




Bin stretching with migration on two hierarchical machines

Islam Akaria¹ · Leah Epstein¹ 

Received: 3 May 2022 / Revised: 19 June 2023 / Accepted: 11 July 2023 /

Published online: 19 July 2023

© The Author(s), under exclusive licence to Springer-Verlag GmbH Germany, part of Springer Nature 2023

Abstract

In this paper, we consider semi-online scheduling with migration on two hierarchical machines, with the purpose of minimizing the makespan. The meaning of two hierarchical machines is that one of the machines can run any job, while the other machine can only run specific jobs. Every instance also has a fixed parameter $M \geq 0$, known as the migration factor. Jobs are presented one by one. Each new job has to be assigned to a machine when it arrives, and at the same time it is possible to modify the assignment of previously assigned jobs, such that the moved jobs have a total size not exceeding M times the size of the new job. The semi-online variant studied here is called *bin stretching*. In this problem, the optimal offline makespan is provided to the scheduler in advance. This is still a non-trivial variant for any migration factor $M > 0$. We prove tight bounds on the competitive ratio for any migration factor M . The design and analysis is split into several cases, based on the value of M , and on the resulting competitive ratio. Unlike the online variant with migration for two hierarchical machines, this case allows an online fully polynomial time approximation scheme.

Keywords Hierarchical machines · Competitive ratio · Migration factor · Semi-online scheduling

1 Introduction

In this work, we study semi-online scheduling on two hierarchical machines. Jobs are presented one by one, over a list. Each job j has a positive size (or processing time) p_j , and a hierarchy $g_j \in \{1, 2\}$. The hierarchy corresponds to the grade of service (GoS), which this job requires. There are two machines to be used for processing these jobs, where the speeds of the machines are unit, but the machines are different with

✉ Leah Epstein
lea@math.haifa.ac.il

Islam Akaria
islam.akaria@gmail.com

¹ Department of Mathematics, University of Haifa, Haifa, Israel

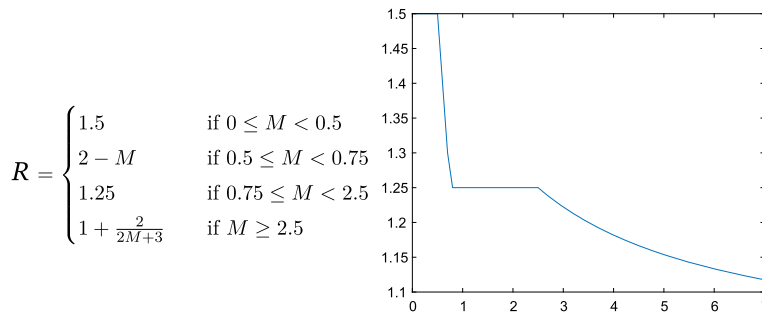


Fig. 1 The tight competitive ratio R for the bin stretching, which is the semi-online variant of our hierarchical scheduling problem with known makespan, as a function of the migration factor M

respect to their capabilities. The first machine m_1 or machine 1 can run any job, while the second machine m_2 or machine 2 cannot run jobs of hierarchy 1, and it can only receive jobs of hierarchy 2. A limited amount of migration is allowed, and when a job arrives, the algorithm may reassign some jobs, as long as their total size is (at most) proportional to the processing time of the arriving job. Specifically, there is a fixed parameter $M > 0$, called *migration factor*, and when a job j arrives, the total size of moved jobs cannot exceed $M \cdot p_j$.

The completion time (or load) of a machine is the total size of its jobs, and the makespan is defined in a standard way as the maximum load over all machines. The goal is to schedule the input jobs so as to minimize the maximum completion time, that is, the goal is to minimize the makespan. We use the usual measure for online algorithms, called the competitive ratio, for the analysis. The competitive ratio is the worst case ratio between the makespan of an online or a semi-online algorithm and the makespan of an optimal offline algorithm (for the same input).

Differently from the purely online version, we consider a semi-online model. In this semi-online variant, the optimal offline makespan is known in advance, and for convenience, we assume that its value is 1, by scaling. The variant is called bin stretching. Thus, the total size of jobs does not exceed 2, and the total size of jobs of GoS 1 does not exceed 1. These simple properties are not restrictive, but they are used in the analysis of algorithms.

In this study, we consider all possible finite and positive migration factors. We show tight bounds on the competitive ratio for every possible $M > 0$, which divides the problem into four cases, based on different values of the migration factor. The specific competitive ratios for the different cases are stated below and appear as a graph in Fig. 1.

Our algorithm will obviously schedule all jobs with hierarchy 1 to the first machine. In fact, our algorithms will never migrate jobs in this case. The reason for this is that such jobs can be arbitrarily small (and a larger job can be replaced with very small jobs, essentially without changing the input). Thus, the algorithms examine the migration option only in cases where a new job with hierarchy 2 arrives. We study the pure online variant of our scheduling problem in another article (Akaria and Epstein 2022), where the problem is similar, but the algorithm does not know the optimal offline makespan

in advance. That work still deals with two hierarchical machines and jobs arriving one by one over a list, but the optimal offline makespan changes frequently as additional jobs are presented. We note that in the current work, where we deal with bin stretching or the semi-online problem with known optimal cost, we find that the competitive ratio tends to 1 as M grows. This is in contrast to the online case with migration (Akaria and Epstein 2022), where the competitive ratio cannot be smaller than a fixed constant, no matter how large M is. Specifically, in the pure online problem (Akaria and Epstein 2022), the best possible competitive ratio is $\frac{3}{2} = 1.5$ for $M \geq 1$, and it is strictly larger for smaller M . In fact, it is exactly $\frac{5}{3}$ for $M \leq \frac{1}{3}$ (Park et al. 2006; Jiang et al. 2006; Akaria and Epstein 2022), and it is equal to $1 + \frac{1}{M+1}$ for $\frac{1+\sqrt{5}}{2} \leq M \leq 1$.

The tight bound for online bin stretching on two hierarchical machines and the case $M = 0$, which is $\frac{3}{2}$, follows from earlier work (Park et al. 2006) (see also Wu et al. 2012). In Park et al. (2006), the variant where the total size of all jobs is known in advance is studied. For this last model it is not hard to see that any algorithm is also valid for the case where the optimal offline makespan is known in advance (see below), and the lower bound on the competitive ratio, presented in that work, is simple, and the proof is valid for the case of known makespan (we show this later as a special case of the case $M < \frac{1}{2}$). Knowing the cost of an optimal solution allows us to design better algorithms in terms of their competitive ratios (compared to the case where the makespan may be arbitrary), but it does not simplify the design of an algorithm, and often the design becomes more advanced (see Graham 1966; Kellerer et al. 1997; Azar and Regev 2001; Epstein et al. 2001; Epstein 2003 for a comparison between algorithms with and without the knowledge of the cost of an optimal solution and without it, for two machines).

The two similar semi-online problems, bin stretching (Azar and Regev 2001; Epstein 2003; Böhm et al. 2017b, a; Gabay et al. 2017; Kellerer and Kotov 2013; Gabay et al. 2015), and scheduling with known total size (Kellerer et al. 1997; Angelelli et al. 2004; Cheng et al. 2005; Albers and Hellwig 2012; Kellerer et al. 2015), were both studied, in particular, for identical machines. It is not hard to see that knowing the makespan is a stronger assumption compared to knowing the total size, so an algorithm for the latter can be used for the former (with an unchanged competitive ratio), and a lower bound construction for the competitive ratio of the former can be used for the latter. The issue of the relation between the two models was addressed specifically (see Lee and Lim 2013). For two identical machines, there is essentially no difference between the two variants (Azar and Regev 2001; Kellerer et al. 1997), but for a general number of machines, it was proven that the two models are different (Albers and Hellwig 2012; Böhm et al. 2017b). It follows from previous work (Park et al. 2006; Wu et al. 2012) that for two hierarchical machines (without migration) the two variants are also very similar. In the last section, we address the same question for the problem studied here, and show that the competitive ratio for any migration factor M will be bounded away from 1, that is, it will not get closer to 1 as M grows (the limit of the function for M growing to infinity is not 1), as it is the case for bin stretching.

Online scheduling with migration, where it is allowed to migrate jobs whose total size is proportional to the size of an arriving job, was first proposed by Sanders et al. (2009), who studied the problem on identical machines (without hierarchies). That

work contains in particular a linear time *online polynomial time approximation scheme*, which is a family of online algorithms with migration, such that the competitive ratio can be arbitrarily close to 1 for sufficiently large migration factors. The required migration factor increases as the competitive ratio is closer to 1, and it is exponential in the value $\frac{1}{\varepsilon}$, where ε is the competitive ratio minus 1. For two identical speed machines, it is known that the required migration factor is polynomial in $\frac{1}{\varepsilon}$ (Wakrat 2012), and given the results here, one can also obtain such a scheme for bin stretching and two hierarchical machines. Seeing this algorithm as an offline algorithm, this gives us a fully polynomial time approximation scheme (FPTAS), though for the offline problem, this result is not new (Horowitz and Sahni 1976). However, as explained above, this is possible since the optimal offline makespan is known in advance, and not possible for online algorithms with migration (Akaria and Epstein 2022). Job migration in scheduling according to this migration model has been studied further for other variants of scheduling and other combinatorial optimization problems (Skutella and Verschae 2016; Epstein and Levin 2014, 2009, 2019; Gálvez et al. 2020; Berndt et al. 2020; Levin 2022).

As we explain above, the case of identical machines allows an online approximation scheme for any m , and this is not the case for hierarchical machines. Moreover, the lower bound will hold for any $m \geq 2$ (see a definition for multiple machines below) by constructing the input used for two machines without defining any jobs that can run on machines with indices larger than 2. However, one can still improve the result without migration by using migration for $m \geq 2$ as we show here. For the more general machine model of restricted assignment (where every job has a subset of machines where it can run), migration is not helpful at all for two machines (Akaria and Epstein 2022). Here, we show not only that bin stretching for two hierarchical machines has an online approximation scheme, but also that the migration factor is polynomial in $\frac{1}{\varepsilon}$. As explained above, we show that such a scheme is not possible for the similar variant of known total size of jobs.

The online hierarchical scheduling problem for multiple machines, or scheduling with grades of service (GoS) was first proposed by Bar-Noy et al. (2001), where an algorithm whose competitive ratio is a constant was designed. Each job, as well as each machine, has a hierarchy associated with it. A job can be scheduled on a machine only when its hierarchy is no higher than that of the machine. There are further studies of different hierarchical variants of online and semi-online scheduling problems. The work of Park et al. (2006), mentioned above, contained a study of the case of two machines for online and semi-online algorithms, while the work of Jiang et al. (2006) contained a study of the case of two machines for online algorithms (including a preemptive variant). Wu et al. (2012) investigated semi-online versions for two hierarchical machines. Specifically, they showed tight bounds of $(1 + \sqrt{5})/2$ for the case where the largest processing time of any job is known in advance. There are also multiple studies of the online hierarchical scheduling problem for parallel machines, its special cases where there cannot be a large number of hierarchies (Zhang et al. 2009; Crescenzi et al. 2004; Jiang 2008; Tan and Zhang 2011; Lim et al. 2011; Lee et al. 2014), and semi-online variants with known total size of jobs (Lee et al. 2014). There is vast literature focusing on semi-online hierarchical scheduling on two

machines. Xiao et al. (2019) investigated several variants of the problem (including the case where total size of low-hierarchy is known, and the case where the total size of each hierarchy is known, see also Luo and Xu 2014; Chen et al. 2015). There is work on other semi-online models and hierarchical machines, such as bounded sizes and combined information (Liu et al. 2011; Luo and Xu 2016; Zhang et al. 2015). There is also work for similar variants with different objectives (Qi and Yuan 2019; Luo and Xu 2015). It is worth noting that such models of semi-online scheduling were also studied for other scheduling problems without hierarchies (see for example He and Zhang 1999; Tan and He 2002; Dósa and He 2004; Min et al. 2011).

Our work deals with a setting where there are two machines. This models many home offices and small businesses. Moreover, hierarchical machines model a situation where one of the two computers cannot run tasks of a certain type, for example, due to memory limitations. The goal of migration is to obtain better solutions at the expense of allowing some changes in the schedule. Our results show that the exact amount of migration is important, since increasing the migration factor allows one to obtain a better performance. Interestingly, many of our algorithms apply a migration step only once, and this already allows them to obtain an improved performance. This last property is of interest from a real-life point of view, since changes to the solution are time consuming. The assumption of the knowledge of the optimal offline makespan is a model of situations where one knows what to expect from an input, but does not know the specific input.

Roadmap The paper is organized as follows. We provide definitions and notation in the second section. In the other sections, we discuss the problem and split it into four cases with respect to the value of the migration factor M (see Fig. 1 for a specification of the bounds). Obviously, the competitive ratio as a function of M is monotonically non-increasing. We design and analyze four algorithms, and we also prove a matching lower bound on the competitive ratio for each case, obtaining tight bounds for all finite values of M . The first case is $M \geq \frac{5}{2}$, for which the tight bound on the competitive ratio as a function of M is $\frac{2M+5}{2M+3}$. In the second case, $\frac{3}{4} \leq M < \frac{5}{2}$, the tight bound is on the competitive ratio is 1.25, achieved by an algorithm whose migration factor does not exceed $\frac{3}{4}$, while the lower bound of 1.25 on the competitive ratio will hold for any semi-online algorithm with $M \leq 2.5$. In the third case, $\frac{1}{2} \leq M < \frac{3}{4}$, we prove a tight bound of $2 - M$ by presenting two algorithms defined over two different domains in the interval $[\frac{1}{2}, \frac{3}{4})$, and proving a lower bound on the competitive ratio for any algorithm and the suitable value of M . In the last case, $0 \leq M < \frac{1}{2}$, we show that this case is equivalent to the case without migration, by presenting a lower bound of $\frac{3}{2}$ on the competitive ratio for any online algorithm whose migration factor is below $\frac{1}{2}$. The idea for the lower bound is similar to that which was presented in Park et al. (2006), and the algorithm presented in that work (with migration factor $M = 0$) can be used, since this is an algorithm for known total size. At the end of the fourth case, we show that if replacing the assumption of known makespan with the assumption of known total size, the problem would have been different, unlike the case $M = 0$, for which the two problems are known to be similar (Park et al. 2006; Wu et al. 2012).

2 Preliminaries

Throughout the paper we will use the following notation. Every job will be denoted by its index in $\{1, 2, \dots, n\}$, and it will be presented to the online algorithm in this order, where the number of jobs n is not known in advance. Each job j (also denoted by J_j) is an ordered pair $J_j = (p_j, g_j)$ (or $J_j : (p_j, g_j)$), such that $p_j > 0$ is the processing time and $g_j \in \{1, 2\}$ is the grade of service (GoS or hierarchy) of job j . A job with hierarchy g_j can be assigned to any machine in $\{1, \dots, g_j\}$.

The set of jobs with GoS 1 is denoted by X , and by definition, these jobs are always assigned to machine 1. We let Y be the set of jobs with GoS 2, that are assigned to the second machine by a fixed algorithm at a certain point in time during the run of the algorithm. Similarly, the set of jobs with GoS 2 that are scheduled on the first machine at a certain time (by the same algorithm) is denoted by Z . Note that jobs of hierarchy 2, which are assigned to a machine at a certain time, could have been scheduled to that machine by the algorithm at their arrival times, or they might have been assigned to the other machine, but they were migrated to this machine at a later time. Any job may be migrated multiple times. Our algorithms will use an additional variable, W . This variable denotes a subset of jobs with GoS 2 selected by the algorithm, where the algorithm may migrate it or it may keep it assigned to the second machine.

We let x_j, y_j, z_j , and w_j denote the total sizes of the jobs of X, Y, Z , and W , respectively, at time j , that is, just after job j was scheduled (and all migrations corresponding to the arrival of j have been performed). We let $x_0 = y_0 = z_0 = w_0 = 0$, since the sets are empty, before any job was presented to the algorithm.

Thus, the loads of m_1 and m_2 , after j was assigned, are $x_j + z_j$ and y_j , respectively. We sometimes let X_j, Y_j, Z_j , and W_j denote the sets X, Y, Z , and W (respectively) just after j was assigned, for clarity. The maximum processing time and the second maximum processing time out of the processing times of the jobs of Y_{j-1} are denoted by $p_j^{\max Y}$ and $p_j^{\max Y, 2}$, and the suitable jobs are denoted by $j^{\max Y}$, $j^{\max Y, 2}$, respectively. Each of these values is defined to be zero if it is not well-defined (which happens in the case $|Y_{j-1}| \leq 1$).

We let T_1 and T_2 be the completion times (or loads) at the end of the input of the first and the second machine respectively, for the algorithm, i.e. the makespan is equal to $\max\{T_1, T_2\}$. We call those loads *final*. We also let OPT_j denote the makespan of an optimal offline solution OPT just after job j was scheduled. For a complete input with n jobs, we have $T_1 = x_n + z_n$ and $T_2 = y_n$. For any input I , let $c^*(I)$, $c_{Alg}(I)$ denote the makespan of an optimal solution and the solution of algorithm Alg , respectively. In particular, we have $c^*(I) = OPT_n$. The competitive ratio for input I is therefore $\frac{c_{Alg}(I)}{c^*(I)}$, and the competitive ratio of Alg is the supremum of these values over all possible instances I .

An illustration of the definitions is provided in Fig. 2. The form of W is according to Algorithm A defined in Sect. 3.

