio of \( 1 + \varepsilon \), for any \( \varepsilon > 0 \). The number of parallel schedules will be \( \lfloor (2m/\varepsilon) + 1 \rfloor^{\lfloor \log(2/\varepsilon) / \log(1+\varepsilon/2) \rfloor} \). The algorithm will yield a \((1 + \varepsilon)\)-competitive strategy for MPS and, furthermore, will be useful in the next section where we develop a \((4/3 + \varepsilon)\)-competitive algorithm for MPS\textsubscript{opt}. There \( A_1(\varepsilon) \) will be used as subroutine for a small, constant number of \( m \).

Description of \( A_1(\varepsilon) \): Let \( \varepsilon > 0 \) be arbitrary. Recall that in MPS\textsubscript{opt} the optimum makespan \( \text{OPT}(\sigma) \) for the incoming job sequence \( \sigma \) is initially known. Assume without loss of generality that \( \text{OPT}(\sigma) = 1 \). Then all job processing times are in \((0, 1]\). Set \( \varepsilon' = \varepsilon/2 \). First we partition the range of possible job processing times into intervals \( I_0, \ldots, I_l \) such, within each interval \( I_i \) with \( i \geq 1 \), the values differ by a factor of at most \( 1 + \varepsilon' \). Such a partitioning is standard and has been used e.g. in the PTAS for offline makespan mininization devised by Hochbaum and Shmoys [27]. Let \( l = \lfloor \log(1/\varepsilon') / \log(1 + \varepsilon') \rfloor \). Set \( I_0 = (0, \varepsilon'] \) and \( I_i = ((1 + \varepsilon')^{i-1}\varepsilon', (1 + \varepsilon')^i\varepsilon'] \), for \( i = 1, \ldots, l \). Obviously \( I_0 \cup \cdots \cup I_l = (0, (1 + \varepsilon')^l\varepsilon'] \) and \((0, 1] \subseteq (0, (1 + \varepsilon')^l\varepsilon'] \).

A job is small if its processing time is at most \( \varepsilon' \) and hence contained in \( I_0 \); otherwise the job is large.

Each job sequence \( \sigma \) with \( \text{OPT}(\sigma) = 1 \) contains at most \( \lfloor m/\varepsilon' \rfloor \) large jobs. For each possible distribution of large jobs over the processing time intervals \( I_1, \ldots, I_l \), algorithm \( A_1(\varepsilon) \) prepares one algorithm/schedule. Let \( V = \{(v_1, \ldots, v_l) \in \mathbb{N}_0^l \mid v_i \leq \lfloor m/\varepsilon' \rfloor \} \). There holds \( |V| = \lfloor (m/\varepsilon') + 1 \rfloor^l \). Let \( A_1(\varepsilon) = \{A_v\}_{v \in V} \). For any vector \( v = (v_1, \ldots, v_n) \in V \), algorithm \( A_v \) works as follows. It assumes that the incoming job sequence \( \sigma \) contains exactly \( v_i \) jobs with a processing time in \( I_i \), for \( i = 1, \ldots, l \). Moreover, it pessimistically assumes that each processing time in \( I_i \) takes the largest possible value \((1 + \varepsilon')^i\varepsilon' \). Hence, initially \( A_v \) computes an optimal schedule \( S_v^* \) for a job sequence consisting of \( v_i \) jobs with a processing time of \((1 + \varepsilon')^i\varepsilon' \), for \( i = 1, \ldots, l \). Small jobs are ignored. Since running time is not an issue in the design of online algorithms, such a schedule \( S_v^* \) can be computed exactly. Alternatively, an \((1 + \varepsilon')\)-approximation to the optimal schedule can be computed using the PTAS by Hochbaum and Shmoys [27]. Let \( n_i^v(j) \) denote the number of jobs with a processing time of \((1 + \varepsilon')^i\varepsilon' \in I_i \) assigned to machine \( M_j \) in \( S_v^* \), where \( 1 \leq i \leq l \) and \( 1 \leq j \leq m \). Moreover, let \( \ell^v(j) = \sum_{i=1}^l n_i^v(j)(1 + \varepsilon')^i\varepsilon' \) be the load on machine \( M_j \) in \( S_v^* \), \( 1 \leq j \leq m \).
Algorithm $A_1(\varepsilon)$

1. $A_1(\varepsilon) = \{A_v\}_{v \in V}$, where $V = \{(v_1, \ldots, v_l) \in N_0^l : v_i \leq \lfloor m/\varepsilon' \rfloor\}$ with $\varepsilon' = \varepsilon/2$ and $l = \lceil \log(1/\varepsilon')/\log(1 + \varepsilon') \rceil$.

2. $A_v$ works as follows.
   (a) Compute optimal schedule $S_v^*$ for input consisting of $v_i$ jobs of processing time $(1 + \varepsilon')/\varepsilon'$, $1 \leq i \leq l$.
   (b) In $S_v$, each $J_i$ is sequenced in the following way.
       - If $p_t > \varepsilon'$, then determine $I_i$ such that $p_t \in I_i$. If $\exists M_j$ with $n^*_j(j) - n_i(j) > 0$, then assign $J_i$ to it; otherwise assign $J_i$ to an arbitrary machine.
       - If $p_t \leq \varepsilon'$, then assign $J_i$ to $M_j$ with the smallest value $\ell^*(j) + \ell_i(j)$.

Fig. 2 The algorithm $A_1(\varepsilon)$

When processing the actual job sequence $\sigma$ and constructing a real schedule $S_v$, $A_v$ uses $S_v^*$ as a guideline to make scheduling decisions. At any time during the scheduling process, let $n_i(j)$ be the number of jobs with a processing time in $I_i$ that have already been assigned to machine $M_j$ in $S_v$, where again $1 \leq i \leq l$ and $1 \leq j \leq m$. Each incoming job $J_i$, $1 \leq i \leq n$, is handled as follows. If $J_i$ is large, then let $I_i$ with $1 \leq i \leq l$ be the interval such that $p_t \in I_i$. Algorithm $A_v$ checks if there is a machine $M_j$ such that $n^*_j(j) - n_i(j) > 0$, i.e. there is a machine that can still accept a job with a processing time in $I_i$ as suggested by the optimal schedule $S_v^*$. If such a machine $M_j$ exists, then $J_i$ is assigned to it; otherwise $J_i$ is scheduled on an arbitrary machine.

If $J_i$ is small, then $J_i$ is assigned to a machine $M_j$ with the smallest current value $\ell^*(j) + \ell_i(j)$. Here $\ell_i(j)$ denotes the current load on machine $M_j$ caused by small jobs in $S_v$. A summary of $A_1(\varepsilon)$ is given in Fig. 2.

**Theorem 2** For any $\varepsilon > 0$, $A_1(\varepsilon)$ is $(1 + \varepsilon)$-competitive and uses at most $(\lfloor 2m/\varepsilon \rfloor + 1)^{\lceil \log(2/\varepsilon)/\log(1 + \varepsilon) \rceil}$ schedules.

**Proof** The bound on the number of schedules simply follows from the fact that $A_1(\varepsilon)$ maintains $|V| = (\lfloor m/\varepsilon' \rfloor + 1)^l$ schedules where again $\varepsilon' = \varepsilon/2$ and $l = \lceil \log(1/\varepsilon')/\log(1 + \varepsilon') \rceil$.

Let $\sigma$ be an arbitrary job sequence and let $v_i$ be the number of jobs with a processing time in $I_i$, for $i = 1, \ldots, l$. Since any $v_i$ is upper bounded by $\lfloor m/\varepsilon' \rfloor$, the resulting vector $v = (v_1, \ldots, v_l)$ is in $V$. For this vector $v$, consider the associated algorithm $A_v$. We prove that when $A_v$ has finished processing $\sigma$, the resulting schedule $S_v$ has a makespan of at most $(1 + \varepsilon) = (1 + \varepsilon)\text{OPT}(\sigma)$. Recall again that we assume without loss of generality that $\text{OPT}(\sigma) = 1$.

We analyze the steps in which $A_v$ assigns jobs $J_i$, $1 \leq i \leq n$, to machines in $S_v$. If $J_i$ is large with $p_t \in I_i$, $1 \leq i \leq l$, then there must exist a machine $M_j$ in the current schedule $S_v$ such that $n^*_j(j) - n_i(j) > 0$. Algorithm $A_v$ will assign $J_i$ to such a machine. Hence after the processing of $\sigma$, for any $M_j$ in $S_v$, the total load caused by large jobs is upper bounded by $\ell^*(j)$. We next argue that this value is at most $(1 + \varepsilon')\text{OPT}(\sigma)$. Consider an optimal schedule $S_{\text{opt}}$ for $\sigma$. Modify this schedule by (a) deleting all small jobs and (b) rounding each job processing time in $I_i$ to $(1 + \varepsilon')\varepsilon'$, for $i = 1, \ldots, l$. The resulting schedule $S'_{\text{opt}}$ has a makespan of at most $(1 + \varepsilon')\text{OPT}(\sigma)$. Furthermore $S'_{\text{opt}}$ is a schedule for an input sequence consisting of $v_i$ jobs of processing
time \((1+\varepsilon')^j\varepsilon'\). Since \(S^*_v\) is an optimal schedule for this input, each machine load \(\ell^*(j)\) is upper bounded by \((1+\varepsilon')\text{OPT}(\sigma)\).

We finally show that when \(A_v\) has to sequence a small job \(J_t\), then there is a machine \(M_j\) such that \(\ell^*(j) + \ell_s(j)\) is upper bounded by \((1+\varepsilon')\text{OPT}(\sigma)\). This implies that the assignment of \(J_t\) causes a machine load of at most \((1+\varepsilon')\text{OPT}(\sigma) + p_t \leq (1+2\varepsilon')\text{OPT}(\sigma) = (1+\varepsilon)\text{OPT}(\sigma)\) in the final schedule \(S_v\).

