Journal of Scheduling (2022) 25:643–657
https://doi.org/10.1007/s10951-022-00732-y

Online algorithms to schedule a proportionate flexible flow shop of batching machines

Christoph Hertrich[1,2] · Christian Weiß[3] · Heiner Ackermann[3] · Sandy Heydrich[3] · Sven O. Krumke[1]

Accepted: 25 February 2022 / Published online: 11 April 2022

Abstract

This paper is the first to consider online algorithms to schedule a proportionate flexible flow shop of batching machines (PFFB). The scheduling model is motivated by manufacturing processes of individualized medicaments, which are used in modern medicine to treat some serious illnesses. We provide two different online algorithms, proving also lower bounds for the offline problem to compute their competitive ratios. The first algorithm is an easy-to-implement, general local scheduling heuristic. It is 2-competitive for PFFBs with an arbitrary number of stages and for several natural scheduling objectives. We also show that for total/average flow time, no deterministic algorithm with better competitive ratio exists. For the special case with two stages and the makespan or total completion time objective, we describe an improved algorithm that achieves the best possible competitive ratio $\varphi = \frac{1+\sqrt{5}}{2}$, the golden ratio. All our results also hold for proportionate (non-flexible) flow shops of batching machines (PFB) for which this is also the first paper to study online algorithms.

Keywords Planning of pharmaceutical production · Proportionate flow shop · Flexible flow shop · Batching machines · Online algorithms · Competitive analysis

1 Introduction

In modern pharmacy, in order to treat various serious illnesses, individualized medicaments are produced to order for a specific patient. These production processes often take place in a complex production line, consisting of many different steps.

As long as each step of the process is still performed manually, for example by a laboratory worker, usually at each step only one patient can be handled at a time. However, once the process is scaled to industrial production levels, instead, for some steps, machines, like pipetting robots, are used. Indeed, at that point, in order to scale up production, usually several machines are used in parallel at each step. What is more, these types of machines can often handle multiple patients simultaneously. If scheduled efficiently, this special feature can drastically increase the throughput of the production line. Clearly, in such an environment efficient operative planning is crucial in order to optimize the performance of the manufacturing process and treat as many patients as possible as quickly as possible.

In a practical setting, the producer of the individualized drug knows nothing about the patient until the medicine is actually ordered. This naturally creates an online scheduling scenario, for which efficient, computable and, if possible, easy-to-understand scheduling rules are needed. Therefore, in this paper, we specifically deal with the online version of the problem, although some of our findings are interesting in terms of offline scheduling as well.

✉ Christian Weiß
christian.weiss@itwm.fraunhofer.de

Christoph Hertrich
c.hertrich@lse.ac.uk

Heiner Ackermann
heiner.ackermann@itwm.fraunhofer.de

Sandy Heydrich
sandy.heydrich@itwm.fraunhofer.de

Sven O. Krumke
krumke@mathematik.uni-kl.de

[1] Department of Mathematics, Technische Universität Kaiserslautern, 67663 Kaiserslautern, Germany

[2] Institute of Mathematics, Technische Universität Berlin, 10623 Berlin, Germany

[3] Department of Optimization, Fraunhofer Institute for Industrial Mathematics ITWM, 67663 Kaiserslautern, Germany

Formally, the manufacturing process studied in this paper is structured in a *flexible flow shop* manner (also called *hybrid flow shop* in the literature). A *job* J_j, $j = 1, 2, \dots, n$, representing the production of a drug for a specific patient, has to be processed in s *stages* $S_1, S_2, \dots, S_s$ in order of their numbering. At each stage S_i there are available m_i identical, parallel machines $M_i^{(1)}, M_i^{(2)}, \dots, M_i^{(m_i)}$ to process the jobs. If each stage consists of only one machine, we may drop the machine index and instead identify each stage S_i with its single machine M_i.

Each job J_j has a *release date* $r_j \geq 0$, denoting the time at which the job J_j is available for processing at the first stage S_1. We assume that jobs are indexed in earliest release date order. Furthermore, a job is only available for processing at stage S_i, $i = 2, 3, \dots, s$, when it has finished processing at the previous stage S_{i-1}.