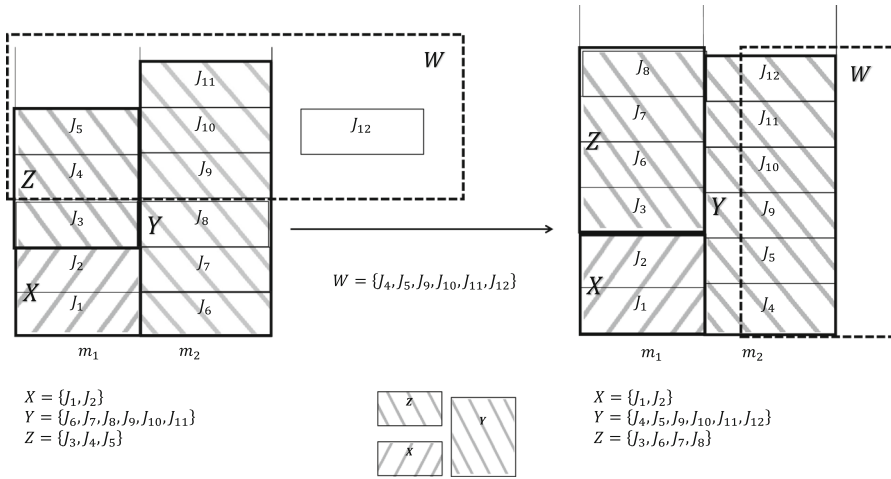


Fig. 2 An illustration of the sets X, Y, X , and W , the selection of W by Algorithm A, and the resulting migration action

3 The case $M \geq 2.5$

In this section, we prove a tight bound on the competitive ratio for the semi-online problem and the case $M \geq 2.5$. Let us define a value μ , which is a function of M , by $\mu = \frac{2}{2M+3}$, where $0 < \mu \leq \frac{1}{4}$ (and $\frac{1}{\mu} \geq 4$). The tight bound on the competitive ratio for this case of the problem is equal to $1 + \mu = 1 + \frac{2}{2M+3}$, where this value tends to 1 for M growing to infinity, and it is never exceeds $\frac{5}{4}$. The algorithm whose competitive ratio is $1 + \mu$ is based on keeping the final completion time of the second machine in the interval $[1 - \mu, 1 + \mu]$. It is obvious that if this is achieved, the completion time for the first machine will not exceed $1 + \mu$, because as mentioned earlier, the sum of all jobs is not greater than 2. After we complete the analysis of our algorithm, we will prove a lower bound on the competitive ratio for every algorithm that is equal to $1 + \mu$. As a result, we get a tight bound of $1 + \mu$ on the competitive ratio for this case, where μ is the above function of M .

One of the ideas of this algorithm and our other algorithms is that the second machine should not be underfull, since the first machine may receive jobs that cannot be assigned to the second machine. Due to the migration option, if the algorithm encounters a situation where a job of GoS 2 is large and cannot be assigned in a suitable way, our algorithms will try to migrate jobs, exploiting the fact that the new job is large, and a large total size can be moved.

3.1 An algorithm

Algorithm A

- Let $X = \emptyset, Y = \emptyset, Z = \emptyset$;
- Repeat until all jobs have been assigned:

1. Receive job j with p_j and g_j ;
2. If $g_j = 1$ holds or $y_{j-1} \geq 1 - \mu$ holds (or both hold), schedule j on the first machine and update: $X \leftarrow X \cup \{j\}$, or $Z \leftarrow Z \cup \{j\}$, respectively. return to step 1.
3. If $y_{j-1} + p_j \leq 1 + \mu$ holds, schedule j on the second machine and update: $Y \leftarrow Y \cup \{j\}$, return to step 1.
4. Let W be a subset of $Z \cup Y \cup \{j\}$ of maximum total size not exceeding 1 (which is found by solving a subset sum problem). Update the schedule such that all jobs of W are assigned to the second machine, and all other jobs are assigned to the first machine. Update $Y \leftarrow W$, $Z \leftarrow \{Z \cup Y \cup \{j\}\} \setminus W$, return to step 1.

Examples for this algorithm are provided in “Appendix A.1”. The last step of the algorithm is applied when the new job is relatively large. In this case the algorithm tries to imitate an optimal solution. The algorithm applies a procedure that searches for a subset W with a compatible total size, in order to maintain the balance of the machines, such that the completion times will not exceed $1 + \mu$ immediately after step 4 is applied (in fact, they will not exceed 1 in this case, as we show in the proof of Lemma 3.4). Steps 2 and 3 deal with simple cases, where the new job has GoS 1, or the job can be assigned without any migration such that no machine will have completion time above $1 + \mu$. For the cases with a job with a grade of service 2, this will be obvious, while the case where the grade of service of the new job is 1 will be analyzed. In particular, if the section condition of step 2 is satisfied, all new jobs can be assigned to the first machine without any migration. Step 4 is the case where jobs of GoS 2 should be rearranged, such that the schedule is more similar to that of the current optimal solution. We present examples for the action of this algorithm and the other algorithms in the appendix. We show that for this algorithm (only), the rearrangement step may be applied multiple times.

Note that the running time of this algorithm is exponential due to the last step where the subset sum problem is solved. It can be found by a slow exact algorithm. By applying a fully polynomial time approximation scheme for this problem rather than an exact algorithm, we can obtain a polynomial running time at the expense of a slightly larger competitive ratio. Now, we prove an upper bound on the competitive ratio, and analyze its migration factor.

Consider a fixed optimal offline solution OPT , and let o_{ji} denote the load of m_i in this solution for $i = 1, 2$, after j jobs were assigned. We have $o_{ji} \leq 1$, since the final optimal offline makespan is 1.

Lemma 3.1 *After j jobs have been assigned by the algorithm, the load of the second machine y_j is at least $\min\{1 - \mu, o_{j2}\}$.*

Proof Assume by contradiction that there is an index j for which $y_j < \min\{1 - \mu, o_{j2}\}$, and that j is the minimum index for which this holds. Consider the most recent time before time j , $j' \leq j$, that a job of GoS 2 was assigned by the algorithm not by applying step 3, if such a time exists.

If there is no such j' , this means that for jobs $\{1, 2, \dots, j\}$ it holds that all jobs of GoS 2 are assigned by step 3 to machine m_2 (this set may contain any number of jobs,

and it could be empty), while machine m_1 has exactly the jobs of GoS 1. Machine m_2 has a subset of jobs with GoS 2 in any solution including the optimal one, while the algorithm assigns all these jobs to m_2 , and therefore $y_j \geq o_{j2} \geq \min\{1 - \mu, o_{j2}\}$.

If there was an assignment of a job $j' \leq j$ whose grade of service is 2 by step 2, and there was no assignment by step 4 (or by step 2) up to and including job j , this means that $y_{j'-1} \geq 1 - \mu$, and the load of m_2 could not decrease until time j (since step 4 is not applied, there was no migration, and loads are monotonically non-decreasing as a function of the job index, starting from time $j' - 1$), so $y_j \geq y_{j'-1} \geq 1 - \mu$.

Finally, assume that step 4 was applied for job $j' \leq j$, and afterwards no job with GoS 2 was assigned by step 2 (or by step 4) up to and including time j . Machine m_1 received a set of jobs of GoS 2 in the application of step 4 for job j' . When step 4 is performed, machine m_2 receives a set of jobs of maximum size not exceeding 1 out of all possible subsets of already existing jobs of GoS 2. Since the optimal offline makespan at termination is 1, the optimal makespan at this time is also at most 1, and therefore m_2 has a set of jobs of total size at most 1 in optimal solutions, that is, $o_{j2} \leq 1$, and since one of the options for W is the set of jobs of m_2 in the optimal solution at this time, we find $y_{j'} = w_{j'} \geq o_{j2}$. If $j' = j$, we are done (since we already reach a contradiction), and otherwise, according to the assumption we have that $y_j < \min\{1 - \mu, o_{j2}\}$. By the choice of j' , all jobs in $\{j' + 1, \dots, j\}$ whose GoS is 2 are assigned by step 3. Thus, the jobs assigned to machine m_2 at time j (the jobs of the set Y_j) are exactly those of $W_{j'}$ together with all jobs of GoS 2 among $\{j' + 1, \dots, j\}$. Let the total size of this last set of jobs be denoted by $\lambda \geq 0$. Based on the optimal solution, we find that among jobs $1, 2, \dots, j'$ there is a subset of jobs of GoS 2 whose total size is at least $o_{j2} - \lambda$, where $o_{j2} - \lambda \leq 1$ since $o_{j2} \leq 1$ and $\lambda \geq 0$. This subset could have been chosen as $W_{j'}$ and therefore $w_{j'} \geq o_{j2} - \lambda$. However, the current load of m_2 for the algorithm after j is assigned satisfies $y_j = w_{j'} + \lambda$. Combining with $y_j < o_{j2}$ gives $o_{j2} > y_j = w_{j'} + \lambda \geq o_{j2}$, which is a contradiction. \square

Corollary 3.2 *After $j \geq 1$ jobs have been assigned, the load of the first machine is at most $1 + \mu$.*

Proof By Lemma 3.1, we have $y_j \geq \min\{1 - \mu, o_{j2}\}$. We also have $x_j + y_j + z_j = o_{j1} + o_{j2}$, which is the total size of the first j jobs. We bound from above the load of m_1 , which is equal to $x_j + z_j$ as follows. If $y_j \geq 1 - \mu$, using $x_j + y_j + z_j \leq 2$, we get $x_j + z_j \leq 2 - y_j \leq 1 + \mu$. If $y_j \geq o_{j2}$, we get $x_j + z_j = o_{j1} + o_{j2} - y_j \leq o_{j1} \leq 1$, since an optimal solution for a sub-input has makespan not exceeding 1. \square

Lemma 3.3 *After j jobs have been assigned, the load of the second machine is at most $1 + \mu$.*

Proof We prove the claim by induction. Obviously, it holds for $j = 0$ since $y_0 = 0$. If $y_j = y_{j-1}$, we are done. Otherwise, one of steps 3 and 4 was applied. If j is assigned in step 3, then we have $y_j = y_{j-1} + p_j \leq 1 + \mu$. Otherwise, $y_j = w_j \leq 1$. \square

Lemma 3.4 *For every $M \geq 2.5$, the migration factor of the algorithm is at most $M = \frac{1}{\mu} - \frac{3}{2}$.*

Proof The only step where jobs may be migrated is step 4, and in this case the GoS of the new job j is 2 ($g_j = 2$). Note that j was not assigned to any machine prior to this step and therefore it is not migrated but only other jobs of W_j may be migrated.

We analyze the loads of the two machines before and after the reassignment. By the action of the algorithm, $y_j = w_j \leq 1$, and $w_j \geq o_{j2}$ (which holds by definition). Moreover, by the choice of W_j such that $w_j \geq o_{j2}$, we find that the resulting load of m_1 will be at most $o_{j1} + o_{j2} - y_j = o_{j1} + o_{j2} - w_j \leq o_{j1} \leq 1$. Thus, both loads will be at most 1 for the algorithm after the assignment of j is completed.

Before the application of step 4, since step 3 is not applied, it holds that $y_{j-1} + p_j > 1 + \mu$, and we have $x_{j-1} + z_{j-1} \leq 2 - y_{j-1} - p_j \leq 1 - \mu$. Since step 2 was not applied and $g_j = 2$, it was the case that $y_{j-1} < 1 - \mu$, so we have that both loads were at most $1 - \mu$. We note the case where one of the machines already has a total size of at least $1 - \mu$ is in fact easy in the process of assigning a job of GoS 2.

Let $i \in \{1, 2\}$ be the machine that has j after the reassignment. The load of machine i will be at most 1, and the total size of jobs migrated from the other machine m_{3-i} is at most $1 - p_j$. Since the load of m_i before the migration is at most $1 - \mu$, this is an upper bound on the total size of jobs migrating to m_{3-i} . We find that the total size of migrated jobs does not exceed $2 - p_j - \mu$.

We bound the value p_j from below. Since step 3 was not applied, we have $y_{j-1} + p_j > 1 + \mu$. Using $y_{j-1} < 1 - \mu$ as well, we have $p_j > 1 + \mu - y_{j-1} > (1 + \mu) - (1 - \mu) = 2 \cdot \mu$. Thus, the migration factor for the step of assigning j is below $\frac{2 - p_j - \mu}{p_j} < \frac{2 - 3\mu}{2\mu} = \frac{1}{\mu} - \frac{3}{2}$. □

We conclude with the following theorem.

Theorem 3.5 *The competitive ratio for Algorithm A is at most $1 + \mu = 1 + \frac{2}{2M+3} = \frac{2M+5}{2M+3}$, and its migration factor is at most M .*

Proof The proof follows directly from the lemmas and corollary above. □

3.2 A lower bound on the competitive ratio of the case where $M \geq 2.5$

Theorem 3.6 *The competitive ratio for every algorithm when $M \geq 2.5$ is at least $1 + \mu = 1 + \frac{2}{2M+3}$.*

Proof We define an input based on the actions of a given deterministic online algorithm. Let the first two jobs of the input be $J_1 = (1 - \gamma, 2)$ and $J_2 = (1 - 2\gamma, 2)$, where γ is a constant such that $0 < \gamma < \mu \leq \frac{1}{4}$ (whose value is close to μ).

Consider the schedule of the online algorithm after the first two jobs were presented and all migrations were performed. If both jobs are assigned to the same machine by the algorithm at this time, the makespan is $2 - 3\gamma \geq \frac{5}{4} \geq 1 + \mu$. In this case there are also very small jobs of GoS 2 and total size 3γ (for example, $3M$ jobs of size $\frac{\gamma}{M}$ each), so the makespan of an optimal solution is indeed 1, and the first two jobs cannot be migrated, since the migration factor would be at least $\frac{1-2\gamma}{\frac{\gamma}{M}} = M \cdot (\frac{1}{\gamma} - 2) > 2M$.

If the algorithm scheduled the first job on the first machine, and it scheduled the second job on the second machine, the input proceeds with a third job $J_3 = (2\gamma, 1)$.

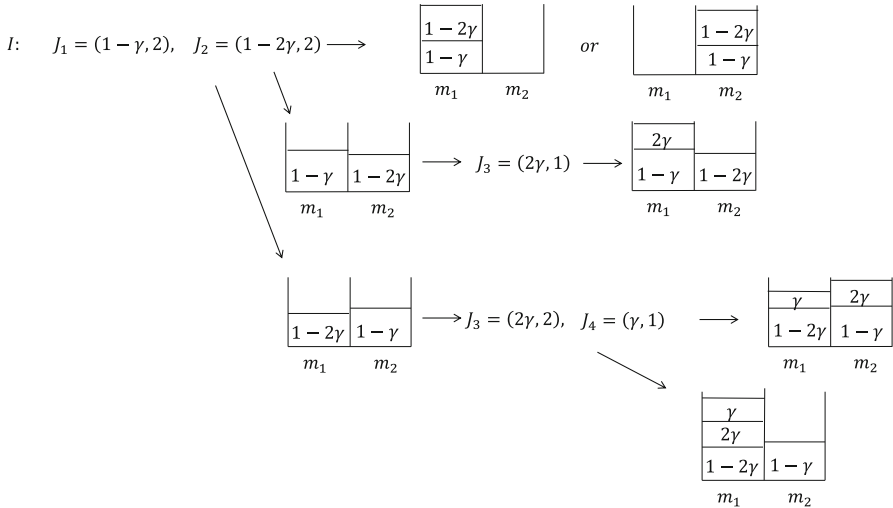


Fig. 3 The schedules produced by an online algorithm with migration factor $M \geq 2.5$ in the main cases of the proof of Theorem 3.6, represented as a decision tree. Some of the cases where both large jobs are assigned to the same machine as a result of migration are omitted

The optimal offline cost now is exactly 1. This job cannot cause the migration of both previous jobs simultaneously as $\frac{2-3\gamma}{2\gamma} = \frac{1}{\gamma} - \frac{3}{2} > \frac{1}{\mu} - \frac{3}{2} = M$. If it causes the migration of one existing job, there will be a machine of load $2 - 3\gamma > 1 + \mu$, and the situation is as in the case where the two jobs were assigned to the same machine. Finally, if there is no migration, the first machine has a load of $1 + \gamma$, as J_3 cannot be assigned to m_2 . As mentioned earlier, an optimal solution has a makespan of 1, since it swaps the machines of the first two jobs.

If the algorithm schedules the first job on the second machine, and the second job on the first machine, we continue to the next jobs in the input that are $J_3 = (2\gamma, 2)$, $J_4 = (\gamma, 1)$. Once again, none of these jobs can cause the migration of more than one job out of the first two jobs, and the migration of just one job results in a makespan above $1 + \gamma$. Consider the schedule after termination. Machine m_1 has jobs J_2 and J_4 . If it also has J_3 , its load is $1 + \gamma$. Otherwise, m_2 has J_1 and J_3 , and its load is $1 + \gamma$, so the makespan of the algorithm is $1 + \gamma$ in both cases. An optimal solution can schedule J_1 and J_4 on m_1 and the other two jobs on m_2 .

Because all optimal schedules for the above inputs have makespan 1, while the algorithm always has makespan not below $1 + \gamma$, by letting γ tend to μ , we get a lower bound of $1 + \mu$ on the competitive ratio of any online algorithm.

The diagram of Fig. 3 describes the difficult cases of the aforementioned input. □

4 The case $\frac{3}{4} \leq M < \frac{5}{2}$

In this section, we prove a tight bound on the competitive ratio for the same semi-online problem, but with a different value for the migration factor M , which will now

satisfy $\frac{3}{4} \leq M < \frac{5}{2}$. The tight bound on the competitive ratio for this case of the problem is equal to 1.25. The proof of a tight bound for this problem in this case is divided into two parts as in the previous case. In the first part we present Algorithm *B*, whose competitive ratio is 1.25 and its migration factor does not exceed $\frac{3}{4}$. The second part is a lower bound of 1.25 on the competitive ratio for any online algorithm with $M \leq 2.5$. This part was already proved in the previous section, since this is the lower bound for the case $M = 2.5$, and we do not repeat it here.

The idea behind this algorithm is similar to that of Algorithm *A*, except that Algorithm *B* maintains the load of machine m_2 such that it will not exceed 1.25, and such that the final load will be within a fixed interval $[0.75, 1.25]$. Due to this property (the lower bound on the load of m_2), the final load of machine m_1 will not exceed 1.25. Now, we present the algorithm and analyze it, and prove an upper bound on the competitive ratio. This algorithm and all further algorithms have polynomial running times.

4.1 An algorithm

Algorithm *B*

Let $X = \emptyset$, $Y = \emptyset$, $Z = \emptyset$, $W = \emptyset$;

Repeat until all jobs have been assigned:

1. Receive job j with p_j and g_j ;
2. If $g_j = 1$ holds or $y_{j-1} \geq 0.75$ holds (or both), assign j to the first machine and update: $X \leftarrow X \cup \{j\}$, or $Z \leftarrow Z \cup \{j\}$, respectively.
return to step 1.
3. If $y_{j-1} + p_j \leq 1.25$ holds, assign j to the second machine and update: $Y \leftarrow Y \cup \{j\}$,
return to step 1.
In the remaining cases Y_{j-1} is non-empty.
4. If $p_j \geq 0.75$ holds, sort the jobs in Y by non-increasing size, and update W to be the maximum length prefix of the sorted list with total size at most $0.75 \cdot p_j$ (where W may be empty).
 - 4.1. If $y_{j-1} - w_j + p_j > 1.25$ holds, assign j to the first machine and update: $Z \leftarrow Z \cup \{j\}$.
 - 4.2. Otherwise, migrate the jobs of W to the first machine, assign j to the second machine and update: $Y \leftarrow (Y \setminus W) \cup \{j\}$ and $Z \leftarrow Z \cup W$.
 return to step 1.
5. If $p_j + p_j^{\max Y} > 1.25$ holds, assign j to the first machine and update: $Z \leftarrow Z \cup \{j\}$.
Otherwise (in the case $p_j + p_j^{\max Y} \leq 1.25$), apply the next steps:
 - 5.1. If $p_j^{\max Y} \geq \frac{y_{j-1}}{2}$ holds, update $W \leftarrow Y \setminus \{j^{\max Y}\}$.
 - 5.2. If $0.25 \leq p_j^{\max Y} < \frac{y_{j-1}}{2}$ holds, update $W \leftarrow \{j^{\max Y}\}$.
 - 5.3. If $p_j^{\max Y} < 0.25$ holds (which is the remaining option), sort the jobs of Y by non-increasing size and update W to be the minimum length prefix of the sorted list with total size at least 0.25, and the entire list Y if no such prefix

- exists. If $w_j > 0.75 \cdot p_j$ holds, update $W \leftarrow Y \setminus W$, and otherwise keep W unchanged.
- 5.4. No matter how W was computed, migrate all jobs of W to the first machine, assign j to the second machine and update: $Y \leftarrow (Y \setminus W) \cup \{j\}$ and $Z \leftarrow Z \cup W$.
- return to step 1.