So suppose that upon the arrival of a small job \(J_t\) there holds \(\ell^*(j) + \ell_s(j) > (1+\varepsilon')\text{OPT}(\sigma)\) for all machines \(M_j\), \(1 \leq j \leq m\). Recall that \(\ell_s(j)\) is the load on machine \(M_j\) caused by small jobs in the current schedule \(S_v\). Note that \(\sum_{j=1}^m \ell^*(j)\) is the total processing time of large jobs in \(\sigma\) if processing times in \(I_i\) are rounded up to \((1+\varepsilon')^j\varepsilon'\), for \(i = 1, \ldots, l\). Hence \(1/(1+\varepsilon') \sum_{j=1}^m \ell^*(j)\) is a lower bound on the total processing time of large jobs in \(\sigma\). It follows that the total processing time of all jobs in \(\sigma\) is at least \(1/(1+\varepsilon') \sum_{j=1}^m \ell^*(j) + \sum_{j=1}^m \ell_s(j) + p_t \geq 1/(1+\varepsilon') \sum_{j=1}^m (\ell^*(j) + \ell_s(j)) + p_t\). The assumption that \(\ell^*(j) + \ell_s(j) > (1+\varepsilon')\text{OPT}(\sigma)\) holds for all machines \(M_j\) implies that the total processing time of jobs in \(\sigma\) is at least \(m \cdot \text{OPT}(\sigma) + p_t \geq m \cdot \text{OPT}(\sigma)\), which contradicts the fact that \(\text{OPT}(\sigma)\) is the optimum makespan. \( \square \)

4 A \((4/3 + \varepsilon)\)-Competitive Algorithm for MPS_{opt}

We present an algorithm for MPS_{opt} that is \((4/3 + \varepsilon)\)-competitive, for any \(0 < \varepsilon \leq 1\). The number of required schedules is \(1/\varepsilon O(\log(1/\varepsilon))\), which is a constant independent of \(n\) and \(m\).

4.1 Description of the Algorithm

We develop an algorithm \(A_2(\varepsilon)\) that is \((4/3 + \varepsilon)\)-competitive, for any \(0 < \varepsilon \leq 1\), if the number \(m\) of machines is not too small. We then combine \(A_2(\varepsilon)\) with \(A_1(\varepsilon)\), presented in the last section, and derive a strategy \(A_3(\varepsilon)\) that is \((4/3 + \varepsilon)\)-competitive, for arbitrary \(m\).

Before describing \(A_2(\varepsilon)\) in detail, we explain the main ideas of the algorithm. One concept is identical to that used by \(A_1(\varepsilon)\): Partition the range of possible job processing times into intervals or job classes and consider distributions of jobs over these classes. However, in order to achieve a constant number of schedules we have to refine this scheme and incorporate new ideas. First, the job classes have to be chosen properly so as to allow a compact packing of jobs on the machines. Loosely speaking a job will be small if its processing time is at most \(1/3\); otherwise the job will be large. The processing times greater than \(1/3\) are divided into a constant number of job classes, where this constant depends on \(\log(1/\varepsilon)\). In the schedules constructed by the algorithm most of the machines will be devoted to a specific class of large jobs. A machine will either contain one job of processing time greater than \(2/3\) or two jobs of size between \(1/3\) and \(2/3\). An important, new aspect in the construction of \(A_2(\varepsilon)\) is that we will not consider the entire set \(V\) of tuples specifying how large jobs of an input sequence \(\sigma\) are distributed over the job classes. Given that most of the machines are devoted job classes, this would lead to a polynomial number of schedules. In order to
achieve a constant number of schedules we will instead define a suitable sparsification $V'$ of $V$. Each $v \in V'$ represents an estimate or guess on the number of large jobs arising in $\sigma$. More specifically, if $v = (v_1, \ldots, v_l)$, then it is assumed that $\sigma$ contains at least $v_i$ jobs with a processing time of job class $i$.

Obviously, the job sequence $\sigma$ may contain more large jobs, the exact number of which is unknown. Furthermore, it is unknown which portion of the total processing time of $\sigma$ will arrive as small jobs. In order to cope with these uncertainties $A_2(\varepsilon)$ has to construct robust schedules. To this end the number of machines is partitioned into two sets $M_c$ and $M_r$. Roughly $(1 - \delta)m$ machines will belong to $M_c$, for a suitably chosen $\delta$. For the machines of $M_c$, the algorithm initially determines a good assignment or configuration assuming that $v_i$ jobs of job class $i$ will arrive. As mentioned before, each such machine is devoted to a particular class of large jobs. It will contain one job of processing time greater than $2/3$ or two jobs of size between $1/3$ and $2/3$. The machines of $M_r$ are reserve machines and will be assigned additional large jobs as they arise in $\sigma$. The scheduling policy is simple. Whenever a large job arrives, assign it to a machine in $M_c$ devoted to that respective job class, provided such a machine with remaining capacity exists. Otherwise assign it to a machine in $M_r$ by applying the Best-Fit strategy. Small jobs will always be placed on machines in $M_c$. Preference is given to machines that already contain small jobs. The initial configuration determined for these machines has the property that, no matter how many small jobs arrive, a machine load never exceeds $4/3 + \varepsilon$ times the optimum makespan.

We proceed to describe $A_2(\varepsilon)$ in detail. Let $0 < \varepsilon \leq 1$. Moreover, set $\varepsilon' = \varepsilon/8$. Again we assume without loss of generality that, for an incoming job sequence, there holds $\text{OPT}(\sigma) = 1$. Hence the processing time of any job is upper bounded by 1.

**Job classes:** A job $J_t$, $1 \leq t \leq n$, is small if $p_t \leq 1/3 + 2\varepsilon'$; otherwise $J_t$ is large. We divide the range of possible job processing times into job classes. Let $I_s = (0, 1/3 + 2\varepsilon']$ be the interval containing the processing times of small jobs. Let $\lambda = \lfloor \log_3 \frac{3}{8} + \frac{1}{48\varepsilon} \rfloor$ and $l = \lambda + 2$, where the logarithm is taken to base 2. For $i = 1, \ldots, l$, let

$$a_i = \max \left\{ \frac{1}{3} - 2\varepsilon' + \left( \frac{1}{12} + \frac{3}{2} \varepsilon' \right) \frac{1}{2^{\lambda+1-i}}, \frac{1}{2} + 2\varepsilon' \right\} \quad \text{and}$$

$$b_i = \frac{1}{3} - 2\varepsilon' + \left( \frac{1}{12} + \frac{3}{2} \varepsilon' \right) \frac{1}{2^{\lambda-i}}.$$  

It is easy to verify that $a_1 = 1/3 + 2\varepsilon'$ and $a_i < b_i$, for $i = 1, \ldots, l$. Furthermore $b_{l-1} = 1/2 + \varepsilon'$ and $b_l = 2/3 + 4\varepsilon'$. For $i = 1, \ldots, l$ define $I_i = (a_i, b_i]$. There holds $\bigcup_{1 \leq i \leq l} I_i = (1/3 + 2\varepsilon', 2/3 + 4\varepsilon']$. Moreover, for $i = 1, \ldots, l - 1$, let $I_{l+i} = (2a_i, 2b_i]$. Intuitively, $I_{l+i}$ contains the processing times that are twice as large as those in $I_i$, $1 \leq i \leq l - 1$. There holds $\bigcup_{1 \leq i \leq l-1} I_{l+i} = (2/3 + 4\varepsilon', 1 + 2\varepsilon']$. Hence $I_s \cup I_1 \cup \ldots \cup I_{2l-1} = (0, 1 + 2\varepsilon']$. In the following $I_i$ represents job class $i$, for $i = 1, \ldots, 2l - 1$. We say that a large job $J_t$ is a class-$i$ job if $p_t \in I_i$, where $1 \leq i \leq 2l - 1$.  

© Springer
Definition of target configurations: As mentioned above, for any incoming job sequence \( \sigma \), \( A_2(\varepsilon) \) works with estimates on the number of class-\( i \) jobs arising in \( \sigma \), \( 1 \leq i \leq 2l - 1 \). For each estimate, the algorithm initially determines a virtual schedule or target configuration on a subset of the machines, assuming that the estimated set of large jobs will indeed arrive. Hence we partition the \( m \) machines into two sets \( M_c \) and \( M_r \). Let \( \mu = [(1 + \varepsilon')/(1 + 2\varepsilon') \cdot m] \). Moreover, let \( M_c = \{ M_1, \ldots, M_{\mu} \} \) and \( M_r = \{ M_{\mu+1}, \ldots, M_m \} \). Set \( M_c \) contains the machines for which a target configuration will be computed; \( M_r \) contains the reserve machines. The proportion of \(|M_r|\) to \(|M_c|\) is roughly \( 1:1 + 1/\varepsilon' \).

A target configuration has the important property that any machine \( M_j \in M_c \) contains large jobs of only one job class \( i \), \( 1 \leq i \leq 2l - 1 \). Therefore, a target configuration is properly defined by a vector \( c = (c_1, \ldots, c_{\mu}) \in \{0, \ldots, 2\varepsilon\}^{\mu} \). If \( c_j = 0 \), then \( M_j \) does not contain any large jobs in the target configuration, \( 1 \leq j \leq \mu \). If \( c_j = i \), where \( i \in \{1, \ldots, 2l - 1\} \), then \( M_j \) contains class-\( i \) jobs, \( 1 \leq j \leq \mu \). The vector \( c \) implicitly also specifies how many large jobs reside on a machine. If \( c_j = i \) with \( 1 \leq i \leq l \), then \( M_j \) contains two class-\( i \) jobs. Note that, for general \( i \in \{1, \ldots, l\} \), a third job cannot be placed on the machine without exceeding a load bound of \( 4/3 + \varepsilon \). If \( c_j = i \) with \( l + 1 \leq i \leq 2l - 1 \), then \( M_j \) contains one class-\( i \) job. Again, the assignment of a second job is not feasible in general. Given a configuration \( c \), \( M_j \) is referred to as a class-\( i \) machine if \( c_j = i \), where \( 1 \leq j \leq \mu \) and \( 1 \leq i \leq 2l - 1 \).