Processing times are only dependent on the stage, not on the job or the specific machine where the job is processed (recall that machines at each stage are identical). This means that each stage S_i, $i = 1, 2, \dots, s$, is associated with a fixed *processing time* p_i, which is the same for every job when processed at that stage on any machine. In the literature, a (flexible) flow shop with such job-independent processing times is sometimes called a *proportionate* flow shop, see, e.g., Panwalkar et al. (2013).

Recall that, as a special feature from our application, each machine in the flexible flow shop can potentially handle multiple jobs at the same time. These kind of machines are called *(parallel) batching machines*, and a set of jobs processed at the same time on some machine is called a *batch* on that machine (Brucker, 2007, Chapter 8). All jobs in one batch on some machine $M_i^{(k)}$ of stage S_i have to start processing on $M_i^{(k)}$ at the same time. In particular, all jobs in one batch at stage S_i have to be available for processing at S_i, before the batch can be started. The processing time of a batch on $M_i^{(k)}$ remains p_i, no matter how many jobs are included in this batch. At each stage S_i, $i = 1, 2, \dots, s$, machines have a common *maximum batch size* (or *batch capacity*) b_i, which is the maximum number of jobs a batch on machines of stage S_i may contain.

Given a feasible schedule ς, we denote by $c_{ij}(\varsigma)$ the *completion time* of job J_j at stage S_i. For the completion time of job J_j on the last machine we also write $C_j(\varsigma) = c_{sj}(\varsigma)$. If there is no confusion which schedule is considered, we may omit the reference to the schedule and simply write c_{ij} and C_j.

As optimization criteria, we study the four objective functions *makespan* $C_{\max} = \max\{C_j \mid j = 1, 2, \dots, n\}$, *total completion time* $\sum C_j = \sum_{j=1}^n C_j$, *maximum flow time* $F_{\max} = \max\{F_j \mid j = 1, 2, \dots, n\}$ and *total flow time* $\sum F_j = \sum_{j=1}^n F_j$, where $F_j = C_j - r_j$. Note that the total flow time (or average flow time, if divided by the number of jobs) measures the average time a patient has to wait for his or her medicament, after the production is ordered. As short waiting times are essential in the treatment of life threatening illnesses, this objective is particularly relevant in practice. For more considerations about meaningful performance measures in applications, we refer to Ackermann et al. (2020).

Using the standard three-field notation for scheduling problems (Graham et al., 1979; Pinedo, 2012), our problem is denoted as

$$FFs \mid r_j, p_{ij} = p_i, p\text{-batch}, b_i \mid f,$$

where f is one of the four objective functions from above. We refer to the described scheduling model as *proportionate flexible flow shop of batching machines* and abbreviate it by *PFFB*. If we consider the special case where each stage consists of only one machine, we call this the usual *proportionate flow shop of batching machines* and abbreviate it by *PFB*.

In this paper we deal with the *online* problem to schedule a PFFB where each job is unknown until its release date. In particular, this means that the total number n of jobs remains unknown until the end of the scheduling process.

Throughout this paper, we write $\varphi = \frac{1+\sqrt{5}}{2} \approx 1.618$ for the golden ratio.

Next, we provide an (offline) example in order to illustrate the problem setting.

Example 1 Consider an instance of PFFB with $s = 2$ stages, $m_1 = 1$ machine at stage S_1 and $m_2 = 2$ machines at S_2, maximum batch sizes $b_1 = 3$ and $b_2 = 2$, processing times $p_1 = 3$ and $p_2 = 4$, as well as, $n = 5$ jobs with release dates $r_1 = r_2 = 0$, $r_3 = 1$, and $r_4 = r_5 = 3$. Fig. 1 illustrates a feasible schedule for the instance as job-oriented Gantt chart.