Examples for this algorithm are provided in “Appendix A.2”. The idea of the first three steps in Algorithm *B* is the same idea of the Algorithm *A*. In the last steps, step 4 and 5, when Algorithm *B* is forced to migrate jobs, the idea is different. In these steps, the algorithm tries to find a subset W of Y (in polynomial time), such that it can migrate this set to machine m_1 . Jobs are only migrated from machine m_2 to machine m_1 . The algorithm attempts to find such a subset of jobs that their total size satisfies the following property for w_j : $\gamma_j \leq w_j \leq 0.75 \cdot p_j$, where we let $\gamma_j = y_{j-1} + p_j - 1.25$. The value γ_j is a lower bound on the total size that has to be migrated from the second machine to obtain $y_j \leq 1.25$, in order to allow j to be scheduled on the second machine. If step 4 is reached, by the condition of step 3 (which does not hold), γ_j is strictly positive (and so is y_{j-1} which is implied by $p_j \leq 1$, and this is the reason that in steps 4, 5 it is assumed that Y_{j-1} is non-empty). Specifically, in the analysis of cases 4,5, we will use the property $y_{j-1} > 0$, on which we base the assumptions $Y_{j-1} \neq \emptyset$ and $p_j^{\max Y} > 0$.

The way the algorithm selected W is based on the required migration factor of 0.75, and the correctness is based in particular on the next fact. No input can have three jobs for which the total processing time of every subset of two jobs out of the three is greater than 1. This holds since the optimal offline cost is 1, and by the pigeonhole principle that states that there is a machine that receives at least two of these three jobs. In both steps 4 and 5, if the algorithm does not find a set W as required, then job j will be assigned to machine m_1 (this is tested in advance in both steps). In what follows, we prove that for each input, once the algorithm enters step 4 or step 5, it will never enter any of those steps again. In both cases (no matter whether there is a suitable subset W or not) the final load of the two machines will be at most 1.25.

In the next lemma we discuss step 4, and the case where the new job is assigned to the first machine, which is done since the condition $y_{j-1} - w_j + p_j > 1.25$ holds by the first option of this case. Due to the action of case 4, if W is non-empty, it holds that $0 < p_j^{\max Y} \leq 0.75 \cdot p_j$.

Lemma 4.1 *Consider step 4. If j is assigned to the first machine and W_j is not empty, then it holds that $p_j + p_j^{\max Y, 2} > 1$ and $p_j^{\max Y} + p_j^{\max Y, 2} > 0.75 \cdot p_j > 0.5$ (and $p_j + p_j^{\max Y} > 1$ holds as well).*

Proof We start with the properties which hold for any job j assigned to machine m_1 in step 4. Since this step is applied, we have $p_j \geq 0.75$. Since step 2 was not applied, we have $y_{j-1} < 0.75$. Since step 3 was not applied, we have $y_{j-1} + p_j > 1.25$.

The condition for assignment of j to the first machine is $y_{j-1} - w_j + p_j > 1.25$. By this condition, we get that $w_j < y_{j-1} + p_j - 1.25$ holds. By using $w_j < y_{j-1} + p_j - 1.25$ and $y_{j-1} < 0.75$, we have $w_j < p_j - 0.5$. By using $p_j \leq 1$, we find $w_j < p_j - \frac{p_j}{2} = \frac{p_j}{2}$. By the same property, we also have $y_{j-1} - w_j > 0.25$, so

there is at least one job of Y_{j-1} that was not included in W_j . In particular, we find that $p_j^{\max Y, 2}$ is well-defined (the second largest job is an actual job and the value is not zero), since W_j and $Y_{j-1} \setminus W_j$ are non-empty (for W_j this is one of the conditions of the lemma).

Let J_k be the first job (with the maximum processing time) in $Y \setminus W$ according to the sorted list of Y_{j-1} (which was used to find W_j). Let $\alpha \geq 2$ be the index of k in the list for Y_{j-1} (it holds that $\alpha \neq 1$, since W is non-empty). Every job among the first α jobs of this list has size at least p_k , so $w_j \geq (\alpha - 1) \cdot p_k$.

Consider the case $\alpha \geq 3$, where W_j consists of at least two jobs, and $w_j \geq 2 \cdot p_k$ (or equivalently, $p_k \leq \frac{w_j}{2}$). In this case, we have $w_j + p_k \leq \frac{3 \cdot w_j}{2}$ and $w_j + p_k > 0.75 \cdot p_j$ (by the choice of W), yielding $w_j > \frac{p_j}{2}$, which is a contradiction to a property proved earlier.

Thus, we are left with the case $\alpha = 2$. In particular, $p_k = p_j^{\max Y, 2}$, and by $w_j + p_k > 0.75 \cdot p_j$ we obtain that $p_j^{\max Y} + p_j^{\max Y, 2} > 0.75 \cdot p_j > 0.5$, as $p_j \geq 0.75$. Since $w_j = p_j^{\max Y}$, we get $p_j^{\max Y} < p_j - 0.5$, or alternatively, $p_j - p_j^{\max Y} > 0.5$, and we also use $p_j^{\max Y} + p_j^{\max Y, 2} > 0.5$, where we already proved the last inequality. Taking the sum of the two inequalities provides the first property of the lemma. It also holds that $p_j + p_j^{\max Y} > 1$, because $p_j^{\max Y} \geq p_j^{\max Y, 2}$. □

Lemma 4.2 *For any job j satisfying $p_j \leq 0.5$, j is assigned in step 2 or step 3. Once the property $y_j \geq 0.75$ holds for some job j , all jobs (of any size and GoS) arriving after j will be scheduled in step 2.*

Proof We start with the first part. If $g_j = 1$, j is assigned in step 2.

Consider the case $g_j = 2$. If prior to the arrival of j it holds that $y_{j-1} \geq 0.75$, it is also assigned in step 2. Thus, we are left with the case $g_j = 2$ and $y_{j-1} < 0.75$. In this case, $y_{j-1} + p_j < 1.25$, and therefore j is assigned in step 3.

The second part holds because once $y_j \geq 0.75$ holds, it can be shown inductively that every job is assigned in step 2 (to m_1), and the property will hold after the assignment. □

Lemma 4.3 *For every input, the algorithm enters step 4 at most once. Furthermore, if the algorithm enters step 4, it will assign all future jobs in steps 2 and 3.*

Proof If step 4 was not applied at all during the execution, we are done. Otherwise, let j be the first job for which the algorithm enters step 4, where in particular, we have $p_j \geq 0.75$. Since step 4 is reached, step 3 was not applied and $y_{j-1} + p_j > 1.25$ holds. Recall that in this case Y_{j-1} is non-empty.

If j is assigned to machine m_1 and W is empty, the first job in the sorted order satisfies $p_j^{\max Y} > 0.75 \cdot p_j > 0.5$. Since both j and an earlier job (the largest job of the current set Y) have sizes above $\frac{1}{2}$, two jobs of sizes above 0.5 have arrived already. The input can contain at most two such jobs, so any job that arrives after j has a processing time of at most 0.5, and by the previous lemma, every such job is assigned in step 2 or step 3.

If j is assigned to machine m_1 and W is not empty, then by the second property of Lemma 4.1, the set Y_{j-1} contains two jobs whose total processing time is greater than

$0.75 \cdot p_j > 0.5$. None of the two jobs can be scheduled on the same machine with j in any optimal solution by the first property of Lemma 4.1. Any optimal solution has one machine with j , whose size is above $\frac{1}{2}$, and another machine with two jobs that arrived before j , where their total size is also above $\frac{1}{2}$. Thus, each job that arrives after j has processing time less than 0.5 (since any job that arrives after j should be with scheduled with j or with the above two largest jobs of Y_{j-1} in any optimal solution), and once again all further jobs are scheduled in a step prior to step 4, by the first part of Lemma 4.2.

Finally, if j is assigned to machine m_2 , no matter what the exact properties of W are, we get $y_j \geq p_j \geq 0.75$, and by the second part of Lemma 4.2, this means that all jobs will be assigned in step 2. \square

In what follows, let Λ denote the total size of jobs of GoS 1 in the entire input.

Lemma 4.4 *If step 5 is applied for a new job j , and it holds that $p_j + p_j^{\max Y} \leq 1.25$, then the algorithm always manages to update W such that $\gamma_j \leq w_j \leq 0.75 \cdot p_j$ and $w_j < 0.5$, and after assigning j , the makespan will not exceed 1.25. If afterwards m_1 receives only jobs of GoS 1, then its load will remain at most 1.25.*

Proof Consider job j that is assigned in step 5. The assumption of the lemma is $p_j + p_j^{\max Y} \leq 1.25$, and the set W is computed by the algorithm. Since j is not assigned in an earlier step, and due to the assignment in step 5, we have $0.5 < p_j < 0.75$, $y_{j-1} < 0.75$, and $p_j + y_{j-1} > 1.25$. So $0.5 < y_{j-1} < 0.75$ holds as well. In particular, Y_{j-1} is non-empty, as it holds for every assignment in steps 4 and 5. Additionally, we have $\gamma_j = y_{j-1} + p_j - 1.25 < 0.75 + 0.75 - 1.25 = 0.25$. We first show that in all three options of case 5 for assigning j to the second machine, we have $\gamma_j \leq w_j \leq 0.75 \cdot p_j$ and $w_j < 0.5$.

Now, if $p_j^{\max Y} \geq \frac{y_{j-1}}{2}$ holds, the algorithm updates W to be $Y \setminus \{j^{\max Y}\}$, and in this case we get $w_j = y_{j-1} - p_j^{\max Y} \leq \frac{y_{j-1}}{2} < \frac{3}{8} < 0.75 \cdot p_j$. On the other hand $\gamma_j = y_{j-1} + p_j - 1.25 = (w_j + p_j^{\max Y}) + p_j - 1.25$, this yields $\gamma_j \leq w_j$ as $p_j^{\max Y} + p_j \leq 1.25$.

If $0.25 \leq p_j^{\max Y} < \frac{y_{j-1}}{2}$ holds, the algorithm updates W to contain only $\{j^{\max Y}\}$, i.e. $w_j = p_j^{\max Y}$, and in this case we get $\gamma_j < 0.25 \leq w_j < \frac{y_{j-1}}{2} < \frac{3}{8} < 0.75 \cdot p_j$.

Finally, if $p_j^{\max Y} < 0.25$, the algorithm sorts the jobs of Y by non-increasing size and updates W to be the minimum length prefix of the sorted list with total size at least 0.25. Since $p_j^{\max Y} < 0.25$ (so all jobs of Y_{j-1} are smaller than 0.25), while $y_{j-1} > 0.5$, the algorithm initially updates W to be of total size in $[0.25, 0.5)$. Since $0.5 < y_{j-1} < 0.75$, the total size of the jobs of the complement set is smaller than 0.5. Therefore, no matter whether W is the original one or the complement, both $w_j < 0.5$ and $y_{j-1} - w_j < 0.5$ will hold.

If $w_j \leq 0.75 \cdot p_j$, the set W is not modified further, and we are done also with respect to this property of W . Otherwise, the algorithm updates W to be its complement set with respect to Y . Since $y_{j-1} < 0.75 \leq \frac{3p_j}{2}$, for the partition of Y_{j-1} , at least one of the two subsets has total size at most $\frac{3p_j}{4}$, and therefore the total size of the complement set is smaller than $0.75 \cdot p_j$, and for the final and possibly modified set W , we get

$w_j \leq 0.75 \cdot p_j$. On the other hand $\gamma_j = y_{j-1} + p_j - 1.25 < (0.5 + w_j) + p_j - 1.25 = w_j + p_j - 0.75$. Since $p_j < 0.75$ we get $\gamma_j < w_j$.

To prove the last part, note that j will be assigned to machine m_2 , and recall that we have $\gamma_j \leq w_j \leq 0.75 \cdot p_j$ where the algorithm migrates W to machine m_1 . Thus, after assigning j we get $y_j = p_j + y_{j-1} - w_j \leq p_j + y_{j-1} - \gamma_j = 1.25$.

As for the load of the first machine, we have $y_{j-1} + p_j > 1.25$, and therefore the total size of jobs whose GoS is 1 is below 0.75. We have $z_{j-1} + y_{j-1} + p_j \leq 2 - \Lambda$. If the first machine only receives jobs of GoS 1 after j was assigned, its load is at most $\Lambda + z_{j-1} + w_j \leq 2 - (y_{j-1} + p_j) + w_j < 2 - 1.25 + 0.5 = 1.25$. \square

Lemma 4.5 *For every input, the algorithm enters step 5 at most once. Furthermore, if the algorithm enters step 5, all further jobs will be assigned in steps 2 and 3.*

Proof If the algorithm does not apply step 5, we are done. Otherwise, let j be the first job for which the algorithm applies step 5. Since previous steps were not applied for j , we have $0.5 < p_j < 0.75$ and $y_{j-1} + p_j > 1.25$. If $p_j + p_j^{\max Y} > 1.25$, it holds that $p_j^{\max Y} > 0.5$. Therefore, since there are already two jobs of sizes above 0.5, due to the optimal offline makespan of 1, any job that arrives after j has a processing time of at most 0.5. This means the algorithm will assign all future jobs in steps 2 and 3 (see Lemma 4.2). If $p_j + p_j^{\max Y} \leq 1.25$, then by Lemma 4.4, j will be assigned to machine m_2 , and $w_j < 0.5$, so we get $y_j = y_{j-1} + p_j - w_j \geq 1.25 - 0.5 = 0.75$. Therefore, the algorithm will assign all further jobs by step 2. \square

Corollary 4.6 *For every input, the algorithm will not enter both steps 4 and 5.*

Proof If the algorithm assigns j in step 4, then by Lemma 4.3 it assigns all further jobs in steps 2 and 3. If the algorithm assigns j in step 5, the situation is similar, by Lemma 4.5. Thus, after one of steps 4 and 5 is applied, none of these steps is applied again. \square

Lemma 4.7 *After assigning a job, j , in step 4, the makespan is at most 1.25. The load of m_1 will not exceed 1.25 even if it receives additional jobs of GoS 1 in the future.*

Proof By Corollary 4.6, the algorithm never entered step 4 or 5 before j is presented. Therefore, machine m_1 has no jobs of GoS 2. This holds since jobs of GoS 2 are not assigned to m_1 in step 3, and for step 2, once such a job is assigned to m_1 , all future jobs (for both hierarchies) will be assigned by the same step to the same machine, and the algorithm would not have reached step 4.

Since previous steps were not applied but step 4 is applied, we have $p_j \geq 0.75$, $y_{j-1} < 0.75$, and $y_{j-1} + p_j > 1.25$.

Now, if j is assigned to machine m_2 , we have a subset W such that $y_{j-1} + p_j - 1.25 = \gamma_j \leq w_j \leq 0.75 \cdot p_j$ that the algorithm migrated it to machine m_1 , so we get $0.75 \leq p_j \leq y_j = y_{j-1} + p_j - w_j \leq 1.25$, and $x_j + z_j \leq 2 - y_j \leq 1.25$.

If j is assigned to machine m_1 , it holds that $y_j = y_{j-1} < 0.75$, and we have $y_{j-1} - w_j + p_j > 1.25$. Assume that W is empty. For an assignment in step 4, the set Y_{j-1} is non-empty, so already the first job in the sorted order was too large, and $p_j^{\max Y} > 0.75 \cdot p_j > \frac{1}{2}$ holds. Otherwise, by Lemma 4.1, Y_{j-1} contains a pair of jobs

whose total processing time is greater than $0.75 \cdot p_j$, and the total size of each of them together with j is above 1. In the case where W is empty, a single job plays the role of these two jobs.

In any optimal solution, one machine has j , and the other machine has jobs of GoS 2 with total size above $0.75 \cdot p_j$, and therefore each machine has jobs of total size above $0.75 \cdot p_j$ of GoS 2. This implies that the total size of jobs of GoS 1 is at most $1 - 0.75 \cdot p_j$, since only one machine may have such jobs, and thus $\Lambda \leq 1 - 0.75 \cdot p_j$. As long as machine 1 does not get other jobs of GoS 2, it has jobs of GoS 1 and the set $Z_j = \{j\}$, and its load is at most $p_j + \Lambda \leq 1 - 0.75 \cdot p_j + p_j \leq 1.25$. \square

Lemma 4.8 *In step 5, if $p_j + p_j^{\max Y} > 1.25$, then after assigning j the makespan is at most 1.25. If afterwards m_1 receives only jobs of GoS 1, then its load will remain at most 1.25.*

Proof Similarly to Lemma 4.7, by Corollary 4.6, the algorithm never entered step 4 or 5 before j , and machine m_1 has only jobs of GoS 1 (the set of such jobs may be empty). Since the largest job of Y_{j-1} and job j are assigned to different machines in any optimal solution (as their total size is above 1), the total size of jobs whose GoS is 1 is at most $\max\{1 - p_j, 1 - p_j^{\max Y}\}$, and this is an upper bound on Λ , which we used to denote the total size of jobs of GoS 1 in the entire input. Once again $y_j = y_{j-1} < 0.75$, so it remains to bound the load of the first machine, including all jobs of GoS 1.