With the above interpretation of target configurations, each vector \( c = (c_1, \ldots, c_{\mu}) \) encodes inputs containing \( 2|\{c_j \in \{c_1, \ldots, c_{\mu}\} : c_j = i\}| \) class-\( i \) jobs, for \( i = 1, \ldots, l \), as well as \(|\{c_j \in \{c_1, \ldots, c_{\mu}\} : c_j = i\}| \) class-\( i \) jobs, for \( i = l+1, \ldots, 2l - 1 \). Hence, for an incoming job sequence, instead of considering estimates on the number of class-\( i \) jobs, for any \( 1 \leq i \leq 2l - 1 \), we can equivalently consider target configurations. Unfortunately, it will not be possible to work with all target configurations \( c \in \{0, \ldots, 2\varepsilon\}^{\mu} \) since the resulting number of schedules to be constructed would be \( (2l)^{\mu} = (\log(1/\varepsilon))^{\Omega(m)} \). Therefore, we will work with a suitable sparsification of the set of all configurations.

Sparsification of the set of target configurations: Let \( \kappa = \lceil 2(2 + 1/\varepsilon')(2l - 1) \rceil \) and \( U = \{0, \ldots, \kappa\}^{2l-1} \). We will show that \( \kappa \lceil (m - \mu)/(2l - 1) \rceil \geq m \) if \( m \) is not too small (see Lemma 4). This property in turn will ensure that any job sequence \( \sigma \) can be mapped to a \( u \in U \). For any vector \( u = (u_1, \ldots, u_{2l-1}) \in U \), we define a target configuration \( c(u) \) that contains \( u_i \lceil (m - \mu)/(2l - 1) \rceil \) class-\( i \) machines, for \( i = 1, \ldots, 2l - 1 \), provided that \( \sum_{i=1}^{2l-1} u_i \lceil (m - \mu)/(2l - 1) \rceil \) does not exceed \( \mu \). More specifically, for any \( u = (u_1, \ldots, u_{2l-1}) \in U \), let \( \pi_0 = 0 \) and \( \pi_i = \sum_{j=1}^{i-1} u_j \lceil (m - \mu)/(2l - 1) \rceil \), be the partial sums of the first \( i \) entries of \( u \), multiplied by \( [(m - \mu)/(2l - 1)] \), for \( i = 1, \ldots, 2l - 1 \). Let \( \mu' = \pi_{2l-1} \). First construct a vector \( c'(u) = (c'_1, \ldots, c'_{\mu'}) \) of length \( \mu' \) that contains exactly \( u_i \lceil (m - \mu)/(2l - 1) \rceil \) class-\( i \) machines. That is, for \( i = 1, \ldots, 2l - 1 \), let \( c'_j = i \) for \( j = \pi_{i-1} + 1, \ldots, \pi_i \). We now truncate or extend \( c'(u) \) to obtain a vector of length \( \mu \). If \( \mu' \geq \mu \), then \( c(u) \) is the vector consisting of the first \( \mu \) entries of \( c'(u) \). If \( \mu' < \mu \), then \( c(u) = (c'_1, \ldots, c'_{\mu'}, 0, \ldots, 0) \), i.e. the last \( \mu - \mu' \) entries are set to 0. Let \( C = \{c(u) \mid u \in U\} \) be the set of all target configurations constructed from vectors \( u \in U \).
The algorithm family: Let $A_{2}(\varepsilon) = \{A_{c}\}_{c \in C}$. For any $c \in C$, algorithm $A_{c}$ works as follows. Initially, prior to the arrival of any job of $\sigma$, $A_{c}$ determines the target configuration specified by $c = (c_{1}, \ldots, c_{L})$ and uses this virtual schedule for the machines of $\mathcal{M}_{c}$ to make scheduling decisions. Consider a machine $M_{j} \in \mathcal{M}_{c}$ and suppose $c_{j} > 0$, i.e. $M_{j}$ is a class-$i$ machine for some $i \geq 1$. Let $\ell^{-}(j)$ and $\ell^{+}(j)$ be the targeted minimal and maximal loads caused by large jobs on $M_{j}$, according to the target configuration. More precisely, if $i \in \{1, \ldots, l\}$, then $\ell^{-}(j) = 2a_{i}$ and $\ell^{+}(j) = 2b_{i}$. Recall that in a target configuration a class-$i$ machine contains two class-$i$ jobs if $1 \leq i \leq l$. If $i \in \{l + 1, \ldots, 2l - 1\}$ and hence $i = i' + i''$ for some $i' \in \{1, \ldots, l - 1\}$, then $\ell^{-}(j) = 2a_{i'}$ and $\ell^{+}(j) = 2b_{i'}$. If $M_{j} \in \mathcal{M}_{c}$ is a machine with $c_{j} = 0$, then $\ell^{-}(j) = \ell^{+}(j) = 0$. While the job sequence $\sigma$ is processed, a machine $M_{j} \in \mathcal{M}_{c}$ may or may not be admissible. Again assume that $M_{j}$ is a class-$i$ machine with $i \geq 1$. If $i \in \{1, \ldots, l\}$, then at any time during the scheduling process $M_{j}$ is admissible if it has received less than two class-$i$ jobs so far. Analogously, if $i \in \{l + 1, \ldots, 2l - 1\}$, then $M_{j}$ is admissible if it has received no class-$i$ job so far. Finally, at any time during the scheduling process, let $\ell(j)$ be the current load of machine $M_{j}$ and let $\ell_{s}(j)$ be the load due to small jobs, $1 \leq j \leq m$.

Algorithm $A_{2}$ schedules each incoming job $J_{t}$, $1 \leq t \leq n$, in the following way. First assume that $J_{t}$ is a large job and, in particular, a class-$i$ job, $1 \leq i \leq 2l - 1$. The algorithm checks if there is a class-$i$ machine in $\mathcal{M}_{c}$ that is admissible. If so, $J_{t}$ is assigned to such a machine. If there is no admissible class-$i$ machine available, then $J_{t}$ is placed on a machine in $\mathcal{M}_{r}$. There jobs are scheduled according to the Best-Fit policy. More specifically, $A_{c}$ checks if there exists a machine $M_{j} \in \mathcal{M}_{r}$ such that $\ell(j) + p_{t} \leq 4/3 + \varepsilon$. If this is the case, then $J_{t}$ is assigned to such a machine with the largest current load $\ell(j)$.

A summary of $A_{2}(\varepsilon)$, which focuses on the job assignment rules, is given in Fig. 3.

Theorem 3 $A_{2}(\varepsilon)$ is $(4/3 + \varepsilon)$-competitive, for any $0 < \varepsilon \leq 1$ and $m \geq 2l/(\varepsilon')^{2}$ where $\varepsilon' = \varepsilon/8$. The algorithm uses $1/\varepsilon^{O(\log(1/\varepsilon))}$ schedules.

$A_{2}(\varepsilon)$ is $(4/3 + \varepsilon)$-competitive if, for the chosen $\varepsilon$, the number of machines is at least $2l/(\varepsilon')^{2}$. If the number of machines is smaller, we can simply apply algorithm $A_{1}(\varepsilon)$ with an accuracy of $\varepsilon_{0} = 1/3$. Let $A_{3}(\varepsilon)$ be the following combined algorithm. If for the chosen $\varepsilon$, $m < 2l/(\varepsilon')^{2}$, execute $A_{1}(1/3)$. Otherwise execute $A_{2}(\varepsilon)$.

Corollary 1 $A_{3}(\varepsilon)$ is $(4/3 + \varepsilon)$-competitive, for any $0 < \varepsilon \leq 1$, and uses $1/\varepsilon^{O(\log(1/\varepsilon))}$ schedules.

Proof If $A_{1}(1/3)$ is executed for a machine number $m < 2l/(\varepsilon')^{2}$, then by Theorem 2 the number of schedules is $(\log(1/\varepsilon))/\varepsilon^{3}O(1)$, which is $1/\varepsilon^{O(1)}$.
Algorithm $A_2(\varepsilon)$

1. $A_2(\varepsilon) = \{A_c\}_{c \in C}$, where $C = \{c(u) \mid u \in U\}$
   
   $U = \{0, \ldots, 2\varepsilon\}$, where $\varepsilon = [2(2 + 1/2\varepsilon)(2l - 1)]$, $l = [\log(\sqrt{\varepsilon} + 1/4\varepsilon^2)] + 2$ and $\varepsilon' = \varepsilon/8$
   
   $\mu = \lceil (1 + \varepsilon')/(1 + 2\varepsilon') \cdot m \rceil$

2. $A_c$ works as follows.
   
   (a) Determine target configuration specified by $c = (c_1, \ldots, c_\mu)$.

   (b) Each $J_i$ is sequenced as follows.

   $J_i$ is large: Let $J_i$ be a class-$i$ job. If there is an admissible class-$i$ machine in $M_c$, assign $J_i$ to it. Otherwise check if $\exists M_j \in M_c$ such that $\ell(j) + p_t \leq 4/3 + \varepsilon$. If so, assign $J_i$ to such an $M_j$ with the highest $\ell(j)$; otherwise place $J_i$ on an arbitrary $M_j \in M_c$.

   $J_i$ is small: If $\exists M_j \in M_c$ with $\ell_j(i) > 0$ such that $\ell_j(i) + p_r \leq 4/3 + \varepsilon$, assign $J_i$ to it. Otherwise check if $\exists M_j \in M_c$ with $\ell_j(i) = 0$ such that $\ell_j(i) + p_r \leq 4/3 + \varepsilon$. If so, assign $J_i$ to such an $M_j$ with the lowest $\ell_j(i)$; otherwise place $J_i$ on an arbitrary $M_j \in M_c$.

Fig. 3 The algorithm $A_2(\varepsilon)$

4.2 Analysis of the Algorithm

We prove Theorem 3. The stated number of schedules follows from the fact that $A_2(\varepsilon)$ consists of $|C| = (\kappa + 1)^{2l-1}$ algorithms. Recall that $\kappa = [2(2 + 1/\varepsilon')(2l - 1)]$ and $l = \lambda + 2 = [\log(\sqrt{\varepsilon} + 1/4\varepsilon^2)] + 2$. Hence $l = O((\log(1/\varepsilon))$ and $\kappa = O(1/\varepsilon \log(1/\varepsilon))$, which gives that $|C|$ is $1/\varepsilon O((\log(1/\varepsilon))$.

In order to establish Theorem 3 we prove that, for any job sequence $\sigma$, $A_2(\varepsilon)$ generates a schedule whose makespan is at most $(4/3 + \varepsilon)OPT(\sigma)$. More specifically we first show that, for any $\sigma$, there exists a target configuration $c \in C$ that accurately models the large jobs arising in $\sigma$. We will refer to such a vector as a valid target configuration. Then we show that the corresponding algorithm $A_c$ builds a schedule with a makespan of at most $(4/3 + \varepsilon)OPT(\sigma)$.