Each rectangle labeled by a machine represents a batch of jobs processed together on this machine. The black area indicates that the respective jobs have not been released at this time yet. Note that in this example none of the batches can be started earlier, since either a job of the batch has just arrived when the batch is started, or the machine is occupied before. Still, the schedule does not minimize the makespan,

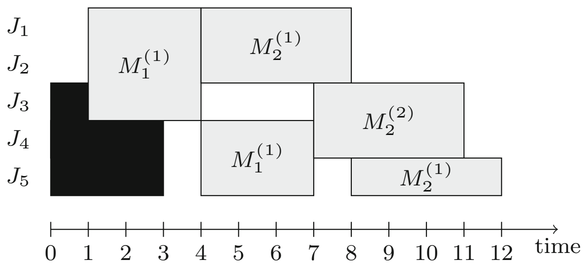

Fig. 1 A feasible example schedule

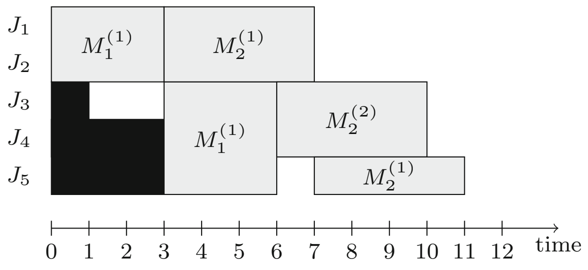

Fig. 2 An example schedule minimizing the makespan

since the schedule shown in Fig. 2 is feasible as well and has a makespan of 11 instead of 12.

The improvement in the makespan was achieved by reducing the size of the first batch on $M_1^{(1)}$ from three to two, which allows to start it one time step earlier. Observe that no job can arrive at S_2 before time step 3. Moreover, one of the two machines at S_2 has to process at least two batches in order to complete all five jobs. This takes at least 8 time units. Thus, 11 is the optimal makespan and no further improvement is possible.

1.1 Our results

This is the first study of P(F)FBs from an online perspective. We concentrate our research on the four practically highly relevant objective functions makespan, total completion time, maximum flow time and total flow time. An overview of our results can be found in Table 1.

Concerning lower bounds on the competitive ratio, we observe that previously known bounds of φ for one-stage PFFBs to minimize makespan or maximum flow time by Zhang et al. (2003) and Jiao et al. (2014), respectively, carry over to arbitrarily many stages by introducing stages with negligible processing times. In addition, we show that φ is also a lower bound with respect to total completion time, while for total flow time even a lower bound of 2 can be achieved.

Concerning upper bounds on the competitive ratio, we provide two algorithms. We first introduce the Never-Wait strategy, where machines are only idle if not enough patients are available for processing (see Sect. 6 for details). This strategy is easy to implement and applicable for all four considered objective functions and very general machine environments, even with large number of stages and/or large numbers of machines per stage. We prove that the Never-Wait strategy achieves a competitive ratio of 2 for any such instance and all four considered objectives. In particular, this is best-possible with respect to total flow time. We also show that the "opposite" strategy, where we always wait until a full batch can be started, is not an α-approximation algorithm (and, in particular, not α-competitive) for any $\alpha > 1$. For the specific scenario of $s = 2$ stages, we also introduce the t-Switch strategy, where the first stage is scheduled in such a way, that as many jobs as possible are available at the second stage at some time t; at the second stage, waiting is allowed until time t and afterward the stage is scheduled as in the Never-Wait strategy (see Sect. 7 for details). Choosing t correctly yields an improved competitive ratio with respect to makespan and total completion time from 2 to φ, which is also best possible for these problems.

1.2 Overview of this paper

The remainder of this paper is structured as follows. In Sect. 2, we give an overview of related literature. In Sect. 3 we provide lower bounds for the competitive ratio in PFFBs for the four objective functions we consider. Sections 4 and 5 are dedicated to establishing the structural results needed in order to prove competitiveness of the online algorithms we provide in the subsequent sections: in Sect. 4 we prove that permutation schedules with jobs ordered by release dates are optimal for all of our four objective functions, and in Sect. 5 we show lower bounds for completion times of jobs in such permutation schedules. The latter are needed to prove competitiveness of our algorithms. Sections 6 and 7 deal with the

Table 1 Known bounds for the competitive ratio of deterministic online algorithms to schedule a PFFB. Note that lower bounds for a single stage ($s = 1$) carry over to arbitrary many stages by introducing stages with negligible processing times