We have $\Lambda + z_j = \Lambda + p_j$. If $\max\{1 - p_j, 1 - p_j^{\max Y}\} = 1 - p_j$, we get $\Lambda + z_j \leq 1$. Otherwise, $\Lambda + z_j \leq 1 - p_j^{\max Y} + p_j < 2 \cdot p_j - 0.25$. In this case it holds that $p_j \leq 0.75$, and therefore $\Lambda + z_j < 1.25$. \square

Theorem 4.9 *The competitive ratio for algorithm B is at most 1.25.*

Proof Since the optimal offline cost is 1, we show by induction that the makespan never exceeds 1.25. The base case is before any job was assigned. We assume that just before job j (for $j \geq 1$) is assigned, the makespan does not exceed 1.25. We already showed that the makespan will not exceed 1.25 after an assignment by step 4 or 5. If j is assigned in step 2 and $y_{j-1} \geq 0.75$, we have $y_j = y_{j-1} \leq 1.25$ by the induction hypothesis and $x_j + z_j \leq 2 - y_j \leq 1.25$. If j is assigned by step 3, the property is satisfied by induction and by the condition of this case. We are left with the case that j is assigned by step 2 while $y_{j-1} < 0.75$, in which case $g_j = 1$ holds. If $z_j = 0$, we get $x_j \leq 1$, since the total size of jobs of GoS 1 does not exceed 1, so we assume that $z_j > 0$. This means that one of steps 4, 5 was applied in the past. In all cases for steps 4 and 5 we saw that as long as m_1 only receives jobs of GoS 1, its load remains no larger than 1.25. Thus, we consider the case that it received a job of GoS 2 afterwards, which could only happen in step 2. At the time of assignment, the total size of jobs in Y was at least 0.75. When only steps 2 and 3 are applied, the total size of jobs of Y cannot decrease, and when a job whose GoS is 2 was assigned to m_1 , this means that the total size for Y was at least 0.75. Thus, $y_{j-1} \geq 0.75$, which contradicts the condition of the case. \square

Lemma 4.10 *The migration factor of the algorithm is at most $\frac{3}{4}$.*

Proof The only two steps where jobs may be migrated, step 4 and 5. In step 4, W is migrated, and it is selected such that its total size does not exceed $0.75 \cdot p_j$. For step 5, a similar property was proved in Lemma 4.4. \square

5 The case $\frac{1}{2} \leq M < \frac{3}{4}$

In this section, we prove a tight bound of $2 - M$ for the case $M \in [\frac{1}{2}, \frac{3}{4})$. In the end of this analysis, we will show a lower bound on the competitive ratio for any algorithm. The upper bound on the competitive ratio is shown by presenting two algorithms (C and D) defined over two different domains of the migration factor of this case. The idea behind both algorithms is similar to the previous algorithms, but still we present two distinct algorithms. The algorithms use different approaches of determining W in order to migrate jobs to m_1 , and maintain a load for m_2 that is not greater than $2 - M$. In the domain $\frac{1}{2} \leq M < \frac{2}{3}$, Algorithm C finds a subset W of Y , whose migration will fulfill the required properties. In the domain $\frac{2}{3} \leq M < \frac{3}{4}$, the Algorithm D finds a subset W of Y also, but the two algorithms differ in size of this subset, which is stemmed from the property that the migration factor is larger and the competitive ratio is smaller. We present the proof of the bounds of the competitive ratios, and the migration factors, after presenting them. We begin with Algorithm C with a migration factor in the interval $[\frac{1}{2}, \frac{2}{3})$:

5.1 An algorithm for the case $\frac{1}{2} \leq M < \frac{2}{3}$

Algorithm C

Let $X = \emptyset$, $Y = \emptyset$, $Z = \emptyset$, $W = \emptyset$;

Repeat until all jobs have been assigned:

1. Receive job j with p_j and g_j ;
2. If $g_j = 1$ holds or $y_{j-1} \geq M$ holds (or both), assign j to the first machine and update: $X \leftarrow X \cup \{j\}$, or $Z \leftarrow Z \cup \{j\}$, respectively.
return to step 1.
3. If $y_{j-1} + p_j \leq 2 - M$ holds, assign j to the second machine and update: $Y \leftarrow Y \cup \{j\}$,
return to step 1.
4. If $p_j^{\max Y} > M \cdot p_j$ holds, assign j to the first machine and update: $Z \leftarrow Z \cup \{j\}$.
return to step 1.
5. Sort the jobs in Y by non-increasing size, and update W to be the minimum length prefix of the sorted list with total size at least $p_j + y_{j-1} - (2 - M)$ (or let $W = Y$ if not such subset exists). Migrate all jobs of W to the first machine, and assign j to the second machine, and update: $Y \leftarrow (Y \setminus W) \cup \{j\}$ and $Z \leftarrow Z \cup W$.
return to step 1.

Examples for this algorithm are provided in “Appendix A.3”. The idea of the first three steps of Algorithm C is similar to the idea of its predecessors. It handles small jobs and ensures that the load of the second machine load is at most $2 - M$. In the last

steps, step 4 and step 5, Algorithm *C* relies on the fact that in each input there cannot be three jobs such that the total processing time of each pair of jobs (out of these three jobs) exceeds 1. Similarly to Algorithm *B*, we prove that Algorithm *C* may apply step 4 or 5, but not both, so such a step (one of these two steps) is applied at most once for every input. Then, we prove that if Algorithm *C* enters step 5, then it will always be able to find a subset W of Y such that $\gamma_j \leq w_j \leq M \cdot p_j$, where γ_j in the case $\frac{1}{2} \leq M < \frac{3}{4}$ is equal to $y_{j-1} + p_j - (2 - M)$. The value γ_j is again a lower bound on the total size that has to be migrated from the second machine to obtain $y_j \leq 2 - M$, which allows the algorithm to schedule j on the second machine.

Note that the if the algorithm reaches step 4 (and it applies this step or step 5), it holds that $y_{j-1} < M$ (since step 2 was not applied) and $y_{j-1} + p_j > 2 - M$ (since step 3 was not applied), which implies that $p_j > 2 - 2 \cdot M \geq \frac{2}{3}$ (by the condition of the case for M). In this case, we will have $M \cdot p_j \geq M \cdot (2 - 2M) > \frac{4}{9}$. The property $y_{j-1} + p_j > 2 - M$ implies that the set W is always selected such that its total size is at least $y_{j-1} + p_j - (2 - M)$. We also have $p_j^{\max Y} \leq M \cdot p_j$ if step 5 is applied. We will show precisely that W is selected appropriately.

Lemma 5.1 *In step 5, the algorithm always manages to update W such that $\gamma_j \leq w_j \leq M \cdot p_j$.*

Proof Suppose that the algorithm enters step 5. The selected set has total size at least $\gamma_j = p_j + y_{j-1} - (2 - M)$ since such a subset exists, and therefore $\gamma_j \leq w_j$ holds. It is left to show that $w_j \leq M \cdot p_j$ holds. By $p_j^{\max Y} \leq M \cdot p_j$, we can consider the following two cases:

1. $\gamma_j \leq p_j^{\max Y} \leq M \cdot p_j$.
2. $p_j^{\max Y} < \gamma_j$.

In the first case, W will consist of one job, which is the largest job in Y , i.e $\gamma_j \leq w_j = p_j^{\max Y} \leq M \cdot p_j$. In the second case $p_j^{\max Y} < \gamma_j$ holds, and we use the following properties.

1. $y_{j-1} > \gamma_j$, because $p_j + M - 2 \leq 1 + \frac{2}{3} - 2 = -\frac{1}{3}$ yields $\gamma_j < y_{j-1} + p_j + (2 - M) = y_{j-1}$.
2. $M + 2 < 4 - 2M$, because $M < \frac{2}{3}$.
3. $(2 - M) \cdot p_j + 2 \cdot y_{j-1} < M + 2$, because $y_{j-1} < M$, $p_j \leq 1$.

The concatenation of the last two properties gives $(2 - M) \cdot p_j + 2 \cdot y_{j-1} < 4 - 2M$. We get $(2 - M)p_j + 2y_{j-1} < 4 - 2M \Leftrightarrow 2(p_j + y_{j-1} - 2 + M) < Mp_j \Leftrightarrow 2\gamma_j < M \cdot p_j$. Every job scheduled on the second machine (any job of Y_{j-1}) has a size of less than γ_j , (because $p_j^{\max Y} < \gamma_j$). Therefore, a minimum length prefix of the sorted list with total size at least γ_j will have a total size of at most $2\gamma_j < M \cdot p_j$. □

Lemma 5.2 *For any job j satisfying $p_j \leq 2 - 2M$, j is assigned in step 2 or step 3. Once the property $y_j \geq M$ holds for some job j , all jobs arriving after j will be scheduled in step 2.*

Proof The proof is the same as the proof of Lemma 4.2, taking into account the different parameters.

If $g_j = 1$, j is assigned in step 2. If prior to the arrival of j it holds that $y_{j-1} \geq M$, it is also assigned in step 2. Thus, we are left with the case $g_j = 2$ and $y_{j-1} < M$. In this case, $y_{j-1} + p_j < 2 - M$, and therefore j is assigned in step 3. \square

Lemma 5.3 *For every input, the algorithm enters the union of steps 4 and 5 at most once.*

Proof If none of these steps is applied, we are done. Otherwise, let j be the first job for which step 3 is considered but not applied. Since steps 2 and 3 were not applied, it holds that $p_j > 2 - 2M$.

If the algorithm enters step 4, i.e. $p_j^{\max Y} > M \cdot p_j > M(2 - 2M)$. We have $(2 - 2M) + M(2 - 2M) > 1$ and $2(2 - 2M) > 1$, since $(2 - 2M) + M(2 - 2M) = 2(1 - M)(1 + M) = 2(1 - M^2) \geq 2(1 - 4/9) > 1$ and $M < \frac{2}{3}$. Therefore, later in the input no job with processing time more than $2 - 2M$ will arrive (because a machine that will get two of the three largest will have load above 1), so by Lemma 5.2, the algorithm will not reach step 4 or step 5.

If $p_j^{\max Y} \leq M \cdot p_j$, then j will be assigned in step 5 to machine m_2 , and in this case we get $y_j \geq p_j > 2 - 2M \geq M$. Therefore, by Lemma 5.2 all jobs that arrive after j will be assigned to the first machine by step 2. \square

Lemma 5.4 *If job j is assigned in step 4 or step 5, after assigning j , the makespan is at most $2 - M$. If m_1 will only get jobs of GoS 1 afterwards, its load will never exceed $2 - M$.*

Proof In this case j has GoS 2, and it holds that $y_{j-1} < M$ and $y_{j-1} + p_j > 2 - M$, since the conditions of steps 2 and 3 do not hold. By Lemma 5.3, the algorithm never entered step 4 or 5 before j . Therefore, machine m_1 has only jobs of GoS 1 (or it may be empty), by Lemma 5.2. If $p_j^{\max Y} > M \cdot p_j$, we have two jobs whose total size is above 1 which must be assigned to different machines in any optimal solution (in the proof of Lemma 5.3, we saw that $p_j + M \cdot p_j > 1$). Thus, the total size of jobs with GoS 1 (recall that it is denoted by Λ) is at most $\max\{1 - p_j, 1 - p_j^{\max Y}\}$. Step 4 is applied, and since $y_j = y_{j-1} < M < 2 - M$, we analyze m_1 . We have $\Lambda + z_j = \Lambda + p_j$. If $p_j \leq p_j^{\max Y}$, we get $\Lambda + p_j \leq 1$. Otherwise, $\Lambda + p_j \leq 1 - p_j^{\max Y} + p_j < 1 + p_j - M \cdot p_j = 1 + (1 - M)p_j \leq 2 - M$, since $p_j \leq 1$.

If $p_j^{\max Y} \leq M \cdot p_j$, step 5 is applied. The algorithm will assign j to machine m_2 , and by Lemma 5.1 it will migrate a subset W of size at least γ_j , so we get: $y_j = y_{j-1} - \gamma_j + p_j \leq 2 - M$, and $x_j + z_j \leq 2 - y_j \leq 2 - p_j < 2 - (2 - 2M) = 2M < 2 - M$. Since steps 4 and 5 will not be applied again, the set Y will always have job j , so the load of m_1 will never exceed $2 - p_j < 2 - M$ (no matter which jobs it will receive). \square

Theorem 5.5 *The competitive ratio for algorithm C is at most $2 - M$.*

Proof We use induction again. We saw that an assignment in step 4 or in step 5 is valid in terms of the competitive ratio. So is an assignment in step 3, and an assignment in

step 2 due to a load of M or more for m_2 . Once again we are left with the case where a job j with $g_j = 1$ is assigned to m_1 . If steps 4 and 5 were never applied, and m_1 has a job of GoS 2, this means that the load of m_2 is at least M and it never decreases. If one of steps 4 and 5 was applied, but all jobs assigned to m_1 later have GoS 1, the load will not exceed $2 - M$. Otherwise, the situation is similar to the case that none of these steps is applied, since a job of GoS 2 is assigned after m_2 already has load of M or more, so m_1 will have a load of at most $2 - M$. \square

Lemma 5.6 *The migration factor of the algorithm is at most M .*

Proof The only step where jobs may be migrated is step 5. By Lemma 5.1, the algorithm always manages to update W such that $w_j \leq M \cdot p_j$, as required. \square

5.2 An algorithm for the case $\frac{2}{3} \leq M < \frac{3}{4}$

We consider Algorithm D , whose migration factor is in the interval $[\frac{2}{3}, \frac{3}{4})$. The structure of Algorithm D is based on principles similar to the previous algorithms. The algorithm tries to load machine m_2 until its load reaches the interval $[M, 2 - M]$, while it keeps the load on machine m_1 relatively small, that is, not exceeding $2 - M$. As in the last two algorithms, the migration is performed in this algorithm only from machine m_2 to machine m_1 . The manner of selecting the subset W is similar to its predecessors, but the selection analysis is more complex. In analyzing the selection of subset W in Algorithm D , we do not only consider the largest job in Y , but also the second largest job. Now, we present the algorithm and analyze it.

Algorithm D

Let $X = \emptyset$, $Y = \emptyset$, $Z = \emptyset$;

Repeat until all jobs have been assigned:

1. Receive job j with p_j and g_j ;
2. If $g_j = 1$ holds or $y_{j-1} \geq M$ holds (or both), assign j to the first machine and update: $X \leftarrow X \cup \{j\}$, or $Z \leftarrow Z \cup \{j\}$, respectively.
return to step 1.
3. If $y_{j-1} + p_j \leq 2 - M$ holds, assign j to the second machine and update: $Y \leftarrow Y \cup \{j\}$,
return to step 1.
4. If $p_j \geq M$ holds, sort the jobs in Y by non-increasing size, and update W to be the maximum length prefix of the sorted list with total size at most $M \cdot p_j$ (which may be empty).
 - 4.1. If $y_{j-1} - w_j + p_j > 2 - M$ holds, assign j to the first machine and update: $Z \leftarrow Z \cup \{j\}$.
 - 4.2. Otherwise, migrate the jobs of W to the first machine, assign j to the second machine and update: $Y \leftarrow (Y \setminus W) \cup \{j\}$ and $Z \leftarrow Z \cup W$.
 return to step 1.
5. Sort the jobs of Y by non-increasing size and update W to be the minimum length prefix of the sorted list with total size at least $\frac{M}{3}$, or the entire set if no such prefix exists.

- 5.1. If $w_j > \min\{\frac{2M}{3}, M \cdot p_j\}$ holds, update $W \leftarrow Y \setminus W$.
- 5.2. If $y_{j-1} - w_j + p_j > 2 - M$ holds, assign j to the first machine and update: $Z \leftarrow Z \cup \{j\}$.
- 5.3. Otherwise, migrate the jobs of W to the first machine, assign j to the second machine and update: $Y \leftarrow (Y \setminus W) \cup \{j\}$ and $Z \leftarrow Z \cup W$.

return to step 1.

Examples for this algorithm are provided in ‘‘Appendix A.4’’. First, we analyze step 4 of Algorithm *D*, and the first job j assigned by this step, if it exists. By the condition of assignment by step 4, $p_j \geq M$ holds. We divide the analysis into two cases:

1. Job j is assigned to machine m_2 .
2. Job j is assigned to machine m_1 .

In both cases we prove that the makespan is at most $2 - M$. We start with the first case.

Lemma 5.7 *In step 4, if the algorithm assigns j to the second machine, then the final load of the two machines will be at most $2 - M$.*

Proof By the condition of assignment of j to m_2 , we have $y_j = y_{j-1} - w_j + p_j \leq 2 - M$. Due to the size of j , we have $y_j \geq p_j \geq M$. From this moment onwards, the new jobs will be scheduled on the first machine by step 2, so the final load of machine m_2 will be at least M and at most $2 - M$, and the final load of machine m_1 will be at most $2 - M$. □

Lemma 5.8 *If j is assigned to m_1 in step 4, it holds that $|W| \leq 1$ and $|W| < |Y|$.*

Proof If $W = Y$, we have $w_j = y_{j-1}$ and $y_{j-1} - w_j + p_j = p_j \leq 1 \leq 2 - M$. Thus, in the case $W = Y$, j is assigned to m_2 .

If $W \geq 2$, while $|Y| > |W|$, letting $k = |W|$ (where $k \geq 2$), we have the following. The first job of Y (in the sorted order) that is not in W is not larger than any job of W , so $w_j \geq k \cdot x$, where x is the size of that job. Since $w_j + x > M \cdot p_j$ (as the next job was not included in W) and $x \leq \frac{w_j}{k}$, we have $\frac{k+1}{k} \cdot w_j > M \cdot p_j$ and by $k \geq 2$, we get $w_j > \frac{2}{3} \cdot M \cdot p_j$.

On the other hand, if we show that $w_j \geq 2M - 2 + p_j$, we will have $y_{j-1} - w_j + p_j \leq y_{j-1} - 2M + 2 \leq 2 - M$, since $y_{j-1} < M$. This would imply that j is assigned to m_2 . It is sufficient to show that $\frac{2}{3} \cdot M \cdot p_j \geq 2M - 2 + p_j$, and by rearranging, it is sufficient to show that $(1 - \frac{2M}{3}) \cdot p_j \leq 2 - 2M$ holds. Since $p_j \leq 1$ and $1 - \frac{2M}{3} > \frac{1}{2} > 0$ for $M < \frac{3}{4}$, it is sufficient to show that $1 - \frac{2M}{3} \leq 2 - 2M$, or equivalently, $\frac{4M}{3} \leq 1$, which holds since $M \leq \frac{3}{4}$. □

Recall that γ_j is a lower bound on the total size of migrated jobs, and therefore $\gamma_j = y_{j-1} + p_j - (2 - M)$ in the current case.