We introduce some notation. Consider any job sequence $\sigma$. For any $i, 1 \leq i \leq 2l - 1$, let $n_i(\sigma)$ be the number of class-$i$ jobs arising in $\sigma$, i.e. $n_i(\sigma)$ is the number of jobs $J_i$ with $p_r \in I_i$. Furthermore, for any target configuration $c = (c_1, \ldots, c_\mu) \in C$ and any $i$ with $1 \leq i \leq 2l - 1$, let $m_i$ be the number of class-$i$ machines in $c$, i.e., $m_i = \sum_{j=1}^{l} m_j$ be the total number of class-$i$ machines with $i \in \{1, \ldots, l\}$. Similarly, $\mu_2 = \sum_{i=l+1}^{2l-1} m_i$ is the total number of class-$i$ machines with $i \in \{l+1, \ldots, 2l-1\}$. Given $\sigma$, vector $c \in C$ will be a valid target configuration if, for any $i = 1, \ldots, 2l - 1$, $\sigma$ contains as many class-$i$ jobs as specified in $c$ and, moreover, if all the additional large jobs can be feasibly scheduled on the $m - \mu$ reserve machines. Recall that in a configuration $c$, any class-$i$ machine with $1 \leq i \leq l$ is supposed to contain two class-$i$ jobs. Formally, $c \in C$ is a valid target configuration if the following three conditions hold.

(i) For $i = 1, \ldots, l$, there holds $2m_i \leq n_i(\sigma)$.

(ii) For $i = l + 1, \ldots, 2l - 1$, there holds $m_i \leq n_i(\sigma)$.

(iii) $\lceil (\sum_{i=1}^{l} n_i(\sigma) - 2\mu_1)/2 \rceil + \sum_{i=l+1}^{2l-1} n_i(\sigma) - \mu_2 \leq m - \mu$

Conditions (i) and (ii) represent the constraint that $\sigma$ contains as many class-$i$ jobs as specified in $c$, $1 \leq i \leq 2l - 1$. Condition (iii) models the requirement that extra large jobs can be feasibly packed on the reserve machines. Here $\sum_{i=1}^{l} n_i(\sigma) - 2\mu_1$ is the extra number of class-$i$ jobs with $i \in \{1, \ldots, l\}$ in $\sigma$. Any two of these can be packed
on one machine since the processing time of any of these jobs is upper bounded by \( b_i \leq 2/3 + 4\varepsilon' \). Hence two jobs incur a machine load of at most \( 4/3 + 8\varepsilon' = 4/3 + \varepsilon \). Analogously, \( \sum_{i=l+1}^{2l-1} n_i(\sigma) - \mu_2 \) is the extra number of class-\( i \) jobs with \( i \in \{l + 1, \ldots, 2l - 1\} \), which cannot be combined together because their processing times are greater than \( 2a_1 \geq 2/3 + 4\varepsilon' \).

In order to prove that, for any \( \sigma \), there exists a valid target configuration we need two lemmas.

**Lemma 3** For any \( \sigma \), there holds \( \left\lfloor \sum_{i=1}^{\ell} n_i(\sigma)/2 \right\rfloor + \sum_{i=\ell+1}^{2\ell-1} n_i(\sigma) \leq m \).

**Proof** Consider any optimal schedule \( S^* \) for \( \sigma \) and recall that we assume without loss of generality that \( \text{OPT}(\sigma) = 1 \). In \( S^* \) any machine containing a class-\( i \) job with \( i \in \{l + 1, \ldots, 2l - 1\} \) cannot contain an additional large job: The class-\( i \) job causes a load greater than \( 2a_1 \geq 2/3 + 4\varepsilon' \) and any additional large job, having a processing time greater than \( 1/3 + 2\varepsilon' \), would generate a total load greater than 1. Furthermore, any machine containing a class-\( i \) job with \( i \in \{1, \ldots, \ell\} \) can contain at most one additional job of the job classes \( 1, \ldots, \ell \) because two further jobs would generate a total load greater than \( 3a_1 \geq 3(1/3 + 2\varepsilon') > 1 \). \( \square \)

**Lemma 4** For any \( 0 < \varepsilon' \leq 1/8 \), there holds \( \kappa \left( (m - \mu)/(2l - 1) \right) \geq m \) if \( m \geq 2l/\varepsilon'^2 \).

**Proof** There holds

\[
\left( (m - \mu)/(2l - 1) \right) \geq 2\left( (m - \mu)/(2l - 1) \right) \\
\geq 2((m - \mu)/(2l - 1) - 1) \\
= 2\left( m - \left\lfloor \frac{1 + \varepsilon'}{1 + 2\varepsilon'} m \right\rfloor \right) \\
\geq 2\left( \frac{\varepsilon' m - 1}{1 + 2\varepsilon'} - 1 \right) \\
= 2\left( \frac{\varepsilon' m - (1 + 2\varepsilon') 2l}{(1 + 2\varepsilon')(2l - 1)} \right).
\]

Hence

\[
\kappa \left( (m - \mu)/(2l - 1) \right) \geq 2(2 + \frac{1}{\varepsilon'}) (2l - 1) \cdot \left( \frac{\varepsilon' m - (1 + 2\varepsilon') 2l}{(1 + 2\varepsilon')(2l - 1)} \right) \\
\geq m + m - (2/\varepsilon')(1 + 2\varepsilon') 2l \\
\geq m,
\]

where the last line follows because of \( m \geq 2l/\varepsilon'^2 \) and \( 2l/\varepsilon'^2 \geq (2/\varepsilon')(1 + 2\varepsilon') 2l \), for any \( \varepsilon' \leq 1/8 \). \( \square \)

The next lemma establishes the existence of valid target configurations.

**Lemma 5** For any \( \sigma \), there exists a valid target configuration \( c \in C \) if \( m \geq 2l/\varepsilon'^2 \).
Proof In this proof let \( m_0 = [(m - \mu)/(2l - 1)] \). Given \( \sigma \), we first construct a vector \( u \in U \). Lemma 3 implies that for any job class \( i \), \( 1 \leq i \leq l \), there holds \( [n_i(\sigma)/2] \leq m \). For any job class \( i \), \( l + 1 \leq i \leq 2l - 1 \), there holds \( n_i(\sigma) \leq m \). By Lemma 4, \( \kappa m_0 \geq m \), which is equivalent to \( m/m_0 \leq \kappa \). For any \( i \) with \( 1 \leq i \leq l \), set \( u_i = [n_i(\sigma)/(2m_0)] \). For any \( i \) with \( l + 1 \leq i \leq 2l - 1 \), set \( u_i = [n_i(\sigma)/m_0] \). Then \( u_i \in \{0, \ldots, \kappa\} \), for \( i = 1, \ldots, 2l - 1 \), and the resulting vector \( u = (u_1, \ldots, u_{2l-1}) \) is element of \( U \). We next show that the vector \( c(u) \) constructed by \( A_2(\varepsilon) \) is a valid target configuration.

When \( A_2(\varepsilon) \) constructs \( c(u) \), it first builds a vector \( c'(u) = (c'_1, \ldots, c'_{\mu}) \) of length \( \mu' = \sum_{i=1}^{2l-1} u_im_0 \) containing exactly \( u_i m_0 \) entries with \( c'_j = i \), for \( i = 1, \ldots, 2l - 1 \). If \( \mu' \geq \mu \), then \( c(u) \) contains the first \( \mu \) entries of \( c'(u) \). If \( \mu' < \mu \), then \( c(u) \) is obtained from \( c'(u) \) by adding \( \mu - \mu' \) entries of value 0. In either case \( c(u) \) contains at most \( u_i m_0 \) entries of value \( i \), for \( i = 1, \ldots, 2l - 1 \). Hence for the target configuration \( c(u) \), there holds \( m_i \leq u_i m_0 \), for \( i = 1, \ldots, 2l - 1 \), where \( m_i \) is again the total number of class-\( i \) machines in \( c(u) \).

If \( i \in \{1, \ldots, l\} \), then \( m_i \leq [n_i(\sigma)/(2m_0)] m_0 \leq n_i(\sigma)/2 \), which is equivalent to \( 2m_i \leq n_i(\sigma) \). Similarly, if \( i \in \{l + 1, \ldots, 2l - 1\} \), then \( m_i \leq [n_i(\sigma)/m_0] m_0 \leq n_i(\sigma) \). Therefore, conditions (i) and (ii) defining valid target configurations are satisfied and we are left to verify condition (iii).

First assume \( \mu' \geq \mu \). In this case the vector \( c(u) \) contains no entries of value 0 and hence \( \mu = \mu_1 + \mu_2 \). Recall that \( \mu_1 = \sum_{i=1}^{l} m_i \) is the total number of class-\( i \) machines with \( i \in \{1, \ldots, l\} \) specified in \( c(u) \). Similarly, \( \mu_2 = \sum_{i=l+1}^{2l-1} m_i \) is the total number of class-\( i \) machines with \( i \in \{l+1, \ldots, 2l-1\} \). By Lemma 3, \( \lceil \sum_{i=1}^{l} n_i(\sigma)/2 \rceil + \sum_{i=l+1}^{2l-1} n_i(\sigma) \leq m \). Subtracting the equation \( \mu_1 + \mu_2 = \mu \), we obtain
\[
\left\lfloor \sum_{i=1}^{l} n_i(\sigma)/2 \right\rfloor - \mu_1 + \sum_{i=l+1}^{2l-1} n_i(\sigma) - \mu_2 \leq m - \mu.
\]
There holds \( \lceil \sum_{i=1}^{l} n_i(\sigma)/2 \rceil - \mu_1 = \lceil (\sum_{i=1}^{l} n_i(\sigma) - 2\mu_1)/2 \rceil \) because \( \mu_1 \) is an integer. Hence condition (iii) defining valid target configurations is satisfied.