Objective function	Number of stages	Lower bound	Upper bound	Reference lower bound	Reference upper bound
$C_{\max}$	1	φ	φ	Zhang et al. (2003)	Zhang et al. (2003)
	2	φ	φ	Zhang et al. (2003)	**Theorem** 23
	≥ 3	φ	2	Zhang et al. (2003)	**Corollary** 12
$\sum C_j$	1, 2	φ	φ	**Theorem** 3	**Theorem** 23
	≥ 3	φ	2	**Theorem** 3	**Corollary** 12
$F_{\max}$	1	φ	φ	Jiao et al. (2014)	Jiao et al. (2014)
	≥ 2	φ	2	Jiao et al. (2014)	**Corollary** 12
$\sum F_j$	≥ 1	2	2	**Theorem** 4	**Corollary** 12

general Never-Wait and the more specialized t-Switch algorithm, respectively. Finally, in Sect. 8 we present conclusions and some further thoughts.

2 Literature

To the best of our knowledge, neither the online PFFB problem, nor its special case, the online PFB problem, have been studied before. A practical study specific to the application introduced in the beginning can be found in Ackermann et al. (2020). Some of the concepts investigated in this paper in general have already been introduced in an application specific sense in that paper.

Even for the offline PFFB problem, little is known. Previous work has been focused on (usually non-proportionate) NP-hard generalizations motivated by applications in various manufacturing industries. Common techniques include mixed-integer programming models or application-specific heuristics. See, e.g., Amin-Naseri and Beheshti-Nia (2009), Luo et al. (2011), Li et al. (2015), Tan et al. (2018).

In the special case of offline PFBs, with only one machine per stage, proportionate versions have been studied explicitly. Sung et al. (2000) propose heuristic approaches to minimize the makespan and the total completion time in a PFB. However, they do not establish any complexity result. For the special case of two stages, Ahmadi et al. (1992), as well as Sung and Yoon (1997) present polynomial time algorithms. In a previous paper (Hertrich et al., 2020), we present a dynamic program that can be used to minimize several traditional objective functions (including the four objectives studied in this paper) in polynomial time for any fixed number s of stages. For the case of s being part of the input, the complexity status of offline PFBs is open. Significant hardness results have, to the best of our knowledge, not been achieved at all. See also (Hertrich et al., 2020), for an in-depth literature review for offline PFBs.

Although the PFFB problem itself has not been investigated from an online or offline perspective before, there are several helpful results for related online problems in the literature. For our purposes, the most interesting family of related problems is online scheduling of single and parallel batching machines. With job-independent processing times, these problems can be viewed as the one-stage versions of P(F)FBs. We refer to Tian et al. (2014) for a survey.

Concerning online makespan minimization for a single batching machine with identical processing times, Zhang et al. (2001) and Deng et al. (2003) show that no deterministic online algorithm can achieve a competitive ratio better than the golden ratio φ. Fang et al. (2011) provide a deterministic online algorithm matching this bound, even in a slightly more general setting, where processing times are not assumed to be identical, but only *grouped*, that is, differing by a factor of at most φ from each other. Moreover, Zhang et al. (2003) show that φ is also the precise competitive ratio for makespan minimization on parallel batching machines with identical processing times, that is, a one-stage PFFB. Li and Chai (2018) extend this result to minimizing the maximum weighted completion time.

Concerning the total completion time objective, Cao et al. (2011) present a 2-competitive online algorithm for parallel batching machines with identical processing times, which even works in the presence of precedence constraints. For the generalization where jobs are allowed to have unequal processing times and the total weighted completion time objective, $(4+\varepsilon)$-competitive algorithms are known (Chen et al., 2004; Ma et al., 2014).

Research on scheduling parallel batching machines to minimize maximum flow time started with the case where batches may have unbounded size (Li & Yuan, 2011). For the bounded batch model, as we consider it in this paper, Jiao et al. (2014) provide a φ-competitive deterministic online algorithm to minimize maximum flow time on parallel batching machines with identical processing times, i.e., a one-stage PFFB, and prove that this is best possible. Hence, the competitive ratio for makespan minimization by Zhang et al. (2003) carries over to maximum flow time. Recently, it has been shown that the competitive ratio of φ for maximum flow time remains also valid for maximum weighted flow time (Chai et al., 2019) or maximum flow time with delivery times (Lin et al., 2019).