Lemma 5.9 *Assume that j is assigned in step 4. If it holds that $p_j^{\max Y} < \gamma_j$, and $p_j^{\max Y} + p_j^{\max Y,2} > M \cdot p_j$, then $p_j + p_j^{\max Y,2} > 1$ holds (and $p_j + p_j^{\max Y} > 1$ holds as well). Moreover, if $p_j^{\max Y} > M \cdot p_j$, then $p_j + p_j^{\max Y} > 1$ holds. In both situations, the total size of all jobs of GoS 1 is at most $1 - M \cdot p_j$.*

Proof We start with the first property. Let us assume (by contradiction) that the conditions hold but $p_j + p_j^{\max Y, 2} \leq 1$. We get (by $y_{j-1} < M$ which holds since step 2 was not applied for j):

$$M \cdot p_j < p_j^{\max Y} + p_j^{\max Y, 2} < \gamma_j + 1 - p_j = y_{j-1} + M - 1 < 2M - 1$$

i.e. $M \cdot p_j < 2M - 1$. Therefore $M^2 < 2M - 1$, which holds since $p_j \geq M$ (since j is assigned in step 4). This is a contradiction, because $M^2 - 2M + 1 = (M - 1)^2 \geq 0$. Thus, we find that $p_j + p_j^{\max Y, 2} > 1$ holds, and it also holds that $p_j + p_j^{\max Y} > 1$, because $p_j^{\max Y} \geq p_j^{\max Y, 2}$.

If we have $p_j^{\max Y} > M \cdot p_j$, we get $p_j + p_j^{\max Y} > (M + 1) \cdot p_j \geq M(M + 1) \geq \frac{2}{3} \cdot \frac{5}{3} > 1$.

In both cases, there is a job or a pair of jobs that cannot be assigned to the machine of j in any optimal solution. In the first case those are the two largest jobs of Y , and in the second case it is the largest job of Y . If j is assigned to m_1 in an optimal solution, the total size of other jobs of m_1 is at most $1 - p_j \leq 1 - M \cdot p_j$ since $M < 1$, and there are no other jobs whose GoS is 1. Otherwise, the upper bound on jobs of GoS 1 is $1 - M \cdot p_j$ due to the size of the largest job or two largest jobs of Y that cannot be assigned to m_2 in this case (where in the first situation, this is based on a pair of jobs and in the second situation it is based on one job). \square

Lemma 5.10 *In step 4, if the algorithm assigns j to the first machine, then the loads of the two machines immediately after the assignment of j will not exceed $2 - M$, and if m_1 will only get jobs of GoS 1 afterwards, its final load after the assignment of all jobs will be at most $2 - M$. Moreover, further jobs will assigned by steps 2 and 3, and the final makespan will not exceed $2 - M$.*

Proof By Lemma 5.8, we have $|W| \leq 1$, and Y has at least one other job. If W is empty, this means that already $p_j^{\max Y} > M \cdot p_j$ holds. In this case, the load of m_1 is at most the total size of jobs of GoS 1 plus p_j , which is at most $1 - M \cdot p_j + p_j$ by Lemma 5.9. We have $1 - M \cdot p_j + p_j = 1 + p_j(1 - M) \leq 2 - M$ by $p_j \leq 1$. The upper bound on the load of m_1 is valid not only after j is assigned to it but also as long as this machine receives only jobs of GoS 1.

If W consists of one job, then $p_j^{\max Y, 2} + p_j^{\max Y} > M \cdot p_j$ holds, and due to the assignment to m_1 , it holds that $p_j^{\max Y} = w_j < \gamma_j$. By Lemma 5.9, the analysis for m_1 is as in the previous case (since the two largest jobs of Y act as a single job). This completes the analysis for m_1 , and the load of m_2 is unchanged in the assignment of j , and it remains below $M < 1$.

Finally, we consider the loads of the machines after additional jobs arrive. Since a job of size at least M was assigned to m_1 (job j), all further jobs of GoS 2 can be assigned in step 3 if they are not assigned in step 2, and such an assignment will not increase the loads above $2 - M$. An assignment of a job of GoS 2 in step 2 means that the load of m_2 is already at least M , so after such an assignment is performed, all further jobs are assigned in step 2, keeping the load of m_1 not larger than $2 - M$ (and keeping the load of m_2 unchanged). \square

For the analysis of an assignment of a job j in step 5, the case where in particular it holds that $p_j < M$, we will first prove some auxiliary lemmas. Note that if step 5 is applied (and the previous steps were not), we have $p_j < M$, and $y_{j-1} + p_j > 2 - M$, which implies $y_{j-1} > 2 - 2M > \frac{2M}{3}$ and $y_{j-1} > 2 - 2M > \frac{1}{2}$ by $M < \frac{3}{4}$. The set W indeed has a total size above $\frac{M}{3}$, since $y_{j-1} > \frac{M}{3}$.

Lemma 5.11 *In step 5, given a set of jobs which is a subset of $\{1, 2, \dots, j-1\}$, whose total size is at most $\frac{M}{2}$, it is possible to migrate these jobs.*

Proof Since j is not assigned in steps 2 and 3, we have $p_j > 2 - 2M$. The proof follows immediately from the inequality: $p_j > 2 - 2M > \frac{1}{2}$, so $M \cdot p_j \geq \frac{M}{2}$. \square

Lemma 5.12 *In step 5, $y_{j-1} > \frac{2M}{3}$ and $p_j > \frac{2M}{3}$ hold.*

Proof In this case $y_{j-1} < M$ and $p_j < M$ hold since previous steps were not applied. If one of them has value of at most $\frac{2M}{3}$, we get $y_{j-1} + p_j < \frac{2M}{3} + M = \frac{5M}{3} < \frac{5}{3} \cdot \frac{3}{4} = 1.25$, which is a contradiction, because the algorithm considered step 3 for j and did not apply it, i.e. $y_{j-1} + p_j > 2 - M > 1.25$. \square

Note that for step 5 we found two constraints for p_j which are $p_j > \frac{2M}{3}$ and $p_j > 2 - 2M$. The second one is stronger than the first one, since $M < \frac{3}{4}$.

Lemma 5.13 *In step 5, if the algorithm migrates jobs from m_2 , and schedules j on m_2 , and we have $\frac{M}{3} \leq y_j - p_j \leq \frac{2M}{3}$, then $M \leq y_j \leq 2 - M$.*

Proof We will use the following properties. We have $2 - 2M < p_j < M$ since previous steps were not applied, and $M < \frac{3}{4}$, which implies $\frac{8M}{3} < 2$. We test the result of such a possible migration. It holds that $p_j + \frac{2M}{3} \leq 2 - M$, since $p_j \leq M \leq \frac{3}{4}$. On the other hand $\frac{M}{3} + p_j > \frac{M}{3} + (2 - 2M) > M$. Thus, the migration is applied, $y_j \leq p_j + \frac{2M}{3} \leq 2 - M$ and $y_j \geq \frac{M}{3} + p_j > M$. \square

Lemma 5.14 *In step 5, if the algorithm migrates jobs from m_2 , where the total size of these jobs is in the interval $[\frac{M}{3}, \frac{2M}{3}]$, and it schedules j on m_2 , then as a result we get $M \leq y_j \leq 2 - M$.*

Proof We will use the following properties. We have $y_{j-1}, p_j < M$ and $p_j + y_{j-1} > 2 - M$, since previous steps were not applied, and $M < \frac{3}{4}$, which implies $\frac{8M}{3} < 2$. If the algorithm tries to migrate a total size which is no less than $\frac{M}{3}$, then we get $p_j + y_{j-1} - \frac{M}{3} \leq 2M - \frac{M}{3} = \frac{5M}{3} < 2 - M$. If the algorithm tries to migrate a total size of jobs which is no more than $\frac{2M}{3}$, then we get $p_j + y_{j-1} - \frac{2M}{3} \geq 2 - M - \frac{2M}{3} > M$. Thus, the migration is applied, $y_j \leq p_j + y_{j-1} - \frac{M}{3} < 2 - M$, and $y_j \geq p_j + y_{j-1} - \frac{2M}{3} > M$. \square

Lemma 5.15 *In step 5, if the algorithm assigns j to machine m_2 , then the final makespan will be at most $2 - M$, and the migration factor is not violated.*

Proof In this case the algorithm checks if migration of W would violate the migration factor, and replaces it with $Y \setminus W$ if it would. By Lemma 5.11, and since $y_{j-1} < M$, at least one of these sets can be migrated without violating the migration factor, since the total size of the set with the smallest total size out of W and $Y \setminus W$ is below $\frac{M}{2}$. Thus, if the original W cannot be migrated, its complement can be migrated with respect to the migration factor. Recall that $y_{j-1} > \frac{2M}{3}$ by Lemma 5.12, and the set W is first defined to have a total size of at least $\frac{M}{3}$. Consider the following cases:

Case 1: $p_j^{\max Y} \leq \frac{2M}{3}$. In this case, the algorithm initially updates W such that $\frac{M}{3} \leq w_j \leq \frac{2M}{3}$ (no matter whether there is migration or not). The lower bound on w_j is based on the total size of Y and on the action of the algorithm. For the upper bound, we find that if the largest job of Y is of size at least $\frac{M}{3}$, then W initially consists of this job. Otherwise all jobs have sizes below $\frac{M}{3}$, and when the last job is added, the total size is below $\frac{M}{3}$. Thus, the tested condition for w_j is only a comparison to $M \cdot p_j$. Recall that it is assumed in this lemma that there is migration, so the two options are migration of the initially defined W or migration of $Y \setminus W$.

If $w_j \leq M \cdot p_j$, the algorithm migrates W to m_1 and schedules j on m_2 , so by Lemma 5.14 we get $M \leq y_j \leq 2 - M$. Otherwise, the algorithm updates W to be $Y \setminus W$, migrates it to m_1 and schedules j on m_2 . In the last case, we get $M \leq y_j \leq 2 - M$ by Lemma 5.13. This holds since $y_j - p_j$ is exactly the total size of jobs of W , before W is replaced with its complement.

Case 2: $p_j^{\max Y} > \frac{2M}{3}$, in this case, the algorithm first selects W to be the maximum size job in Y . After this, since the total size for W is too large, the algorithm updates W to be $Y \setminus W$, migrates it to m_1 (since we assume that there is migration), and schedules j on m_2 . Thus, we get $y_j = p_j + p_j^{\max Y} > \frac{2M}{3} + \frac{2M}{3} > M$ (see Lemma 5.12), and by the condition of assignment of j to m_2 , we have $y_j = y_{j-1} + p_j - w_j \leq 2 - M$.

In all cases we found that $M \leq y_j \leq 2 - M$ holds, so all jobs arriving after j will be scheduled on the first machine by step 2, and we get: $M \leq T_2 = y_j \leq 2 - M$, and $T_1 = 2 - T_2 \leq 2 - M$. \square

Lemma 5.16 *In step 5, if the algorithm assigns j to machine m_1 , then $p_j^{\max Y} + p_j > 2 - M$.*

Proof The analysis is related to the proof of Lemma 5.15, but here we analyze the case where j is assigned to m_1 , which means that $y_{j-1} - w_j + p_j > 2 - M$ holds.

In the case where $p_j^{\max Y} \leq \frac{2M}{3}$, we saw in the proof of Lemma 5.15 that initially W is chosen such that its total size is in $[\frac{M}{3}, \frac{2M}{3}]$. No matter whether W is modified or not, by Lemma 5.13 and Lemma 5.14, the situation $y_{j-1} - w_j + p_j > 2 - M$ will not be encountered.

Thus, we deal with the case $p_j^{\max Y} > \frac{2M}{3}$. In this case W first contains the largest job of Y , and then it is modified. Thus, $w_j = y_{j-1} - p_j^{\max Y}$. An assignment to m_1 therefore means that $p_j^{\max Y} + p_j > 2 - M$. \square

Lemma 5.17 *In step 5, if the algorithm assigns j to machine m_1 , then the final makespan will be at most $2 - M$.*

Proof It holds that $y_{j-1} + p_j > 2 - M$, since the condition of step 3 does not hold. Therefore, since the total size of jobs does not exceed 2, every job k that arrives after j has $p_k < M$, and thus the algorithm will not apply step 4 for any job $k > j$. It still might apply step 5 again, which we exclude next.

By Lemma 5.16, $p_j^{\max Y} + p_j > 2 - M$ holds. Since $p_j < M$ (since step 4 was not applied), we have $p_j^{\max Y} > 2 - 2M > \frac{1}{2}$. For every job k such that the algorithm applies step 5 for it, it also holds that $p_k > 2 - 2M > \frac{1}{2}$. If step 5 is applied with $k > j$, this means that there are three jobs of sizes above $\frac{1}{2}$, which is impossible for an optimal solution of makespan 1.

Therefore, all jobs $k > j$ are assigned in steps 2 and 3. We already have two jobs whose total size is above 1 which must be assigned to different machines in an optimal solution. Thus, the total size of jobs with GoS 1 is at most $\max\{1 - p_j, 1 - p_j^{\max Y}\}$. If machine m_1 will only receive jobs with GoS 1 until the end of the input, then $T_1 = x_n + p_j$. If $p_j \leq p_j^{\max Y}$, we get $x_j + p_j \leq 1$. Otherwise, $x_j + p_j \leq 1 - p_j^{\max Y} + p_j < 1 + p_j - (2 - M - p_j) < 3M - 1 \leq 2 - M$, since $p_j < M$ and $M \leq \frac{3}{4}$. If machine m_1 receives a job with GoS 2 later, then the load of machine m_2 is at least M , and it never decreases. In both cases we get $T_1 \leq 2 - M$ and $T_2 \leq 2 - M$. □

Theorem 5.18 *The competitive ratio for algorithm D is at most $2 - M$, and the migration factor is not larger than M .*

Proof We showed that the competitive ratio is valid if step 4 or 5 is reached. Assignment of a job of GoS 2 by step 3 does not allow m_2 to have a load above $2 - M$. If no jobs of GoS 2 are assigned to m_1 , this machine has a load of at most 1. Otherwise, m_2 has a load of at least M , and m_1 will not have a load above $2 - M$.

As for the migration factor, we observe that jobs are only migrated in steps 4 and 5, while in step 4 by definition we have that the total size of W does not exceed $M \cdot p_j$. For step 5, this was proved in Lemma 5.15. □

5.3 A lower bound on the competitive ratio of the case where $\frac{1}{2} \leq M < \frac{3}{4}$

Theorem 5.19 *The competitive ratio for every algorithm when $\frac{1}{2} \leq M < \frac{3}{4}$ is at least $2 - M$.*

Proof Let $\varepsilon > 0$ be a small constant such that $\varepsilon < 0.1$, such that $\frac{1}{\varepsilon}$ is an integer. The input starts with the first job $J_1 = (M + \varepsilon, 2)$ (where $M + \varepsilon < 1$). If the algorithm schedules the first job on the first machine, then there is a job of the form $(1, 1)$. It will not be possible to migrate the first job as $\frac{M+\varepsilon}{1} > M$. An optimal solution assigns the second job to m_1 and the first job is assigned to m_2 , and the makespan is 1. The algorithm will have a load of $1 + M + \varepsilon > 2 - M$ for m_1 (since $M \geq \frac{1}{2}$ and $\varepsilon > 0$).

If the algorithm schedules the first job on the second machine, then the second job is $J_2 = (1, 2)$, and it will not be possible to migrate the first job since $\frac{M+\varepsilon}{1} > M$. The optimal offline makespan is 1 which is achieved by assigning the first two jobs to different machines. If the algorithm schedules the second job on the second machine

(so both jobs are assigned to the same machine), we will get a competitive ratio of $1 + M + \varepsilon > 2 - M$ again.

If the algorithm schedules the second job on the first machine, then there is a third job $J_3 = (1 - M - \varepsilon, 1)$. This job cannot cause the migration of J_2 or of J_1 , since it is smaller (as $1 - M \leq \frac{1}{2}$). Thus, m_1 will have both J_2 and J_3 , and a load of $2 - M - \varepsilon$. An optimal solution assigns J_2 to m_2 and the other jobs to m_1 , for a makespan of 1. The competitive ratio $2 - M - \varepsilon$ tends to $2 - M$ by letting ε tend to zero. \square

6 The case $0 \leq M < \frac{1}{2}$ and comments

In this section we show that the case where $M < \frac{1}{2}$ is equivalent to the case without migration, by adapting the lower bound construction. Recall that an algorithm of competitive ratio 1.5 without migration is known (Park et al. 2006). This is the last remaining case, and we remind the reader that the results for all values of M were summarized in Fig. 1.

Theorem 6.1 *The competitive ratio for every algorithm when $M < \frac{1}{2}$ is at least 1.5.*

Proof The first job is defined by $J_1 = (0.5, 2)$. If the online algorithm schedules it on the first machine, then there is a job $J_2 = (1, 1)$. Since $M < \frac{1}{2}$, the first job cannot be migrated. An optimal solution schedules the first job on the second machine (and has a makespan of 1 due to the second job), while the algorithm schedules both on the first machine, and the competitive ratio is 1.5. Otherwise (if the online algorithm schedules the first job on the second machine), there are two additional jobs: $J_2 = (1, 2)$ and $J_3 = (0.5, 1)$, where the sizes are such that no job can cause the migration of another job. An optimal solution schedules the first and third jobs to the first machine, and the second job is scheduled to the second machine, and its makespan is 1. The algorithm must schedule the third job to the first machine. No matter whether the second job is assigned to the first machine or the second machine, its makespan is 1.5, and this is also the value of the competitive ratio. \square

6.1 A related variant

The following two semi-online models are related. The first one is the one where the optimal offline makespan is given in advance, and the other one is where the total size of jobs is provided in advance. It is known that an algorithm for the latter can be used as an algorithm for the former with the same competitive ratio (and therefore a lower bound on the competitive ratio for the former is a lower bound for the latter). There are specific problems where the results for the two models are similar and problems for which they are different (see for example Kellerer et al. 1997; Azar and Regev 2001, and Albers and Hellwig 2012; Kellerer et al. 2015 compared to Böhm et al. (2017b)).

We show that for the scheduling problems studied here, the two variants are very different in the sense that for M growing to infinity, the variant with known total size has a competitive ratio not tending to 1 (as the version studied here) but it is bounded away from 1.

Proposition 6.2 Any algorithm for the semi-online problem where the total processing time is given has competitive ratio of at least $\frac{\sqrt{33}-1}{4} \approx 1.18614$ for any migration factor M .

Proof First, we declare that the total processing time is 2.

The input starts with two large jobs of the form $(\theta, 2)$, where $\frac{1}{2} < \theta < \frac{2}{3}$ is a constant satisfying $4\theta^2 + \theta - 2 = 0$, that is, $\theta = \frac{\sqrt{33}-1}{8} \approx 0.59307$. The other jobs will be sufficiently small so that the large jobs cannot be migrated and their initial assignment by an online algorithm is final.

If both large jobs are assigned to m_2 , the remaining jobs are very small jobs (sand) of total size $2 - 2 \cdot \theta$ with GoS 2. An optimal solution can split the jobs evenly such that the makespan is 1 by assigning one large job to each machine with sand of total size $1 - \theta$. The algorithm has a makespan of at least 2θ .