It remains to study the case \( \mu' < \mu \). For any \( i \) with \( i \in \{l + 1, \ldots, 2l - 1\} \), there holds \( u_i = [n_i(\sigma)/m_0] \) and hence \( u_i > n_i(\sigma)/m_0 - 1 \), which is equivalent to \( n_i(\sigma) < (u_i + 1)m_0 \). Hence
\[
\sum_{i=l+1}^{2l-1} n_i(\sigma) < \sum_{i=l+1}^{2l-1} (u_i + 1)m_0 = \sum_{i=l+1}^{2l-1} u_im_0 + (l - 1)m_0.
\]
The sum \( \sum_{i=l+1}^{2l-1} u_im_0 = \sum_{i=l+1}^{2l-1} u_i[(m - \mu)/(2l - 1)] \) is the total number of entries \( c'_j \) with \( c'_j \in \{l + 1, \ldots, 2l - 1\} \) in \( c'(u) \). Since \( \mu' < \mu \), none of these entries is deleted when \( c(u) \) is derived from \( c'(u) \). Hence \( \sum_{i=l+1}^{2l-1} u_im_0 = \mu_2 \) is the total number of class-\( i \) machines with \( i \in \{l + 1, \ldots, 2l - 1\} \) specified in \( c(u) \). We conclude
\[
\sum_{i=l+1}^{2l-1} n_i(\sigma) \leq \mu_2 + (l - 1)m_0. \tag{6}
\]
For any $i$ with $i \in \{1, \ldots, l\}$, there holds $u_i = \lfloor n_i(\sigma)/(2m_0) \rfloor$ and hence $u_i > n_i(\sigma)/(2m_0) - 1$. This implies $n_i(\sigma)/2 < (u_i + 1)m_0$. Since $(u_i + 1)m_0$ is an integer we obtain $n_i(\sigma)/2 \leq (u_i + 1)m_0 - 1$. Thus

$$\sum_{i=1}^{l} n_i(\sigma)/2 \leq \sum_{i=1}^{l} n_i(\sigma)/2 + 1 \leq \sum_{i=1}^{l} (u_i + 1)m_0 = \mu_1 + lm_0. \quad (7)$$

Again $\sum_{i=1}^{l} u_i m_0 = \mu_1$ because $c'(u)$ contains exactly $\sum_{i=1}^{l} u_i m_0$ entries $c'_j$ with $c'_j \in \{1, \ldots, l\}$ and all of these entries are contained in $c(u)$ representing class-$i$ machines for $i \in \{1, \ldots, l\}$. Inequalities (6) and (7) together with the identity $m_0 = [(m - \mu)/(2l - 1)]$ imply

$$\sum_{i=1}^{l} n_i(\sigma)/2 - \mu_1 + \sum_{i=l+1}^{2l-1} n_i(\sigma) - \mu_2 \leq (2l - 1)(m - \mu)/(2l - 1) \leq m - \mu.$$

Since again $\lfloor \sum_{i=1}^{l} n_i(\sigma)/2 \rfloor - \mu_1 = \lceil (\sum_{i=1}^{l} n_i(\sigma) - 2\mu_1)/2 \rceil$, condition (iii) defining valid target configurations holds. \hfill $\Box$

We next analyze the scheduling steps of $A_2(\varepsilon)$. For the proof of Lemma 8 below we need the following two Lemmas 6 and 7.

**Lemma 6** Let $A_c$ be any algorithm of $A_2(\varepsilon)$ processing a job sequence $\sigma$. At any time there exists at most one machine $M_j \in \mathcal{M}_c$ with $\ell_s(j) > 0$ and $\ell^-((j') + \ell_s(j)) < 1 + \varepsilon'$ in the schedule maintained by $A_c$.

**Proof** Consider any point in time while $A_c$ sequences $\sigma$. Suppose that there exists a machine $M_j \in \mathcal{M}_c$ with $\ell_s(j) > 0$ and $\ell^-((j) + \ell_s(j)) < 1 + \varepsilon'$. We show that if a small job $J_t$ arrives and $A_c$ assigns it to a machine $M_{j'} \in \mathcal{M}_c$ with $\ell_s(j') = 0$, then $\ell^-((j') + pt) > 1 + \varepsilon'$ so that no new machine with the property specified in the lemma is generated. A first observation is that $M_j$ is not a class-$l$ machine because in this case $\ell^-((j))$ would be $2a_j = 2b_{l-1} = 1 + 2\varepsilon'$. Also, if $M_{j'}$ is a class-$l$ machine, there is nothing to show because, again, in this case $\ell^-((j')) \geq 1 + 2\varepsilon'$.

So assume that $A_c$ assigns $J_t$ to a machine $M_{j'} \in \mathcal{M}_c$, which is not a class-$l$ machine, and $\ell_s(j') = 0$ prior to the assignment. We first show that $\ell^-((j')) \geq \ell^-((j))$. Consider the scheduling step in which $A_c$ assigned the first small job $J_{t'}$ to $M_j$. Since $M_j$ is not a class-$l$ machine, there holds $\ell^+(j) = 2b_i$, for some $i \in \{1, \ldots, l-1\}$ and the assignment of $J_{t'}$ to $M_j$ led to a load of at most $\ell^+(j) + pt \leq 1 + 2\varepsilon' + 1/3 + 2\varepsilon' = 4/3 + 4\varepsilon' < 4/3 + \varepsilon$. Since $M_{j'}$ is not a class-$l$ machine either, $J_{t'}$ could have also been assigned to $M_{j'}$ incurring a resulting load of at most $\ell^+(j') + pt' < 4/3 + \varepsilon$ on this machine. Note that when an algorithm $A_c$ cannot assign a small job to a machine $M_j \in \mathcal{M}_c$ with $\ell_s(j) > 0$ and instead has to resort to machines $M_k \in \mathcal{M}_c$ with $\ell_s(k) = 0$, it chooses a machine having the smallest $\ell^-(k)$ value. We conclude $\ell^-((j)) \leq \ell^-((j'))$. Next consider the assignment of $J_t$. Algorithm $A_c$ would prefer to place $J_t$ on $M_j$ as it already contains small jobs. Since this is impossible, there holds $\ell^+(j) + \ell_s(j) + pt > 4/3 + \varepsilon$ and thus $pt > 4/3 + 8\varepsilon' - \ell^+(j) - \ell_s(j)$. Since by assumption...
Lemma 7 Let \( \sigma \) be any job sequence and \( A_c \) be an algorithm such that \( c \) is a valid target configuration for \( \sigma \). Let \( m \geq 2l/(\varepsilon')^2 \). Consider any point in time during the scheduling process. If in the schedule of \( A_c \) there holds \( \ell^-(j) + \ell_s(j) < 1 + \varepsilon' \), for all machines \( M_j \in \mathcal{M}_c \) but possibly one, then no further small job can arrive.

Proof Since \( c \) is a valid target configuration for \( \sigma \), the job sequence contains as many class-\( i \) jobs, for any \( i \in \{1, \ldots, l\} \), as indicated by \( c \). Hence the total processing time of large jobs in \( \sigma \) is lower bounded by \( \sum_{j=1}^{\mu} \ell^-(j) \). Hence the total processing time of jobs in \( \sigma \) is at least \( \sum_{j=1}^{\mu} (\ell^-(j) + \ell_s(j)) \), where the machine loads due to small jobs may be considered at an arbitrary point in time. Hence if there exists a time such that \( \ell_s(j) + \ell^-(j) < 1 + \varepsilon' \) for at most one \( M_j \in \mathcal{M}_c \), we obtain

\[
\sum_{j=1}^{\mu} (\ell^-(j) + \ell_s(j)) \geq (1 + \varepsilon')(\mu - 1) \geq (1 + \varepsilon') \left( \frac{1 + \varepsilon'}{1 + 2\varepsilon'} \cdot m - 1 \right) = m + \frac{(\varepsilon')^2}{1 + 2\varepsilon'} m - (1 + \varepsilon') \geq m.
\]

The last inequality holds because \( m \geq 2l/(\varepsilon')^2 \geq 2/(\varepsilon')^2 \geq (1 + \varepsilon')(2\varepsilon' + 1)/(\varepsilon')^2 \), for any \( \varepsilon' \leq 1/8 \). Hence no further small job can arrive. \( \square \)

In the further analysis we then focus on algorithms \( A_c \) such that \( c \) is a valid target configuration for \( \sigma \). The following two lemmas appropriately bound machine loads on machines of \( \mathcal{M}_c \) and \( \mathcal{M}_r \).

Lemma 8 Let \( \sigma \) be any job sequence and \( A_c \) be an algorithm such that \( c \) is a valid target configuration for \( \sigma \). Let \( m \geq 2l/(\varepsilon')^2 \). Then in the final schedule constructed by \( A_c \), each machine in \( \mathcal{M}_c \) has a load of at most \( 4/3 + \varepsilon \).

Proof We consider the scheduling steps in which \( A_c \) assigns a job \( J_t \) to a machine in \( \mathcal{M}_c \). First suppose that \( J_t \) is large. Let \( J_t \) be a class-\( i \) job, where \( 1 \leq i \leq 2l - 1 \). If \( J_t \) is assigned to an \( M_j \in \mathcal{M}_c \), then \( M_j \) must be an admissible class-\( i \) machine, i.e. prior to the assignment of \( J_t \) it contains fewer class-\( i \) jobs than specified by the target configuration. This implies that for any machine \( M_j \in \mathcal{M}_c \), its load due to large jobs
is always at most $\ell^+(j)$. The latter value is upper bounded by $2b_l \leq 2(2/3 + 4\epsilon') = 4/3 + 8\epsilon' = 4/3 + \epsilon$. Hence, in order to establish the lemma it suffices to show that whenever a small job is assigned to a machine $M_j \in \mathcal{M}_c$, the resulting load $\ell^+(j) + \ell_s(j)$ on $M_j$ is at most $4/3 + \epsilon$.