Although total flow time seems to be a very reasonable objective function from a practical perspective, we are not aware of any previous research about competitive algorithms for bounded p-batching problems on single or parallel machines to minimize total flow time.

3 Lower bounds for the competitive ratio

In this section we provide lower bounds on the competitiveness of deterministic online algorithms for PFFB problems with our considered objective functions $C_{\max}$, $\sum C_j$, $F_{\max}$, and $\sum F_j$.

Before we start, note that lower bounds on PFFBs with only one stage naturally extend to PFFBs with arbitrarily many stages by choosing negligible processing times for all stages but the first one. Therefore, in the following, we focus on such instances with $s = 1$.

For the makespan objective, Zhang et al. (2003) prove that the golden ratio φ is a lower bound on parallel batching machines with identical processing times, i.e., a one-stage PFFB. Furthermore, Jiao et al. (2014) prove the same bound of φ for the maximum flow time objective.

Theorem 2 (Zhang et al., 2003; Jiao et al., 2014) *There are no deterministic online algorithms to minimize the makespan or maximum flow time in a PFFB with a competitive ratio less than φ. This result holds even if no stage has maximum batch size larger than* 2.

By using constructions similar to Zhang et al. (2003) and Jiao et al. (2014), we also obtain lower bounds on the competitive ratio for the total completion and flow time objectives. We first show that the golden ratio φ is also a lower bound for the competitiveness for the total completion time objective.

Theorem 3 *There are no deterministic online algorithms to minimize the total completion time in a PFFB with a competitive ratio less than φ.*

Proof Consider a PFFB instance with $s = 1, m_1 = 1, p_1 = 1$ and a fixed batch size b_1. Suppose the first job J_1 is released at time $r_1 = 0$. We distinguish two possible behaviors of a deterministic online algorithm.

Case 1: The algorithm starts a batch consisting of only J_1 at a point in time $t \leq \varphi - 1$. In this case, suppose $b_1 - 1$ further jobs are released at time $t + \varepsilon$ for some small $\varepsilon > 0$. Since these jobs can only be started after J_1 is finished, the schedule produced by the algorithm has a total completion time of at least $(t+1)+(b_1-1)(t+2) = b_1t+2b_1-1$. An optimal schedule would instead process all jobs in a single batch, resulting in a total completion time of $b_1(t + \varepsilon + 1)$. If ε tends to zero, this results in a competitive ratio of at least $\frac{b_1t+2b_1-1}{b_1(t+1)} \xrightarrow{b_1\to\infty} \frac{t+2}{t+1} \geq \frac{\varphi+1}{\varphi} = \varphi$.

Case 2: The algorithm does not start a batch before time $\varphi - 1$. In this case, suppose no further job is released and J_1 is the only job of the whole instance. The schedule produced by the algorithm has a total completion time of at least φ. An optimal schedule would start processing J_1 immediately at time $r_1 = 0$, resulting in a total completion time of 1. Hence, also in this case, the competitive ratio is at least φ. □

Finally, for the total flow time, we can obtain a lower bound of 2. The construction is very similar to the last proof, but instead of using $t = \varphi - 1$ as the border between the two cases, this time we use $t = 1$.

Theorem 4 *There are no deterministic online algorithms to minimize the total flow time in a PFFB with competitive ratio less than* 2.

Proof As in the proof of Theorem 3, consider a PFFB instance with $s = 1$, $m_1 = 1$, $p_1 = 1$ and a fixed batch size b_1. Again, we distinguish two cases, dependent on the start time of the first batch.

Case 1: The algorithm starts a batch consisting of only J_1 at a point in time $t \leq 1$. In this case, suppose $b_1 - 1$ further jobs are released at time $t + \varepsilon$ for some small $\varepsilon > 0$. Since these jobs can only be started after J_1 is finished, the schedule produced by the algorithm has a total flow time of at least $(t+1)+(b_1-1)(t+2-t-\varepsilon) = t+1+(b_1-1)(2-\varepsilon)$. An optimal schedule would instead process all jobs in a single batch, resulting in a total flow time of $(t+\varepsilon+1)+b_1-1$. If ε tends to zero, this results in a competitive ratio of at least $\frac{t+1+2(b_1-1)}{t+b_1} \xrightarrow{b_1\to\infty} 2$.