Otherwise (if the first machine has at least one job), the sand of total size $2 - 2\theta$ has GoS 1. In this case, an optimal solution cannot have makespan 1, and by assigning both large jobs to m_2 , it has makespan 2θ , while the algorithm has to assign all the sand to m_1 and its makespan is at least $\theta + (2 - 2\theta) = 2 - \theta$.

The competitive ratio is at least $\min\{\frac{2\theta}{1}, \frac{2-\theta}{2\theta}\}$. By the equality $\frac{2-\theta}{2\theta} = 2\theta$ and using the approximate value of θ , we find that the competitive ratio of any algorithm and any migration factor is $\frac{\sqrt{33}-1}{4} \approx 1.18614$, even if the total job size is given in advance to the algorithm, and even if the migration factor is arbitrarily large (but fixed). \square

Declarations

Conflict of interest There are no conflicts of interest or competing interests for this work.

A Examples for the action of all algorithms

We provide a large number of examples for our algorithms. The examples cover a multitude of cases occurring in the execution.

A.1 Algorithm A, where $M \geq 2.5$

In this section, we provide three examples for Algorithm A. Two examples are for the migration factor $M = 3$. In both of them step 4 is applied once. In the first example, after applying step 4, only step 2 will be applied, and in the second example, both step 3 and step 2 are applied. The third example is for $M = 20$, and in that example, step 4 is performed several times.

A.1.1 Examples for Algorithm A with $M = 3$

In the two examples of this section, we show the action of Algorithm A with a migration factor of $M = 3$. For this value of M , we have $\mu = \frac{2}{9} \approx 0.2222$, the value used by

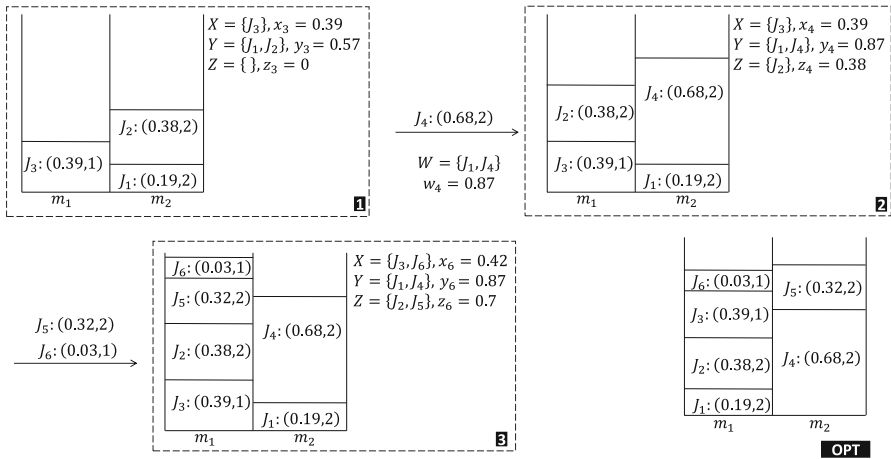


Fig. 4 The schedules produced by Algorithm A with migration factor $M = 3$ for I_1 (see Sect. A.1.1) after three, four, and six jobs, respectively have been presented, and an optimal solution (with makespan 1) for that input. The final makespan for the algorithm is 1.12

the algorithm in step 2 is $1 - \mu = \frac{7}{9} \approx 0.7778$, and the value used in step 3, which is also the competitive ratio for this algorithm, is $1 + \mu = \frac{11}{9} \approx 1.2222$.

Input I_1 is defined as follows:

$$J_1 = (0.19, 2), J_2 = (0.38, 2), J_3 = (0.39, 1), J_4 = (0.68, 2), \\ J_5 = (0.32, 2), J_6 = (0.03, 1).$$

The algorithm schedules the first two jobs on the second machine in step 3, and it schedules the third job on the first machine in step 2. We get $x_3 = 0.39, y_3 = 0.57$, and $z_3 = 0$. When the algorithm receives the fourth job, it reaches step 4 because $1.25 = y_3 + p_4 > 1 + \mu$ holds. In step 4, the algorithm updates W to be the set of the two jobs J_1 and J_4 , and it schedules these jobs on the second machine. The algorithm schedules the other jobs with GoS 2 that are not in W on the first machine (this set consists of a single job, which is J_2), and we get $x_4 = 0.39, y_4 = 0.87, z_4 = 0.38$. Afterwards, the algorithm schedules the J_5 on the first machine in step 2, because the load of the second machine is already not smaller than $1 - \mu$, i.e. $y_4 \geq \frac{7}{9}$. Job J_6 is assigned by step 2 as well, because both $g_6 = 1$ and $y_5 \geq \frac{7}{9}$ hold. Consequently, the final load of the first machine is 1.12, and the load of the second machine is 0.87. See Fig. 4 for an illustration of the process of execution of the algorithm, and an optimal solution.

In the second example, we use input I_2 , defined as follows:

$$J_1 = (0.6, 2), J_2 = (0.65, 2), J_3 = (0.4, 2), J_4 = (0.35, 1).$$

The algorithm schedules the first job on the second machine in step 3, and we get $x_1 = 0, y_1 = 0.6, z_1 = 0$. When the algorithm receives the second job, it reaches

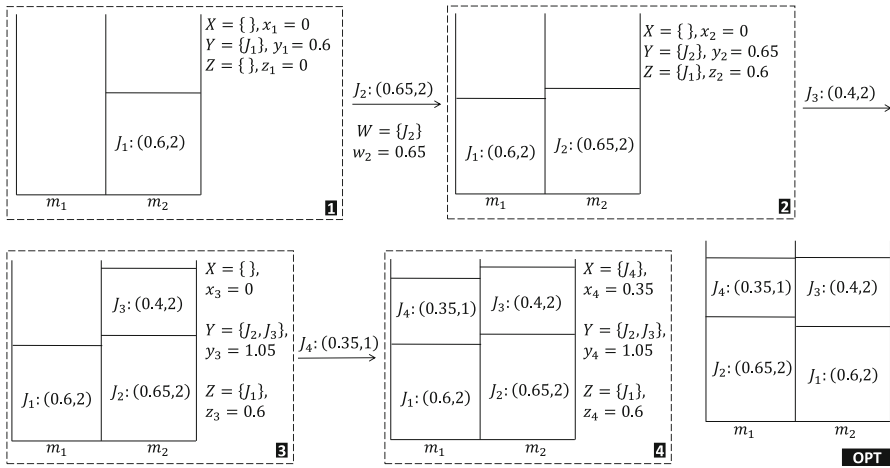


Fig. 5 The schedules produced by Algorithm A with migration factor $M = 3$ for I_2 (see Sect. A.1.1), and an optimal solution (with makespan 1) for that input. The final makespan for the algorithm is 1.05

step 4 because $1.25 = y_1 + p_2 > 1 + \mu$ holds. In step 4, the algorithm updates W to be the set $\{J_2\}$, schedules it on the second machine, and migrates the first job to the first machine. We get $x_2 = 0, y_2 = 0.65, z_2 = 0.6$. The algorithm schedules the third job on the second machine in step 3 because $y_2 + p_3 \leq 1 + \mu$ holds, and the fourth job is scheduled on the first machine in step 2 because $g_4 = 1$ (and additionally, at this time, it holds that $y_3 \geq 1 - \mu$). Consequently, the load of the first machine is 0.95, and the load of the second machine is 1.05. See Fig. 5 for an illustration of the process of execution of the algorithm, and an optimal solution.

A.1.2 An example for Algorithm A with $M = 20$

In this example, we show an input where step 4 is applied three times. We define input I_3 for Algorithm A with a migration factor of $M = 20$. For this value of M , we have $\mu = \frac{2}{43} \approx 0.04651$, the value used by the algorithm in step 2 is $1 - \mu = \frac{41}{43} \approx 0.95349$, and the value used in step 3, which is also the competitive ratio for this algorithm, is $1 + \mu = \frac{45}{43} \approx 1.04651$.

Input I_3 is defined as follows:

$$J_1 = (0.25, 2), J_2 = (0.28, 2), J_3 = (0.29, 2), J_4 = (0.1, 1),$$

$$J_5 = (0.26, 2), J_6 = (0.27, 2), J_7 = (0.49, 2), J_8 = (0.06, 1).$$

The algorithm schedules the first three jobs on the second machine in step 3, and it schedules J_4 on the first machine in step 2. We get $x_4 = 0.1, y_4 = 0.82, z_4 = 0$. When the algorithm receives the fifth job, it reaches step 4 because $1.08 = y_4 + p_5 > 1 + \mu$ holds. In step 4, the algorithm updates W to be $\{J_2, J_3, J_5\}$ (the subset of $Z \cup Y \cup \{J_5\}$ of maximum total size not exceeding 1), it migrates J_1 to the first machine, and it schedules W on the second machine. We get: $x_5 = 0.1, y_5 = 0.83, z_5 = 0.25$.

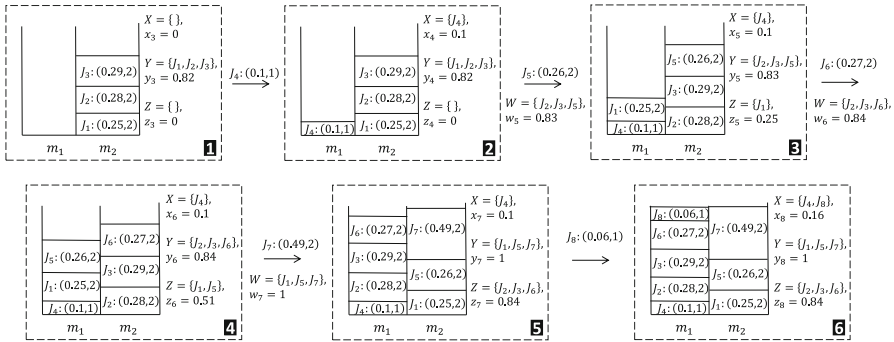


Fig. 6 The schedules produced by Algorithm A with migration factor $M = 20$, after three, four, five, six, seven, and eight jobs were presented, for input I_3 of Sect. A.1.2. There are migrations of J_1 (when J_5 arrives), of J_5 (when J_6 arrives), and of three jobs when J_7 arrives. The final makespan for the algorithm is 1, and the obtained solution is optimal

Afterwards, the algorithm repeats the same process for job J_6 (because $1.1 = y_5 + p_6 > 1 + \mu$ holds), it updates W to be $\{J_2, J_3, J_6\}$, it migrates J_5 to the first machine and schedules W on the second machine, and we get: $x_6 = 0.1$, $y_6 = 0.84$, $z_6 = 0.51$. The algorithm migrates J_1 and J_5 a second time when it receives job J_7 , since $1.33 = y_6 + p_7 > 1 + \mu$ holds. Here the algorithm updates W to be $\{J_1, J_5, J_7\}$, where $w_7 = 1$, it migrates all jobs in Y to the first machine, and it schedules W on the second machine. We get: $x_7 = 0.1$, $y_7 = 1$, $z_7 = 0.84$. Job J_8 is assigned to the first machine by step 2 because both $g_8 = 1$ holds (and additionally, $y_7 \geq 1 - \mu$), and we get: $x_8 = 0.16$, $y_8 = 1$, $z_8 = 0.84$. Consequently, the loads of both machines are equal to 1. Thus, in this specific case, the obtained solution is optimal. See Fig. 6 for an illustration of the process of execution of the algorithm.

A.2 Algorithm B, where $0.75 \leq M < 2.5$

We provide four examples for Algorithm B. In the first one, we show the action of this algorithm when it receives input I_1 from Sect. A.1.1. In the second one, we show the action of this algorithm in step 4 by using input I_4 (which we define). In the third example and in the fourth example, we show the action of the algorithm in step 5, when it receives inputs I_5 and I_6 , which we define, where the actual assignment is performed by step 5.1 and step 5.2, respectively, for these two inputs.

Recall that the competitive ratio for this algorithm is 1.25, and the values used in steps 2 and 3 are based on this value.

A.2.1 An example for Algorithm B, such that there is no migration

Algorithm B schedules input I_1 (see Sect. A.1.1) only using steps 2 and 3, i.e. it does not migrate any jobs. This holds because scheduling the fourth job on the second machine would not exceed the threshold of 1.25, but the load becomes greater than 0.75, i.e. $0.75 \leq y_4 = 1.25$, which keeps the machines balanced with respect to loads.

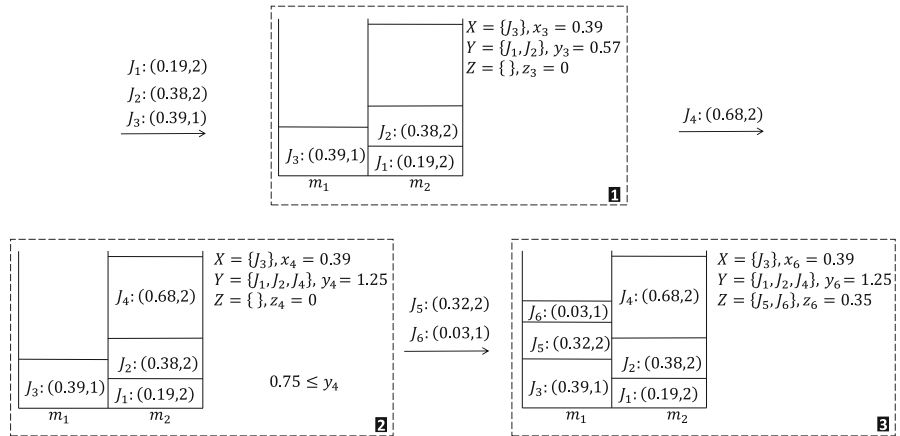


Fig. 7 The schedules produced by Algorithm B for input I_1 of Sect. A.1.1: (1) The algorithm schedules the first two jobs on the second machine in step 3, and the third job is scheduled on the first machine in step 2. (2) The algorithm schedules the fourth job on the second machine in step 3 and we get $0.75 \leq y_4$. (3) All other jobs will be scheduled on the first machine in step 2, and we have $c_B(I_1) = 1.25$ and $c^*(I_1) = 1$ (which holds by assigning J_4 and J_5 to the second machine, and the other jobs to the first machine)

Consequently, $x_6 = 0.42$, $y_6 = 1.25$, $z_6 = 0.32$, the load of the first machine is 0.74, and the load of the second machine is 1.25. See Fig. 7 for an illustration and additional details.

A.2.2 An example for Algorithm B, such that step 4 is applied

In this example, we define input I_4 and in particular, we exhibit the action of the algorithm in step 4 when the fifth job of I_4 arrives. The input is:

$$J_1 = (0.01, 2), J_2 = (0.36, 1), J_3 = (0.2, 2), J_4 = (0.36, 2),$$

$$J_5 = (0.79, 2), J_6 = (0.28, 2).$$

The first four jobs are scheduled in both steps 2 (job J_2) and 3 (the other three jobs), so we get: $x_4 = 0.36$, $y_4 = 0.57$, and $z_4 = 0$. In the process of assignment of the fifth job, the algorithm reaches step 4 because $y_4 + p_5 > 1.25$ and $p_5 \geq 0.75$ hold. We have $0.75 \cdot p_5 = 0.5925$. In step 4, the algorithm updates W be the set of the three jobs J_1, J_3 , and J_4 (whose total size is 0.57), it migrates W to the first machine, and it schedules J_5 on the second machine, because $y_4 - w_5 + p_5 = p_5 \leq 1.25$. We get: $x_5 = 0.36$, $y_5 = 0.79$, and $z_5 = 0.57$. The algorithm schedules the last job on the first machine in step 2 because $y_5 \geq 0.75$ holds, and it updates the variables as follows: $x_6 = 0.36$, $y_6 = 0.79$, $z_6 = 0.85$. Consequently, the load of the first machine is 1.21, and the load of the second machine is 0.79. See Fig. 8.

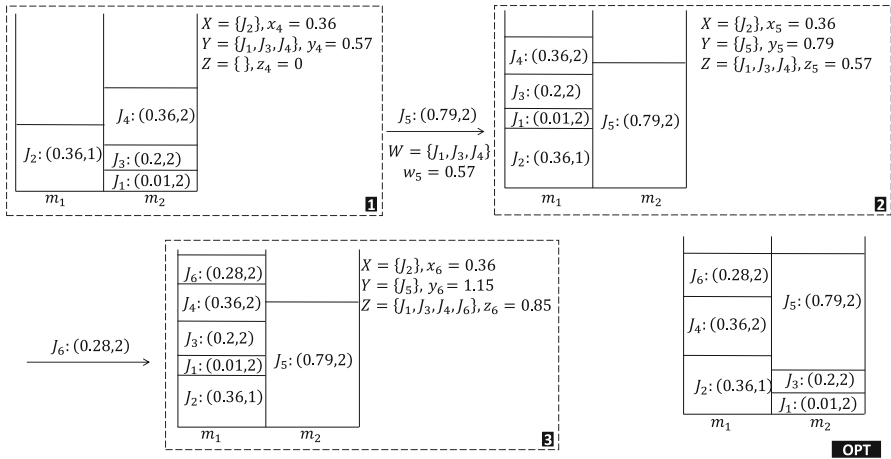


Fig. 8 The schedules produced by Algorithm B for input I_4 of Sect. A.2.2 (from the moment when the fourth job had arrived), and an optimal solution for that input. We have $c_B(I_4) = 1.21$ and $c^*(I_4) = 1$

A.2.3 An example for Algorithm B, such that step 5.1 is applied

In this example, we use a new input I_5 to show the action of Algorithm B. In particular, we exhibit the action of the algorithm in step 5 when the fourth job arrives. Input I_5 is:

$$J_1 = (0.2, 2), J_2 = (0.36, 2), J_3 = (0.07, 2), \\ J_4 = (0.73, 2), J_5 = (0.28, 1), J_6 = (0.36, 2).$$

The first three jobs are scheduled on the second machine in step 3, and we get: $x_3 = 0, y_3 = 0.63, z_3 = 0$. In the process of assignment of the fourth job, the algorithm reaches step 5, because $1.36 = y_3 + p_4 > 1.25$ and $0.73 = p_4 < 0.75$ hold. The largest job of the current set Y is J_2 , and $p_4^{\max Y} = p_2 = 0.36$, and together with the fourth job, it holds that $p_4 + p_4^{\max Y} = 1.09 < 1.25$, so W will be computed. Since $p_4^{\max Y} = p_2 = 0.36 \geq \frac{y_3}{2} = 0.315$ holds, the algorithm defines W in step 5.1, and it is defined to be the set of the two jobs J_1 and J_3 . The algorithm migrates W to the first machine and schedules J_4 on the second machine, and we get: $x_4 = 0, y_4 = 1.09$, and $z_4 = 0.27$. The algorithm schedules the last two jobs on the first machine in step 2 (for both jobs it holds that the load of the second machine is sufficiently large, and for the fifth job the GoS is 1), and we get: $x_6 = 0.28, y_6 = 1.09, z_6 = 0.63$. As a result, the load of the first machine is 0.91, the load of the second machine is 1.09. See Fig. 9.