Suppose on the contrary that a small job $J_i$ arrives and $A_c$ schedules it on a machine in $\mathcal{M}_c$ such that the resulting load is greater than $4/3 + \epsilon$. Algorithm $A_c$ first tries to place $J_i$ on a machine $M_j \in \mathcal{M}_c$ with $\ell_s(j) > 0$, which has already received small jobs. By Lemma 6, among these machines there exists at most one having the property that $\ell^+(j) + \ell_s(j) < 1 + \epsilon'$. Since an assignment to those machines is impossible without exceeding a load of $4/3 + \epsilon$, $A_c$ tries to place $J_i$ on a machine $M_j \in \mathcal{M}_c$ with $\ell_s(j) = 0$. Since this is also impossible without exceeding a load of $4/3 + \epsilon$, any $M_j \in \mathcal{M}_c$ with $\ell_s(j) = 0$ must be a class-$l$ machine. This holds true because for any class-$i$ machine with $i \neq l$, there holds $\ell^+(j) \leq 2b_l - 1 \leq 1 + 2\epsilon'$ and an assignment of a small job would result in a total load of at most $1 + 2\epsilon'+1/3+2\epsilon' < 4/3+\epsilon$. Observe that any class-$l$ machine has a targeted minimal load of $2a_l = 2b_{l-1} \geq 1+2\epsilon' > 1+\epsilon'$.

We conclude that immediately before the assignment of $J_i$ the schedule of $A_c$ contains at most one machine $M_j \in \mathcal{M}_c$ with $\ell^+(j) + \ell_s(j) < 1 + \epsilon'$. Lemma 7 implies that the incoming job $J_i$ cannot be small, and we obtain a contradiction.

**Lemma 9** Let $\sigma$ be any job sequence and $A_c$ be an algorithm such that $c$ is a valid target configuration for $\sigma$. Then in the final schedule constructed by $A_c$, each machine in $\mathcal{M}_r$ has a load of at most $4/3 + \epsilon$.

**Proof** Algorithm $A_c$ assigns only large jobs to machines in $\mathcal{M}_r$. A first observation is that whenever there exists an $M_j \in \mathcal{M}_r$ that contains only one class-$i$ job with $i \in \{1,\ldots,l\}$ but no further jobs, then an incoming class-$i'$ job with $i' \in \{1,\ldots,l\}$ will not be assigned to an empty machine. This holds true because the two jobs can be combined, which results in a total load of at most $2b_l \leq 4/3 + 8\epsilon' = 4/3 + \epsilon$.

The observation implies that at any time while $A_c$ processes $\sigma$, the number of machines of $\mathcal{M}_r$ containing at least one job is upper bounded by $[n_1/2] + n_2$. Here $n_1$ denotes the total number of class-$i$ jobs with $i \in \{1,\ldots,l\}$ that have been assigned to machines of $\mathcal{M}_r$ so far. Analogously, $n_2$ is the total number of class-$i$ jobs with $i \in \{l+1,\ldots,2l-1\}$ currently residing on machines in $\mathcal{M}_r$. Since $c$ is a valid target configuration for $\sigma$, conditions (i) and (ii) defining those configurations imply $0 \leq \sum_{i=1}^{l} n_i(\sigma) - 2\mu_1$ and $0 \leq \sum_{i=l+1}^{2l-1} n_i(\sigma) - \mu_2$. Moreover, since $A_c$ assigns large jobs preferably to machines in $\mathcal{M}_c$, there holds $n_1 \leq \sum_{i=1}^{l} n_i(\sigma) - 2\mu_1$ and $n_2 \leq \sum_{i=l+1}^{2l-1} n_i(\sigma) - \mu_2$. By condition (iii) defining valid target configurations, $\left[\left(\sum_{i=1}^{l} n_i(\sigma) - 2\mu_1\right)/2\right] + \sum_{i=l+1}^{2l-1} n_i(\sigma) - \mu_2 \leq m - \mu$. Hence, while $n_2 < \sum_{i=l+1}^{2l-1} n_i(\sigma) - \mu_2$ there holds $[n_1/2] + n_2 < m - \mu$ and thus exists an empty machine $\mathcal{M}_r$ to which an incoming class-$i$ jobs with $i \in \{l+1,\ldots,2l-1\}$ can be assigned. Similarly, while $n_1 < \sum_{i=1}^{l} n_i(\sigma) - 2\mu_1$, there must exist an empty machine or a machine containing only one class-$i'$ job with $i' \in \{1,\ldots,l\}$ to which in incoming class-$i$ job with $i \in \{1,\ldots,l\}$ can be assigned. In either case, the assignment generates a load of at most $4/3 + \epsilon$ on the selected machine.

Theorem 3 now follows from Lemmas 5, 8 and 9.
5 Algorithms for MPS

We derive our algorithms for MPS. The strategies are obtained by simply combining \(A^*(\epsilon)\), presented in Sect. 2, with \(A_1(\epsilon)\) and \(A_3(\epsilon)\), respectively. In order to achieve a precision of \(\epsilon\) in the competitive ratio, the strategies are combined with a precision of \(\epsilon/2\) in their parameters. We first derive the algorithm that uses a constant number of schedules. For any \(0 < \epsilon \leq 1\), let \(A_3^*(\epsilon)\) be the algorithm obtained by executing \(A_3(\epsilon/2)\) in \(A^*(\epsilon/2)\).

**Corollary 2** \(A_3^*(\epsilon)\) is a \((4/3 + \epsilon)\)-competitive algorithm for MPS and uses \(1/\epsilon^{O(\log(1/\epsilon))}\) schedules, for any \(0 < \epsilon \leq 1\).

**Proof** Theorem 1 and Corollary 1 imply that \(A_3^*(\epsilon)\) is \((4/3 + \epsilon)\)-competitive, for any \(0 < \epsilon \leq 1\), and that the total number of schedules is the product of \(1/\epsilon^{O(\log(1/\epsilon))}\) and \(\lceil\log(1 + 12\rho/\epsilon)/\log(1 + \epsilon/(6\rho))\rceil\), where \(\rho = 4/3 + \epsilon/2\). By the Taylor series for \(\ln(1 + x)\), \(-1 < x \leq 1\), we obtain \(\ln(1 + x) \geq x/2\), for any \(0 < x \leq 1\). Hence the second term of the product is \(1/\epsilon^{O(1)}\). \(\Box\)

Next, for any \(0 < \epsilon \leq 1\), let \(A_1^*(\epsilon)\) be the algorithm obtained by executing \(A_1(\epsilon/2)\) in \(A^*(\epsilon/2)\).

**Corollary 3** \(A_1^*(\epsilon)\) is a \((1 + \epsilon)\)-competitive algorithm for MPS and uses \((m/\epsilon)^{O(\log(1/\epsilon)/\epsilon)}\) schedules, for any \(0 < \epsilon \leq 1\).

**Proof** By Theorems 1 and 2 algorithm \(A_1^*(\epsilon)\) is \((1+\epsilon)\)-competitive, for any \(0 < \epsilon \leq 1\). The total number of schedules is the product of \((\lceil m/\epsilon \rceil + 1)^{\lceil \log(4/\epsilon)/\log(1+\epsilon/4) \rceil}\) and \(\lceil \log(1 + 12\rho/\epsilon)/\log(1 + \epsilon/(6\rho)) \rceil\), where \(\rho = 1 + \epsilon/2\). Again, by the Taylor series, \(\ln(1 + x) \geq x/2\), for any \(0 < x \leq 1\). Hence both terms of the product are upper bounded by \((m/\epsilon)^{O(\log(1/\epsilon)/\epsilon)}\).

6 Lower Bounds

We develop lower bounds that apply to both MPS and MPS\textsubscript{opt}. We first present a general lemma that will allow us to derive Theorems 4 and 5 below.

**Lemma 10** Let \(\epsilon' \leq 1/3\) be a positive number such that \(1/\epsilon' \in \mathbb{N}\). Let \(h \in \mathbb{N}\) be a number with \(he' < 1/2\) and let \(k\) be the largest integer such that \(ke' < 1/2\). Let

\[
M = \left\{(m_0, \ldots, m_{k+1}) \mid m_i \in \mathbb{N}_0, \sum_{i=0}^{k+1} m_i = m \text{ and } (1/\epsilon')m_{k+1} + \sum_{i=1}^{k} im_i = mh\right\}.
\]

If a deterministic online algorithm for MPS or MPS\textsubscript{opt} achieves a competitive ratio smaller than \(\min\{1 + \epsilon', 2(1 - ke')\}\), then it must maintain at least \(|M|\) schedules.

**Proof** Let \(A\) be any deterministic online algorithm for MPS or MPS\textsubscript{opt} that maintains less than \(|M|\) schedules. We construct an adversarial job sequence \(\sigma\) such that each schedule maintained by \(A\) has a makespan of at least \(\min\{1 + \epsilon', 2(1 - ke')\}\text{OPT}(\sigma)\).
In the following we first describe the construction of $\sigma$ and then analyze the makespan of $\mathcal{A}$'s schedules.

Construction of $\sigma$: The job sequence $\sigma$ is composed of two subsequences $\sigma_1$ and $\sigma_2$, i.e. $\sigma = \sigma_1 \sigma_2$. Subsequence $\sigma_1$ consists of $mh$ jobs of processing time $\epsilon'$ each. Subsequence $\sigma_2$ depends on the schedules constructed by $\mathcal{A}$ and will be specified later.

Consider the possible schedules after $\sigma_1$ has been sequenced on the $m$ machines. We restrict ourselves to schedules having the following property: Each machine has a load of exactly 1 or a load that strictly smaller than 1/2. By the choice of $\epsilon'$, each machine of load 1 contains $1/\epsilon'$ jobs. By the choice of $k$, each machine with a load smaller than 1/2 contains up to $k$ jobs. Therefore any schedule with the stated property can be described by a vector $\vec{m} = (m_0, \ldots, m_{k+1})$, where $m_{k+1}$ is the number of machines having a load of 1 and $m_i$ is the number of machines containing exactly $i$ jobs, for $i = 0, \ldots, k$. The vector $\vec{m}$ satisfies $\sum_{i=0}^{k+1} m_i = m$ and $(1/\epsilon') m_{k+1} + \sum_{i=1}^{k} i m_i = mh$. The last equation specifies the constraint that the schedule contains $mh$ jobs. Let $M$ be the set of all these vectors, i.e.

$$M = \left\{ (m_0, \ldots, m_{k+1}) \mid m_i \in \mathbb{N}_0, \sum_{i=0}^{k+1} m_i = m \text{ and } (1/\epsilon') m_{k+1} + \sum_{i=1}^{k} i m_i = mh \right\}.$$ 

We remark that each $\vec{m} \in M$ uniquely identifies one schedule with our desired property. By the choice of $h$ there holds $m h \epsilon' < m/2$ and hence the set $M$ is not empty. Let $S$ be any schedule containing exactly $mh$ jobs of processing time $\epsilon'$ and $\vec{m} = (m_0, \ldots, m_{k+1}) \in M$. We say that $S$ is an $\vec{m}$-schedule if in $S$ there exist $m_{k+1}$ machines of load 1 and $m_i$ machines containing exactly $i$ jobs, for $i = 0, \ldots, k$.