Case 2: The algorithm does not start a batch before time 1. In this case, suppose no further job is released and J_1 is the only job of the whole instance. The schedule produced by the algorithm has a total flow time of at least 2, while an optimal schedule could have started processing J_1 immediately at time $r_1 = 0$, resulting in a total flow time of 1. Hence, also in this case, the competitive ratio is at least 2. □

4 Optimality of permutation schedules

In a *permutation schedule* the order of the jobs is the same on all stages of the flexible flow shop. This means there exists a permutation π of the job indices such that $c_{i\pi(1)} \leq c_{i\pi(2)} \leq \ldots \leq c_{i\pi(n)}$, for all $i = 1, \ldots, s$. Since the processing times only depend on the stage and no preemption is possible, clearly the same then also holds for starting times instead of completion times. Note that this definition is not dependent on the specific machine where a job is scheduled. If there exists an optimal schedule which is a permutation schedule with a certain ordering π of the jobs, we say that *permutation schedules are optimal*. A job ordering π which gives rise to an optimal permutation schedule is then called an *optimal job ordering*. Finally, an ordering π of the jobs is called an *earliest release date ordering*, if $r_{\pi(1)} \leq r_{\pi(2)} \leq \ldots \leq r_{\pi(n)}$.

Using the techniques in the proofs of Lemma 2 and Theorem 3 from (Hertrich et al., 2020), we obtain the following theorem. For the interested reader, details of the proof are available in the appendix.

Theorem 5 *For a PFFB with objective function $C_{\max}$, $\sum C_j$, $F_{\max}$, or $\sum F_j$, permutation schedules are optimal. Moreover, any earliest release date order is an optimal ordering of the jobs.*

From now on, we assume that jobs are indexed in earliest release date order and restrict our attention to permutation schedules where the job order is given by the indices. We therefore drop the notation of π. It remains to decide, for each stage, how the job set should be divided into batches on the individual machines and when to process these batches. In other words, every time a machine becomes idle and at least one job is available for processing, we have to decide how long to wait for the arrival of more jobs before starting the next batch. Waiting for additional jobs incurs the cost of delaying the already available jobs.

5 Lower bounds for the completion times at each stage

When analyzing approximation or online algorithms, one typically compares the quality of the solution produced by the algorithm with the optimal offline solution. However, for many problems, including PFFBs, it is difficult to make a precise statement about the quality of such an optimal solution in the offline scenario. In these cases, a common approach is to use a lower bound for comparison instead. Therefore, in the first part of this section we develop a lower bound c^*_{ij} for the completion time c_{ij} of each job J_j at each stage S_i. No feasible permutation schedule with job order given by the indices can yield smaller completion times.

In the second part of this section, we show how this lower bound can be interpreted as a solution of a proportionate flexible flow shop problem without batching machines.

Finally, we conclude the section by comparing our bound to another one given by Sung et al. (2000).

5.1 Recursive formula for the lower bounds

We start by observing two properties that hold for the completion times of any permutation schedule with job order $J_1, J_2 \dots, J_n$.

Firstly, since a job needs to finish stage S_{i-1} before it can be started at stage S_i, we have

$$c_{ij} \geq c_{(i-1)j} + p_i \tag{1}$$

for all $i \in \{2, 3, \dots, s\}$ and $j \in [n]$.

Secondly, for a stage index i and a job index j, consider the two jobs J_j and $J_{j-m_i b_i}$. Suppose there exists a point in time t at which both jobs are simultaneously processed at stage S_i. Due to the fixed job permutation, this would imply that all the $m_i b_i+1$ jobs $J_{j-m_i b_i}, J_{j-m_i b_i+1}, \dots, J_j$ would be processed at stage S_i at this time t. However, this contradicts the batch capacity restriction b_i of the m_i machines. Hence, such a point in time t cannot exist and we may conclude

$$c_{ij} \geq c_{i(j-m_i b_i)} + p_i \tag{2}$$

for all $i \in [s]$ and all $j = m_i b_i + 1, m_i b_i + 2, \dots, n$.