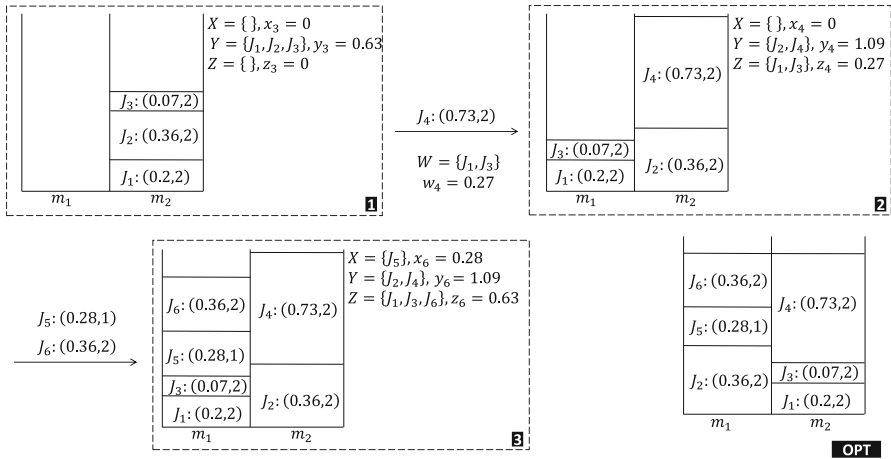


Fig. 9 The schedules produced by Algorithm B for input I_5 of Sect. A.2.3 (from the moment when the third job had arrived), and an optimal solution for that input. We have $c_B(I_5) = 1.09$ and $c^*(I_5) = 1$

A.2.4 An example for Algorithm B, such that step 5.2 is applied

In this example, we use a new input I_6 to show the action of Algorithm B. We exhibit the action of the algorithm in step 5 when the fourth job arrives. Input I_6 is:

$$J_1 = (0.17, 2), J_2 = (0.2, 2), J_3 = (0.26, 2), J_4 = (0.7, 2),$$

$$J_5 = (0.1, 1), J_6 = (0.57, 2).$$

The first three jobs are scheduled on the second machine in step 3, and we get: $x_3 = 0, y_3 = 0.63, z_3 = 0$. For the fourth job, the algorithm reaches step 5 because $1.33 = y_3 + p_4 > 1.25$ and $0.7 = p_4 < 0.75$ hold. The largest job of the current set Y is J_3 , and $p_4^{\max Y} = p_3 = 0.26$, and together with the fourth job, it holds that $p_4 + p_4^{\max Y} = 0.96 < 1.25$, so W will be computed. Since $p_4^{\max Y} = p_3 = 0.26$ while $y_3 = 0.63$ holds, the algorithm defines W in step 5.2. The set W is defined to be the set $\{J_3\}$. Specifically, step 5.2 is applied because $0.25 \leq p_4^{\max Y} < \frac{y_3}{2} = 0.315$ holds. The algorithm migrates W to the first machine, and it schedules J_4 on the second machine. We get: $x_4 = 0, y_4 = 1.07$, and $z_4 = 0.26$. Similarly to the previous examples, the algorithm schedules the last two jobs on the first machine in step 2, and it defines its new values as follows: $x_6 = 0.1, y_6 = 1.07, z_6 = 0.83$. As a result, the load of the first machine is 0.93, the load of the second machine is 1.07. See Fig. 10.

A.3 Algorithm C, where $0.5 \leq M < \frac{2}{3}$

Here, we provide three examples for Algorithm C. In the first example, we show the action of the algorithm when it receives input I_1 of Sect. A.1.1. In the second one, we show the action of the algorithm in step 4 by using a new input I_7 . In the third

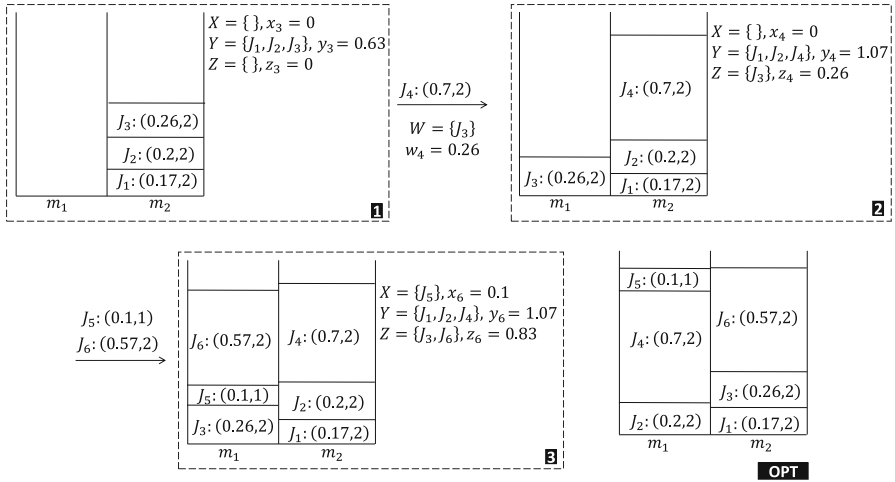


Fig. 10 The schedules produced by Algorithm *B* for input I_6 of Sect. A.2.4 (from the moment when the third job had arrived), and an optimal solution for that input. We have $c_B(I_6) = 1.07$ and $c^*(I_6) = 1$

example, we show the action of the algorithm in step 5 when it receives input I_8 , which we define.

In all three examples for this algorithm, we use $M = 0.6$, and thus the value used in step 2 is 0.6, and the value used in step 3 (and the competitive ratio) is 1.4.

A.3.1 An example Algorithm C, using I_1

Algorithm *C* with migration factor $M = 0.6$ schedules I_1 of Sect. A.1.1 only using the two steps 2 and 3, i.e. it does not migrate any jobs. This is because the load of the second machine after scheduling the fourth job on this machine does not exceed the upper bound (competitive ratio) $2 - M = 1.4$, and this keeps loads relatively balanced. Consequently, $x_6 = 0.42$, $y_6 = 1.25$, $z_6 = 0.32$, the load of the first machine is 0.74, and the load of the second machine is 1.25 (see the output of Fig. 7, which belongs to the run of another algorithm and the same input, though the output is identical).

A.3.2 An example for Algorithm C, such that step 4 is applied

In this example, we exhibit the action of the algorithm in step 4 when the third job of our new input arrives. Input I_7 is:

$$J_1 = (0.5, 2), J_2 = (0.09, 2), J_3 = (0.82, 2), J_4 = (0.18, 2), J_5 = (0.41, 1).$$

Recall that $M = 0.6$. The first two jobs are scheduled on the second machine in step 3, and we get: $x_2 = 0$, $y_2 = 0.59$, $z_2 = 0$. When the third job arrives, the algorithm reaches step 4 because $p_3^{\max Y} = p_1 = 0.5 > M \cdot p_3 = 0.492$ holds. In step 4, the algorithm schedules J_3 on the first machine, and we get: $x_3 = 0$, $y_3 = 0.59$, and $z_3 = 0.82$. The algorithm schedules the fourth job on the second machine in step 3

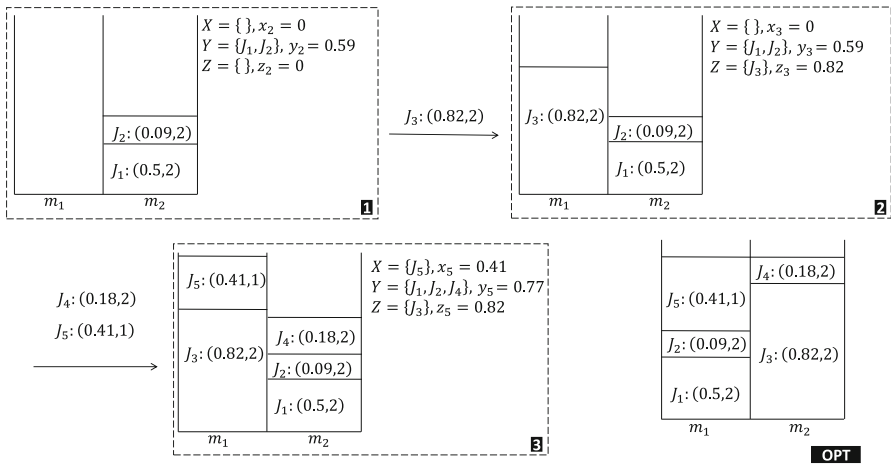


Fig. 11 The schedules produced by Algorithm C with migration factor $M = 0.6$ for input I_7 of Sect. A.3.2 (from the moment when the second job had arrived), and an optimal solution for that input. We have $c_C(I_7) = 1.23$ and $c^*(I_7) = 1$

because $0.77 = y_3 + p_4 \leq 1.4 = 2 - M$ holds, and the last job on the first machine in step 2 because $g_6 = 1$ holds. As a result, the load of the first machine is 1.23, the load of the second machine is 0.77. See Fig. 11.

A.3.3 An example for Algorithm C, such that step 5 is applied

In this example, we exhibit the action of the algorithm in step 5 when the fifth job of the input arrives. Input I_8 is:

$$J_1 = (0.35, 2), J_2 = (0.11, 2), J_3 = (0.13, 2), J_4 = (0.41, 1), J_5 = (0.82, 2), J_6 = (0.18, 2).$$

Recall that $M = 0.6$. The first three jobs are scheduled on the second machine in step 3, and the fourth job is scheduled on the first machine in step 2. At this time, we get: $x_4 = 0.41, y_4 = 0.59, z_4 = 0$. When the fifth job arrives, the algorithm reaches step 5, because $p_5^{\max Y} = p_1 = 0.35 \leq M \cdot p_5 = 0.492$. It updates W to be $\{J_1\}$ (the minimum length prefix with total size at least $p_5 + y_4 - (2 - M) = 0.01$), it migrates W to the first machine, and it schedules J_5 on the second machine. We get: $x_5 = 0.41, y_5 = 1.06, z_5 = 0.35$. The algorithm schedules the last job on the first machine in step 2 and updates: $x_6 = 0.41, y_6 = 1.06, z_6 = 0.35$. As a result, the load of the first machine is 0.94, the load of the second machine is 1.06. See Fig. 12.

A.4 Algorithm D, where $\frac{2}{3} \leq M < 0.75$

In this part we provide six examples for Algorithm D. In the first example we show the action of the algorithm when it receives input I_1 of Sect. A.1.1. In the second and

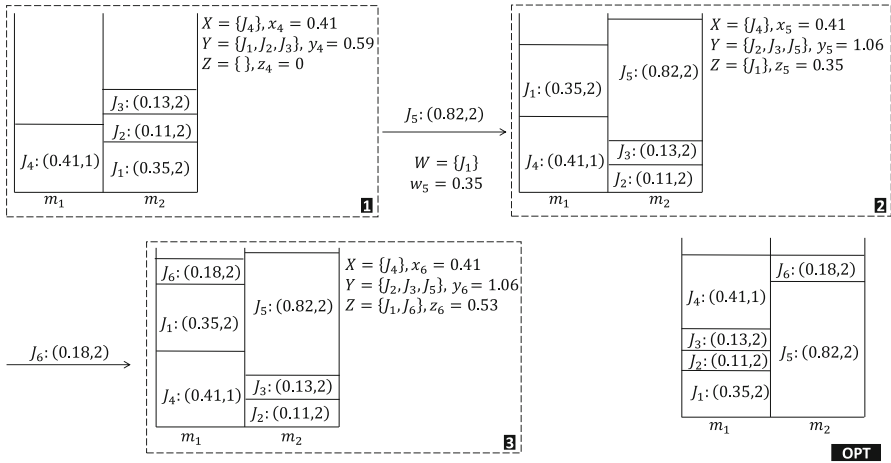


Fig. 12 The schedules produced by Algorithm C with migration factor $M = 0.6$ for input I_8 of Sect. A.3.3 (from the moment when the fourth job had arrived), and an optimal solution for that input. We have $c_C(I_8) = 1.06$ and $c^*(I_8) = 1$

the third examples we show the action of the algorithm in step 4, by using inputs I_9, I_{10} , which we define. In the last three examples, we show the action of the algorithm in step 5 for three inputs I_{11}, I_{12} and I_{13} , which we also define here.

In all six examples, we use $M = 0.7$. As a result, the value used by the algorithm in step 2 is $M = 0.7$, and the value used in step 3, which is also the competitive ratio for this algorithm, is $2 - M = 1.3$.

A.4.1 An example for Algorithm D, using I_1

Recall that $M = 0.7$. Algorithm D schedules input I_1 of Sect. A.1.1 only using the two steps 2 and 3, i.e. it does not migrate any jobs. This holds because after scheduling the fourth job on the second machine we get: $x_4 = 0.31, y_4 = 1.25, z_4 = 0$, i.e. the load of the second machine is between the two bounds $M = 0.7$ and $2 - M = 1.3$, so any job that arrives after J_4 will be scheduled on the first machine. Consequently, $x_6 = 0.42, y_6 = 1.25, z_6 = 0.32$, and the makespan is 1.25 (see Fig. 7 again).

A.4.2 An example for Algorithm D, such that step 4.2 is applied

In this example, we exhibit the action of the algorithm in step 4 where $y_{j-1} - w_j + p_j \leq 2 - M$ holds, which is tested when the fourth job in input I_9 arrives. Input I_9 is:

$$J_1 = (0.54, 2), J_2 = (0.14, 2), J_3 = (0.32, 1), J_4 = (0.78, 2), J_5 = (0.22, 2).$$

Recall that $M = 0.7$. The first two jobs are scheduled on the first machine in step 3, and the third job on the first machine in step 2. We get: $x_3 = 0.32, y_3 = 0.68$, and $z_3 = 0$. When the fourth job arrives, the algorithm reaches step 4, because $1.46 = y_3 + p_4 > 2 - M = 1.3$ and $p_4 = 0.78 \geq M = 0.7$ hold. In step 4, the

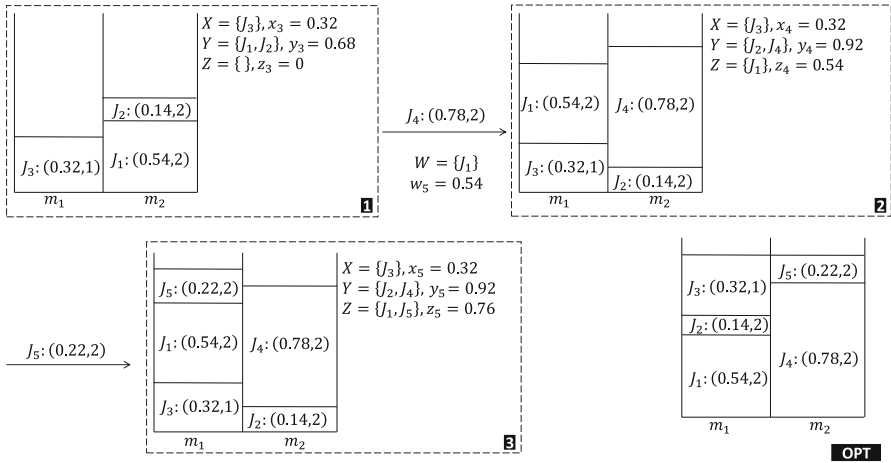


Fig. 13 The schedules produced by Algorithm *D* with migration factor $M = 0.7$ for input I_9 of Sect. A.4.2 (from the moment when the third job had arrived), and an optimal solution for that input. We have $c_D(I_9) = 1.08$ and $c^*(I_9) = 1$

algorithm updates W to be $\{J_1\}$ (the maximum length prefix with total size at most $M \cdot p_4 = 0.7 \cdot 0.78 = 0.546$, where $M \cdot p_4 \geq p_1 = 0.54$, but $p_1 + p_2 = 0.68 > 0.546$). We have $y_3 - w_4 + p_4 = 0.68 - 0.54 + 0.78 = 0.92 \leq 1.3 = 2 - M$. Thus, the algorithm continues to step 4.2, and it migrates W to the first machine. The algorithm schedules J_4 on the second machine, and we get: $x_4 = 0.32$, $y_4 = 0.92$, and $z_4 = 0.54$. The algorithm schedules the fifth job on the first machine in step 2 and it has $x_5 = 0.32$, $y_5 = 0.92$, and $z_5 = 0.76$. As a result, the load of the first machine is 1.08, the load of the second machine is 0.92. See Fig. 13.

A.4.3 An example for Algorithm *D*, such that step 4.1 is applied

In this example, we exhibit the action of the algorithm in step 4 where $y_{j-1} - w_j + p_j > 2 - M$ holds, which holds for the fourth job arrives. Input I_{10} which is used for this example is:

$$J_1 = (0.54, 2), J_2 = (0.14, 2), J_3 = (0.32, 1), J_4 = (0.71, 2), J_5 = (0.29, 2).$$

Recall that $M = 0.7$. The first two jobs are scheduled on the second machine in step 3, and the third job is scheduled on the first machine in step 2. At this time we have $x_3 = 0.32$, $y_3 = 0.68$, and $z_3 = 0$. When the fourth job arrives, the algorithm reaches step 4, because $p_4 = 0.71 \geq M = 0.7$ holds, and it did not apply earlier steps because $1.39 = y_3 + p_4 > 2 - M = 1.3$ holds while $y_3 < M = 0.7$. In step 4, the algorithm updates W to be an empty set (the maximum length prefix with total size at most $M \cdot p_4 = 0.7 \cdot 0.71 = 0.497 < p_1 = 0.54$), and we get: $y_3 - w_4 + p_4 = 0.68 - 0 + 0.71 = 1.39 > 1.3 = 2 - M$, so the algorithm schedules J_4 on the first machine, and we get: $x_4 = 0.32$, $y_4 = 0.68$, and $z_4 = 0.71$. The algorithm schedules the fifth job on the second machine in step 3 and it has

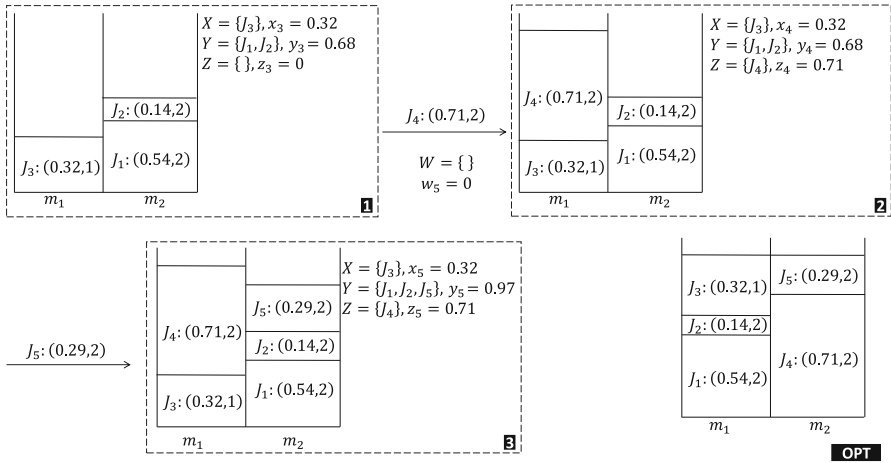


Fig. 14 The schedules produced by Algorithm D with migration factor $M = 0.7$ for input I_{10} of Sect. A.4.3 (from the moment when the third job had arrived), and an optimal solution for that input. We have $c_D(I_{10}) = 1.03$ and $c^*(I_{10}) = 1$

$x_5 = 0.32$, $y_5 = 0.97$, and $z_5 = 0.71$. As a result, the load of the first machine is 1.03, the load of the second machine is 0.97. See Fig. 14.