Now suppose that $\mathcal{A}$ maintains less than $|M|$ schedules. Let $\mathcal{S}$ be the set of schedules constructed by $\mathcal{A}$ after all jobs of $\sigma_1$ have arrived. Since $|\mathcal{S}| < |M|$ there must exist an $\vec{m}^* = (m_0^*, \ldots, m_{k+1}^*) \in M$ such that no schedule of $\mathcal{S}$ is an $\vec{m}^*$-schedule. Let $\mathcal{S}^*$ be an $\vec{m}^*$-schedule in which machines are numbered in order of non-decreasing load such that $\ell^*(1) \leq \ldots \leq \ell^*(m)$. Subsequence $\sigma_2$ consists of $m - m_{k+1}^*$ jobs, where job $j$ has a processing time of $1 - \ell^*(j)$, for $j = 1, \ldots, m - m_{k+1}^*$. Hence $\sigma_2$ consists of $m_i^*$ jobs of processing time $1 - \epsilon'$, for $i = 0, \ldots, k$. These jobs arrive in order of non-increasing processing time. Each job has a processing time of at least $1 - k \epsilon' > 1/2$. The makespan of an optimal schedule for $\sigma$ is 1: The jobs of $\sigma_1$ are sequenced so that an $\vec{m}^*$-schedule is obtained. Machines are again numbered in order of non-decreasing load. Then, while the jobs of $\sigma_2$ arrive, the $j$-th job of the subsequence is assigned to machine $M_j$ in $\mathcal{S}^*$, $1 \leq j \leq m - m_{k+1}^*$.

Analysis of the makespan: We show that after $\mathcal{A}$ has sequenced $\sigma_2$, each of its schedules has a makespan of at least $\min\{1 + \epsilon', 2(1 - k \epsilon')\}$. So consider any $S \in \mathcal{S}$ and, as always, number the machines in order of non-decreasing load such that $\ell(1) \leq \ldots \leq \ell(m)$. If in $S$ there exists a machine that has a load of at least 1 + $\epsilon'$ and hence contains at least $1/\epsilon' + 1$ jobs, then there is nothing to show. So assume that each machine in $S$ contains at most $1/\epsilon'$ jobs and thus has a load of at most 1. We study the assignment of the jobs of $\sigma_2$ to $S$. If $\mathcal{A}$ places two jobs of $\sigma_2$ on the same machine, then we are done because each job has a processing time of at least 1 - $k \epsilon'$ and the resulting makespan is at least 2(1 - $k \epsilon'$). Therefore we focus on the case that $\mathcal{A}$ assigns the jobs of $\sigma_2$ to different machines.
Schedules $S$ and $S^*$ both contain jobs of total processing time $m h e'$. Since $S$ is not an $\tilde{m}$-schedule there must exist a $j_0$, $1 \leq j_0 \leq m$, such that $\ell(j_0) > \ell^*(j_0)$ and hence $\ell(j_0) \geq \ell^*(j_0) + \epsilon'$. For, if $\ell(j_0) \leq \ell^*(j_0)$ held for all $j = 1, \ldots, m$, then $\ell(j_0) = \ell^*(j_0)$ for all $j = 1, \ldots, m$ because $S$ and $S^*$ both contain jobs with a total processing time of $m h e'$. Thus $S$ would be an $\tilde{m}$-schedule and we obtain a contradiction. Each machine in $S$ has a load of at most 1 while the last $m_{k+1}^*$ machines in $S^*$ have a load of exactly 1. This implies $j_0 \leq m - m_{k+1}^*$. The first $j_0$ jobs of $\sigma_2$ each have a processing time of at least $1 - \ell^*(j_0)$. However, there exist at most $j_0 - 1$ machines in $S$ having a load strictly smaller than $\ell^*(j_0)$. Hence after $A$ has sequenced the first $j_0$ jobs of $\sigma_2$ there must exist a machine in $S$ with a load of at least $\ell(j_0) + 1 - \ell^*(j_0) \geq \ell^*(j_0) + \epsilon' + 1 - \ell^*(j_0) = 1 + \epsilon'$.

**Theorem 4** Let $A$ be a deterministic online algorithm for MPS or MPS$_{\text{opt}}$. If $A$ achieves a competitive ratio smaller than $4/3$, then it must maintain at least $\lfloor m/3 \rfloor + 1$ schedules.

**Proof** We simply apply Lemma 10 by setting $\epsilon' = 1/3$ and $h = 1$. Then $k = 1$ and

$$M = \left\{ (m_0, m_1, m_2) \mid m_i \in \mathbb{N}_0, \sum_{i=0}^{2} m_i = m \quad \text{and} \quad 3m_2 + m_1 = m \right\}.$$ 

There holds $1 + \epsilon' = 2(1 - k \epsilon') = 4/3$, which establishes the desired competitive ratio. Set $M$ contains exactly $\lfloor m/3 \rfloor + 1$ tuples because $m_2$ can take any value between 0 and $\lfloor m/3 \rfloor$ and $m_1 = m - 3m_2$. \qed

The next theorem gives a lower bound on the number of schedules required by a $(1 + \epsilon)$-competitive algorithm, where $0 < \epsilon < 1/4$. It implies that, for any fixed $\epsilon$, the number asymptotically depends on $m^{\Omega(1/\epsilon^2)}$, as $m$ increases. For instance, any algorithm with a competitive ratio smaller than 1 + 1/12 requires $\Omega(m^2)$ schedules. Any algorithm with a competitive ratio smaller than 1 + 1/16 needs $\Omega(m^3)$ schedules.

**Theorem 5** Let $A$ be a deterministic online algorithm for MPS or MPS$_{\text{opt}}$. If $A$ achieves a competitive ratio smaller than $1 + \epsilon$, where $0 < \epsilon \leq 1/4$, then it must maintain at least $(\frac{m + h - 1}{2})$ schedules, where $m' = \lceil m/2 \rceil$ and $h = \lfloor 1/(4\epsilon) \rfloor$. The binomial coefficient increases as $\epsilon$ decreases and is at least $\Omega((4\epsilon m)^{1/(4\epsilon^2)} - 1/2)$. We apply Lemma 10 with $\epsilon = 1/(4h)$ and the setting of $h = \lfloor 1/(4\epsilon) \rfloor$. Note that $\epsilon' \geq \epsilon$. There holds $k = 2h - 1$ because $(2h - 1)\epsilon' = 2h/(4h) - \epsilon' = 1/2 - \epsilon'$ and $2h\epsilon' = 2h/(4h) = 1/2$. The lower bound on the competitive ratio is min\{1 + $\epsilon'$, 2(1 - $k\epsilon'$)\} $\leq$ min\{1 + $\epsilon'$, 2(1 - (2h - 1)$\epsilon'$)\} $\leq$ min\{1 + $\epsilon'$, 2(1/2 + $\epsilon'$)\} = 1 + $\epsilon'$. This is at least 1 + $\epsilon$. The set $M$ is

$$M = \left\{ (m_0, \ldots, m_{2h}) \mid m_i \in \mathbb{N}_0, \sum_{i=0}^{2h} m_i = m \quad \text{and} \quad (1/\epsilon')m_{2h} + \sum_{i=1}^{2h-1} im_i = mh \right\}$$ 

and it remains to estimate $|M|$.
We first focus on the case that \( m \) is even. In the definition of \( M \) the critical constraint is \((1/\varepsilon')m_{2h} + \sum_{i=1}^{2h-1} im_i = mh\), which implies that not every vector of \( \{0, \ldots, m\}^{2h+1} \) represents a schedule that can be built of \( mh \) jobs. In particular, the vector \((0, \ldots, 0, m)\) of length \( 2h + 1 \) would require \( m/\varepsilon' = 4h \) jobs. Therefore, we introduce a set \( M' \) and show \( |M'| \leq |M| \). Set \( M' \) contains vectors of length \( 2h + 1 \) in which the first \( h + 1 \) entries as well as the last one are equal to 0. The other entries sum to at most \( m/2 \), i.e.

\[
M' = \left\{ (0, \ldots, 0, m'_{h+1}, \ldots, m'_{2h-1}, 0) \mid m'_i \in \mathbb{N}_0 \text{ and } \sum_{i=1}^{h-1} m'_{h+i} \leq m/2 \right\}.
\]

We show that each \( \tilde{m}' \in M' \) can be mapped to a \( \tilde{m} \in M \). The mapping has the property that any two different vectors of \( M' \) are mapped to different vectors of \( M \). This implies \( |M'| \leq |M| \).

Consider any \( \tilde{m}' = (0, \ldots, 0, m'_{h+1}, \ldots, m'_{2h-1}, 0) \in M' \). Let \( \tilde{m} = (m_0, \ldots, m_{2h}) \) be defined as follows. For \( i = h + 1, \ldots, 2h \), let \( m_i = m'_i \). For \( i = 0, \ldots, h-1 \), let \( m_i = m_{2h-i} \). Finally, let \( m_h = m - 2 \sum_{i=1}^{h-1} m_i \). Note that \( m_0 = m_{2h} = 0 \). We next show that \( \tilde{m} \in M \). There holds \( \sum_{i=0}^{2h} m_i = \sum_{i=1}^{2h-1} m_i = 2 \sum_{i=1}^{h-1} m_i + mh = m \).

Furthermore,

\[
m_{2h}/\varepsilon' + \sum_{i=0}^{2h-1} im_i = \sum_{i=1}^{2h-1} im_i = \sum_{i=1}^{h-1} (i + 2h - i)m_i + hm_h
= 2h \sum_{i=1}^{h-1} m_i + h \left( m - 2 \sum_{i=1}^{h-1} m_i \right) = mh.
\]

It follows, as desired, \( \tilde{m} \in M \). Note that the last \( h \) entries of \( \tilde{m} \) are identical to the last \( h \) entries of \( \tilde{m}' \). Hence no two vectors of \( M' \) that differ in at least one entry are mapped to the same vector of \( M \). Hence \( |M'| \leq |M| \). If the number \( m \) of machines is odd, then in the definition of \( M' \) the entries of a vector sum to at most \((m-1)/2 \).