A.4.4 An example for Algorithm D , such that both steps 5.1 and 5.3 are applied

In this example, we exhibit the action of the algorithm in step 5 where $w_j > \min\{\frac{2M}{3}, M \cdot p_j\}$ and $y_{i-1} - w_j + p_j \leq 2 - M$ hold, which happens when the fourth input job arrives. Input I_{11} which is used for this example is:

$$J_1 = (0.54, 2), J_2 = (0.14, 2), J_3 = (0.32, 1), J_4 = (0.69, 2), J_5 = (0.31, 2).$$

Recall that $M = 0.7$. The first two jobs are scheduled on the second machine in step 3, and the third job is scheduled on the first machine in step 2. At this time, we get: $x_3 = 0.32$, $y_3 = 0.68$, $z_3 = 0$. When the fourth job arrives, the algorithm reaches step 5 because $1.37 = y_3 + p_4 > 2 - M = 1.3$ and $p_4 = 0.69 < M = 0.7$ hold. In step 5 the algorithm updates W twice. First, W is defined as $\{J_1\}$ (the minimum length prefix with total size at least $\frac{M}{3} = \frac{7}{30} \approx 0.2333 \leq p_1 = w_4 = 0.54$). At this time, we have: $w_4 > \min\{\frac{2M}{3}, M \cdot p_4\} = \min\{\frac{7}{15} \approx 0.46667, 0.483\}$. The algorithm updates W in step 5.1 again to be $Y \setminus W = \{J_2\}$. In step 5.3, the algorithm migrates W to the first machine, and it schedules J_4 on the first machine. Step 5.2 is not applied because $y_3 - w_4 + p_4 = 1.23 \leq 1.3 = 2 - M$ holds. We get: $x_4 = 0.32$, $y_4 = 1.23$, and $z_4 = 0.14$. The algorithm schedules the last job on the first machine in step 2 and we get: $x_5 = 0.32$, $y_5 = 1.23$, and $z_5 = 0.45$. As a result, the load of the first machine is 0.77, and the load of the second machine is 1.23. See Fig. 15.

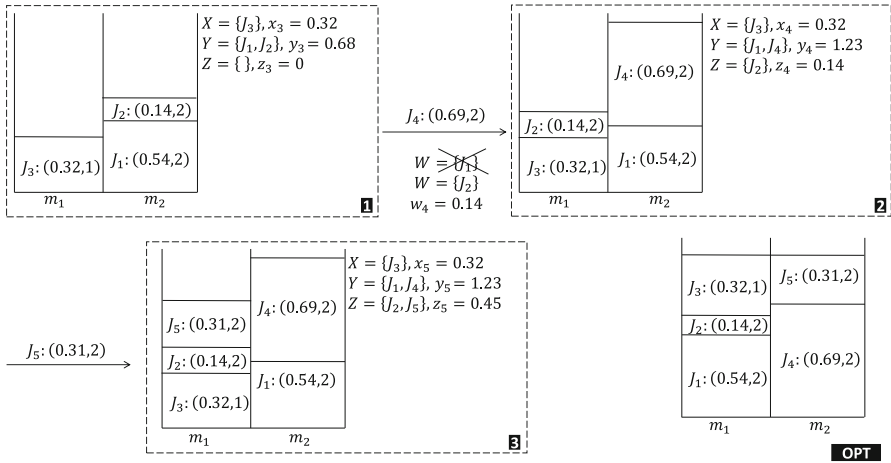


Fig. 15 The schedules produced by Algorithm D with migration factor $M = 0.7$ for input I_{11} of Sect. A.4.4 (from the moment when the third job had arrived), and an optimal solution for that input. We have $c_D(I_{11}) = 1.23$ and $c^*(I_{11}) = 1$

A.4.5 An example for Algorithm D , such that step 5.3 is applied

In this example, we exhibit the action of the algorithm in step 5 where $w_j \leq \min\{\frac{2M}{3}, M \cdot p_j\}$ and $y_{i-1} - w_j + p_j \leq 2 - M$ hold, and this happens when the second input job arrives. Input I_{12} which is used for this example is:

$$J_1 = (0.45, 2), J_2 = (0.24, 2), J_3 = (0.65, 2), J_4 = (0.31, 1), J_5 = (0.35, 2).$$

Recall that $M = 0.7$. The first two jobs are scheduled on the second machine in step 3, and we get: $x_2 = 0, y_2 = 0.69, z_2 = 0$. When the third job arrives, the algorithm reaches step 5 because $1.34 = y_2 + p_3 > 2 - M$ and $p_3 = 0.65 < M = 0.7$ hold. In step 5, the algorithm defines W to be $\{J_1\}$ (the minimum length prefix with total size at least $\frac{M}{3} = \frac{7}{30} \approx 0.2333 \leq p_1 = w_3 = 0.45$), here we get: $w_3 \leq \min\{\frac{2M}{3}, M \cdot p_3\} = \min\{\frac{7}{15}, 0.455\}$. In this case $0.89 = y_2 - w_3 + p_3 \leq 2 - M = 1.3$ holds, so the algorithm only applies step 5.3, and it migrates W to the first machine while scheduling J_3 on the second machine. We get: $x_3 = 0, y_3 = 0.89, z_3 = 0.45$. The algorithm schedules the last two jobs on the first machine in step 2 and it updates its variables as follows: $x_5 = 0.31, y_5 = 0.89, z_5 = 0.8$. As a result, the load of the first machine is 1.11, the load of the second machine is 0.89. See Fig. 16.

A.4.6 An example for Algorithm D , such that both steps 5.1 and 5.2 are applied

In this example, we exhibit the action of the algorithm in step 5 where $w_j > \min\{\frac{2M}{3}, M \cdot p_j\}$ and $y_{i-1} - w_j + p_j > 2 - M$ hold, and this happens when the second input job arrives. Input I_{13} which is used for this example is:

$$J_1 = (0.69, 2), J_2 = (0.62, 2), J_3 = (0.38, 2), J_4 = (0.31, 1).$$

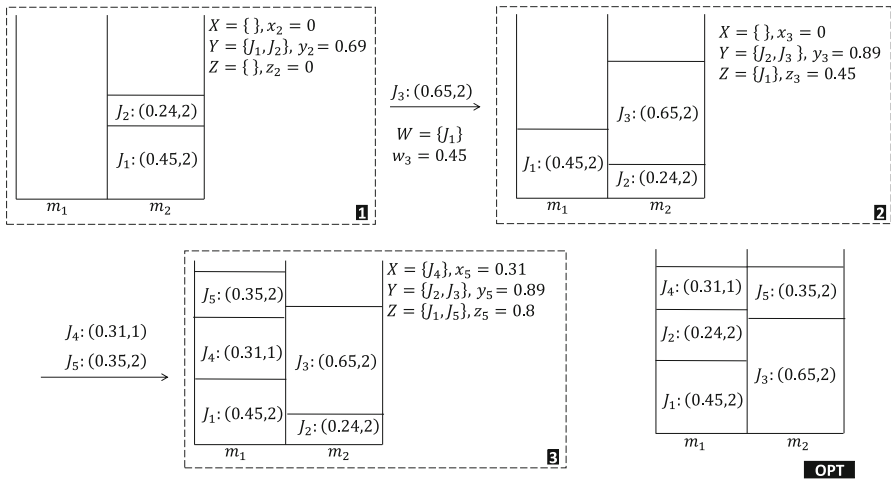


Fig. 16 The schedules produced by Algorithm D with migration factor $M = 0.7$ for input I_{12} of Sect. A.4.5 (from the moment when the second job had arrived), and an optimal solution for that input. We have $c_D(I_{12}) = 1.11$ and $c^*(I_{12}) = 1$

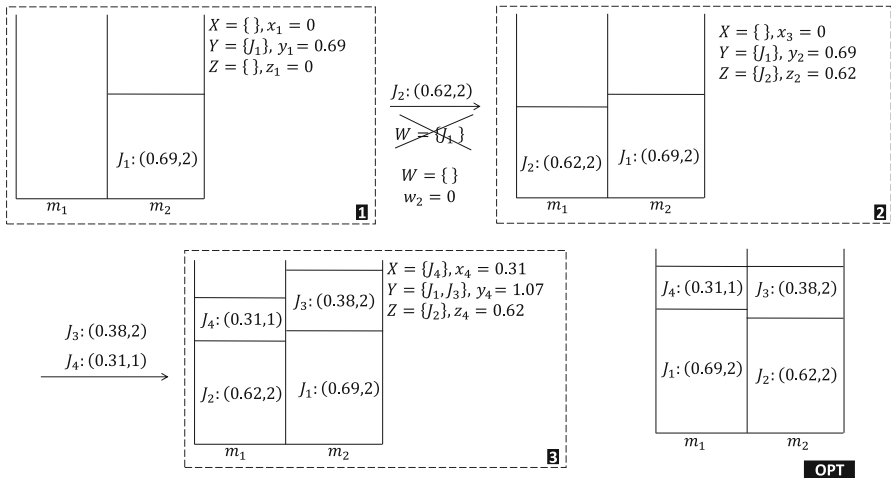


Fig. 17 The schedules produced by Algorithm D with migration factor $M = 0.7$ for input I_{13} of Sect. A.4.6, and an optimal solution for that input. We have $c_D(I_{13}) = 1.07$, $c^*(I_{13}) = 1$

Recall that $M = 0.7$. The first job is scheduled on the second machine in step 3, and we get: $x_1 = 0$, $y_1 = 0.69$, $z_1 = 0$. When the second job arrives, the algorithm reaches step 5 because $1.31 = y_1 + p_2 > 2 - M = 1.3$ and $p_2 = 0.62 < M = 0.7$ hold. In step 5, the algorithm defines W to be $\{J_1\}$ (the minimum length prefix with total size at least $\frac{M}{3} = \frac{7}{30} \approx 0.2333 \leq p_1 = w_2 = 0.69$). Here, we get: $w_2 > \min\{\frac{2M}{3}, M \cdot p_2\} = \min\{\frac{7}{15}, 0.434\}$, and in this case the algorithm updates W again to be $Y \setminus W = \{\}$, it schedules J_2 on the first machine, because $y_1 - w_2 + p_2 = 1.31 > 1.3 = 2 - M$ holds, and we get: $x_2 = 0$, $y_2 = 0.69$, $z_2 = 0.62$. The algorithm schedules the third

job on the second machine in step 3, and it schedules the last job on the first machine in step 2. We get: $x_4 = 0.31$, $y_4 = 1.07$, and $z_4 = 0.62$. As a result, the load of the first machine is 0.93, the load of the second machine is 1.07. See Fig. 17.

References

- Akaria I, Epstein L (2022) Online scheduling with migration on two hierarchical machines. *J Combin Optim* 44:3535–3538
- Albers S, Hellwig M (2012) Semi-online scheduling revisited. *Theoret Comput Sci* 443:1–9
- Angelesli E, Nagy Á.B, Speranza MG, Tuza Zs (2004) The on-line multiprocessor scheduling problem with known sum of the tasks. *J Sched* 7(6):421–428
- Azar Y, Regev O (2001) On-line bin-stretching. *Theoret Comput Sci* 268(1):17–41
- Bar-Noy A, Freund A, Naor JS (2001) On-line load balancing in a hierarchical server topology. *SIAM J Comput* 31(2):527–549
- Berndt S, Jansen K, Klein K (2020) Fully dynamic bin packing revisited. *Math Program* 179(1):109–155
- Böhm M, Sgall J, van Stee R, Veselý P (2017) Online bin stretching with three bins. *J Sched* 20(6):601–621
- Böhm M, Sgall J, van Stee R, Veselý P (2017) A two-phase algorithm for bin stretching with stretching factor 1.5. *J Combinator Optim* 34(3):810–828
- Chen X, Ding N, Dósa G, Han X, Jiang H (2015) Online hierarchical scheduling on two machines with known total size of low-hierarchy jobs. *Int J Comput Math* 92(5):873–881
- Cheng TCE, Kellerer H, Kotov V (2005) Semi-on-line multiprocessor scheduling with given total processing time. *Theoret Comput Sci* 337(1–3):134–146
- Crescenzi P, Gambosi G, Penna P (2004) On-line algorithms for the channel assignment problem in cellular networks. *Discret Appl Math* 137(3):237–266
- Dósa G, He Y (2004) Semi-online algorithms for parallel machine scheduling problems. *Computing* 72(3):355–363
- Epstein L (2003) Bin stretching revisited. *Acta Informatica* 39(2):97–117
- Epstein L, Levin A (2009) A robust APTAS for the classical bin packing problem. *Math Program* 119(1):33–49
- Epstein L, Levin A (2014) Robust algorithms for preemptive scheduling. *Algorithmica* 69(10):26–57
- Epstein L, Levin A (2019) Robust algorithms for total completion time. *Discret Optim* 33:70–86
- Epstein L, Noga J, Seiden SS, Sgall J, Woeginger GJ (2001) Randomized online scheduling on two uniform machines. *J Sched* 4(2):71–92
- Gabay M, Brauner N, Kotov V (2017) Improved lower bounds for the online bin stretching problem. *4OR: Q J Belgian French Ital Oper Res Soc* 15(2):183–199
- Gabay M, Kotov V, Brauner N (2015) Online bin stretching with bunch techniques. *Theoret Comput Sci* 602:103–113
- Gálvez W, Soto JA, Verschae J (2020) Symmetry exploitation for online machine covering with bounded migration. *ACM Trans Algorithms* 16(4):43:1–43:22
- Graham RL (1966) Bounds for certain multiprocessing anomalies. *Bell Syst Tech J* 45(9):1563–1581
- He Y, Zhang G (1999) Semi online scheduling on two identical machines. *Computing* 62(3):179–187
- Horowitz E, Sahni S (1976) Exact and approximate algorithms for scheduling nonidentical processors. *J ACM* 23(2):317–327
- Jiang Y (2008) Online scheduling on parallel machines with two GoS levels. *J Comb Optim* 16(1):28–38
- Jiang Y, He Y, Tang C (2006) Optimal online algorithms for scheduling on two identical machines under a grade of service. *J Zhejiang Univ-Sci A* 7(3):309–314
- Kellerer H, Kotov V (2013) An efficient algorithm for bin stretching. *Oper Res Lett* 41(4):343–346
- Kellerer H, Kotov V, Gabay M (2015) An efficient algorithm for semi-online multiprocessor scheduling with given total processing time. *J Sched* 18(6):623–630
- Kellerer H, Kotov V, Speranza MG, Tuza Zs (1997) Semi on-line algorithms for the partition problem. *Oper Res Lett* 21(5):235–242
- Lee K, Hwang H-C, Lim K (2014) Semi-online scheduling with GoS eligibility constraints. *Int J Prod Econ* 153:204–214
- Lee K, Lim K (2013) Semi-online scheduling problems on a small number of machines. *J Sched* 16(5):461–477

- Levin A (2022) Robust algorithms for preemptive scheduling on uniform machines of non-increasing job sizes. *Inf Process Lett* 174:106211
- Lim K, Lee K, Chang SY (2011) Improved bounds for online scheduling with eligibility constraints. *Theoret Comput Sci* 412(39):5211–5224
- Liu M, Chu C, Xu Y, Zheng F (2011) Semi-online scheduling on 2 machines under a grade of service provision with bounded processing times. *J Comb Optim* 21(1):138–149
- Luo T, Xu Y (2014) Semi-online scheduling on two machines with GoS levels and partial information of processing time. *Sci World J* 2014:576234
- Luo T, Xu Y (2015) Semi-online hierarchical load balancing problem with bounded processing times. *Theoret Comput Sci* 607(1):75–82
- Luo T, Xu Y (2016) Optimal algorithm for semi-online scheduling on two machines under GoS levels. *Optim. Lett.* 10(1):207–213
- Min X, Liu J, Wang Y (2011) Optimal semi-online algorithms for scheduling problems with reassignment on two identical machines. *Inf Process Lett* 111(9):423–428
- Park J, Chang SY, Lee K (2006) Online and semi-online scheduling of two machines under a grade of service provision. *Oper Res Lett* 34(6):692–696
- Qi X, Yuan J (2019) Semi-online hierarchical scheduling on two machines for l_p -norm load balancing. *Asia-Pac J Oper Res* 36(1):1
- Sanders P, Sivasadan N, Skutella M (2009) Online scheduling with bounded migration. *Math Oper Res* 34(2):481–498
- Skutella M, Verschae J (2016) Robust polynomial-time approximation schemes for parallel machine scheduling with job arrivals and departures. *Math Oper Res* 41(3):991–1021
- Tan Z, He Y (2002) Semi-on-line problems on two identical machines with combined partial information. *Oper Res Lett* 30(6):408–414
- Tan Z, Zhang A (2011) Online hierarchical scheduling: an approach using mathematical programming. *Theoret Comput Sci* 412(3):246–256
- Wakrat I (2012) Online and semi-online scheduling with reordering and reassignment. Master's thesis, Department of Computer Science, University of Haifa, Haifa, Israel
- Wu Y, Ji M, Yang Q (2012) Optimal semi-online scheduling algorithms on two parallel identical machines under a grade of service provision. *Int J Prod Econ* 135(1):367–371
- Xiao M, Wu G, Li W (2019) Semi-online machine covering on two hierarchical machines with known total size of low-hierarchy jobs. In: *Proceedings of the 37th national conference of theoretical computer science (NCTCS'19)*, pp 95–108
- Zhang A, Jiang Y, Fan L, Hu J (2015) Optimal online algorithms on two hierarchical machines with tightly-grouped processing times. *J Comb Optim* 29:781–795
- Zhang A, Jiang Y, Tan Z (2009) Online parallel machines scheduling with two hierarchies. *Theoret Comput Sci* 410(38–40):3597–3605

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.