The rest of the construction and analysis is the same. Thus, for a general number \( m \) of machines

\[
M' = \left\{ (0, \ldots, 0, m'_{h+1}, \ldots, m'_{2h-1}, 0) \mid m'_i \in \mathbb{N}_0 \text{ and } \sum_{i=1}^{h-1} m'_{h+i} \leq \lfloor m/2 \rfloor \right\}.
\]

This set contains exactly \( \binom{m'+h-1}{h-1} \) elements, where again \( m' = \lfloor m/2 \rfloor \). In the remainder of this proof we lower bound this binomial coefficient.

By Stirling’s approximation \([20]\), for any \( k \in \mathbb{N} \), there holds \( \sqrt{2\pi e(k/e)^{k+1/2}} \leq k! \leq 2\sqrt{2\pi e(k/e)^{k+1/2}} \). Hence

\[\text{(equation)}\]
\[
\binom{m' + h - 1}{h - 1} = \frac{(m' + h - 1)!}{m'!(h - 1)!} \geq \frac{(m' + h - 1)^{m' + h - 1/2}}{4\sqrt{2\pi} (m')^{m' + 1/2} (h - 1)^{h - 1/2}} \\
= \frac{1}{4\sqrt{2\pi m'}} \left(1 + \frac{h - 1}{m'}\right)^{m'} \left(1 + \frac{m'}{h - 1}\right)^{h - 1/2} \\
> \frac{1}{4\sqrt{2\pi m'}} \left(1 + \frac{m'/2 - 1/2}{1/(4\varepsilon)}\right)^{h - 1/2}.
\]

The last expression is \(\Omega((\varepsilon m)^{1/(4\varepsilon)} - 1/2 / \sqrt{m})\).

7 Conclusions

In this paper we have explored classical online makespan minimization with an essentially new model of resource augmentation, originally introduced by Kellerer et al. [32] for \(m = 2\) machines. In this framework an online algorithm may construct several schedules in parallel, the best of which is finally selected. We have presented a \((4/3 + \varepsilon)\)-competitive algorithm, for any \(0 < \varepsilon \leq 1\), that uses a constant number \(\frac{1}{\varepsilon} O(\log(1/\varepsilon))\) of schedules. We also showed a \((1 + \varepsilon)\)-competitive algorithm, for any \(0 < \varepsilon \leq 1\), that builds a polynomial number \((m/\varepsilon)^{O(\log(1/\varepsilon)/\varepsilon)}\) of schedules. For the given competitive ratios, the number of required schedules is tight or nearly tight.

An interesting direction for future research is to extend the considered model of resource augmentation, where an algorithm may maintain several candidate solutions, to other algorithmic problems. As already indicated in the introduction, online graph problems and online problems in robotics are sensible areas of investigation. The strategies developed in this paper resort to a problem variant MPS\txt{opt}, where the value of the optimum makespan is known in advance. Surprisingly, if only a single schedule may be constructed, the best competitiveness achievable by online algorithms is unknown. The ratio is in the range \([4/3, 3/2]\), see [7,14]. An exciting open problem is to settle the exact competitive factor. Last not least, a challenging open problem is to examine if the approach of constructing parallel schedules can be used in the design of good randomized online algorithms for online makespan minimization.

Acknowledgements We thank two anonymous referees for helpful comments improving the presentation of this paper.

References

1. Albers, S.: Better bounds for online scheduling. SIAM J. Comput. 29, 459–473 (1999)
2. Albers, S., Hellwig, M.: On the value of job migration in online makespan minimization. In: Proceedings of the 20th Annual European Symposium on Algorithms, Springer LNCS 7501, pp. 84–95 (2012)
3. Angelelli, E., Nagy, A.B., Speranza, M.G., Tuza, Z.: The on-line multiprocessor scheduling problem with known sum of the tasks. J. Sched. 7, 421–428 (2004)
4. Angelelli, E., Speranza, M.G., Tuza, Z.: Semi-on-line scheduling on two parallel processors with an upper bound on the items. Algorithmica 37, 243–262 (2003)
5. Angelelli, E., Speranza, M.G., Tuza, Z.: New bounds and algorithms for on-line scheduling: two identical processors, known sum and upper bound on the tasks. Discrete Math. Theor. Comput. Sci. 8, 1–16 (2006)
6. Azar, Y.: On-line load balancing. In: Fiat, A., Woeginger, G. (eds.) Online Algorithms: The State of the Art, vol. 1441, pp. 178–195. Springer, New York (1998)
7. Azar, Y., Regev, O.: On-line bin-stretching. Theor. Comput. Sci. 268, 17–41 (2001)
8. Bartal, Y., Karloff, H., Rabani, Y.: A better lower bound for on-line scheduling. Inf. Process. Lett. 50, 113–116 (1994)
9. Bartal, Y., Fiat, A., Karloff, H., Vohra, R.: New algorithms for an ancient scheduling problem. J. Comput. Syst. Sci. 51, 359–366 (1995)
10. Bender, M.A., Slonim, D.K.: The Power of team exploration: two robots can learn unlabeled directed graphs. In: 35th Annual Symposium on Foundations of Computer Science, pp. 75–85 (1994)
11. Blum, A., Chalasani, P.: An online algorithm for improving performance in navigation. SIAM J. Comput. 29, 1907–1938 (2000)
12. Blum, A., Raghavan, P., Schieber, B.: Navigating in unfamiliar geometric terrain. SIAM J. Comput. 26, 110–137 (1997)
13. Böckenhauer, H.J., Komm, D., Královič, R., Královič, R.: On the advice complexity of the k-server problem. In: Proceedings of the 38th International Colloquium on Automata, Languages and Programming, Springer LNCS 6755, pp. 207–218 (2011)
14. Böhm, M., Sgall, J., van Stee, R., Veselý, P.: Better algorithms for online bin stretching. In: Proceedings of the 12th International Workshop on Approximation and Online Algorithms, Springer LNCS 8952, pp. 23–34 (2014)
15. Cheng, T.C.E., Kellerer, H., Kotov, V.: Semi-on-line multiprocessor scheduling with given total processing time. Theor. Comput. Sci. 337, 134–146 (2005)
16. Deng, X., Kameda, T., Papadimitriou, C.H.: How to learn an unknown environment I: the rectilinear case. J. ACM 45, 215–245 (1998)
17. Emek, Y., Fraigniaud, P., Korman, A., Rosén, A.: Online computation with advice. Theor. Comput. Sci. 241(24), 2642–2656 (2011)
18. Englert, M., Ozden, M., Westermann, M.: The power of reordering for online makespan minimization scheduling. SIAM J. Comput. 43(3), 1220–1237 (2014)
19. Faigle, U., Kern, W., Turan, G.: On the performance of on-line algorithms for partition problems. Acta Cybern. 9, 107–119 (1989)
20. Feller, W.: An Introduction to Probability Theory and its Applications. Wiley, New York (1968)
21. Fleischer, R., Wahl, M.: Online scheduling revisited. J. Sched. 3, 343–353 (2000)
22. Fraigniaud, P., Gasieniec, L., Kowalski, D.R., Pelc, A.: Collective tree exploration. Networks 48, 166–177 (2006)
23. Gabay, M., Kotov, V., Brauner, N.: Semi-online bin stretching with bunch tech- niques. HAL preprint hal-00869858 (2013)
24. Galambos, G., Woeginger, G.: An on-line scheduling heuristic with better worst case ratio than Graham’s list scheduling. SIAM J. Comput. 22, 349–355 (1993)
25. Graham, R.L.: Bounds for certain multi-processing anomalies. Bell Syst. Tech. J. 45, 1563–1581 (1966)
26. Gormley, T., Reingold, N., Torng, E., Westbrook, J.: Generating adversaries for request-answer games. In: Proceedings of the 11th ACM-SIAM Symposium on Discrete Algorithms, pp. 564–565 (2000)
27. Hochbaum, D.S., Shmoys, D.B.: Using dual approximation algorithms for scheduling problems: theoretical and practical results. J. ACM 34, 144–162 (1987)
28. Irani, S.: Coloring inductive graphs on-line. Algorithmica 11, 53–72 (1994)
29. Karger, D.R., Phillips, S.J., Torng, E.: A better algorithm for an ancient scheduling problem. J. Algorithms 20, 400–430 (1996)
30. Karp, R.M., Vazirani, U.V., Vazirani, V.V.: An optimal algorithm for on-line bipartite matching. In: Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, pp. 352–358 (1990)
31. Kellerer, H., Kotov, V.: An efficient algorithm for bin stretching. Oper. Res. Lett. 41(4), 343–346 (2013)
32. Kellerer, H., Kotov, V., Speranza, M.G., Tuza, Z.: Semi on-line algorithms for the partition problem. Oper. Res. Lett. 21, 235–242 (1997)
33. López-Ortiz, A., Schuierer, S.: On-line parallel heuristics, processor scheduling and robot searching under the competitive framework. Theor. Comput. Sci. 310, 527–537 (2004)
34. Lovász, L., Saks, M.E., Trotter, W.A.: An on-line graph coloring algorithm with sublinear performance ratio. Discrete Math. 75, 319–325 (1989)
35. Raghavan, P., Snir, M.: Memory versus randomization in on-line algorithms. IBM J. Res. Dev. 38, 683–708 (1994)
36. Renault, M.P., Rosén, A., van Stee, R.: Online Algorithms with advice for bin packing and scheduling problems. Theor. Comput. Sci. 600, 155–170 (2015)
37. Rudin III, J.F.: Improved bounds for the on-line scheduling problem. Ph.D. Thesis, The University of Texas at Dallas, May (2001)
38. Rudin III, J.F., Chandrasekaran, R.: Improved bounds for the online scheduling problem. SIAM J. Comput. 32, 717–735 (2003)
39. Sanders, P., Sivadasan, N., Skutella, M.: Online scheduling with bounded migration. Math. Oper. Res. 34(2), 481–498 (2009)
40. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Commun. ACM 28, 202–208 (1985)