Now we construct the desired lower bound via recursion, using the right-hand sides of the two properties above. As starting values, we define $c^*_{0j} = r_j$ for all $j \in [n]$ and $c^*_{ij} = -\infty$ for $i \in [s]$ and $j \leq 0$. Then, we define

$$c^*_{ij} = \max\{c^*_{(i-1)j}, c^*_{i(j-m_i b_i)}\} + p_i. \tag{3}$$

Clearly, by inductive application of the two properties (1) and (2), the values c^*_{ij} defined this way are a lower bound for the completion times of a permutation schedule for a PFFB, as stated in the following lemma.

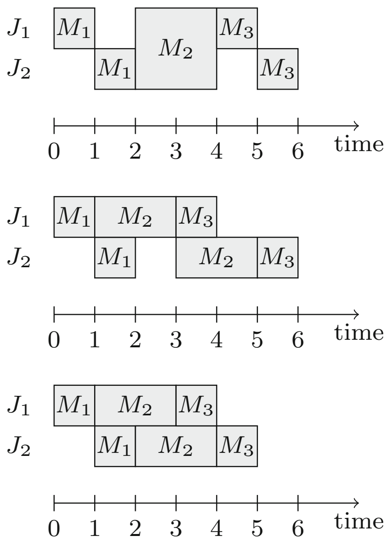

Fig. 3 The first two figures show the two optimal permutation schedules (ordered by their indices) in Example 7; the third figure shows the *infeasible* schedule implied by lower bound (3), where machine M_2 is allowed to start a new batch while another is still running

Lemma 6 *Any feasible permutation schedule with job order $J_1, J_2, \dots, J_n$ satisfies $c_{ij} \geq c^*_{ij}$ for all $i \in [s]$ and $j \in [n]$.*

Before we proceed to the next part, we give an example to show that the lower bound cannot always be achieved by a feasible schedule. Hence, it is not tight in general.

Example 7 Consider a PFFB instance without release dates, consisting of three stages with one machine each and two jobs (i.e., $s = 3$, $m_1 = m_2 = m_3 = 1$, $n = 2$). Processing times and batch capacities at each stage are given by $p_1 = b_1 = p_3 = b_3 = 1$ and $p_2 = b_2 = 2$. The only batching decision to make is whether to batch both jobs together at stage S_2 or not. The two corresponding permutation schedules are illustrated in Fig. 3. For ease of notation, we identify each stage S_i with its single associated machine M_i. In both cases we have $c_{32} = 6$. However, recursively applying (3) yields $c^*_{32} = 5$, illustrated in the third part of Fig. 3.

5.2 The lower bounds as completion times of a proportionate flexible flow shop

Given an instance of PFFB, consider the following instance of a proportionate flexible flow shop (PFF) problem: at each stage, instead of m_i parallel batching machines with maximum batch size b_i, there are $m_i b_i$ parallel machines with maximum batch size 1. In other words, any batching machine with batch size b_i at stage S_i is replaced by b_i identical parallel machines without batching. All other data of the instance remain the same. Considering the original PFFB instance, we call the instance of PFF constructed this way *the corresponding instance without batching*.

The main difference between the PFFB instance and its corresponding instance without batching is that the $m_i b_i$ machines in the instance without batching can start jobs independently from one another, whereas in the PFFB instance, jobs in the same batch have to be started at the same time. So the corresponding instance without batching allows for $m_i b_i$ independent starts, while the PFFB setting only allows for m_i many.

Clearly, any feasible schedule for the PFFB instance implies a feasible schedule in the corresponding instance without batching, by keeping all start and finish times the same and only splitting up batches across machines such that one job runs on each machine. The reverse does not work. Indeed, considering again Example 7, we can see that the third part of Fig. 3 shows a solution to the corresponding instance without batching which cannot be transformed into a solution for the PFFB instance.

The next theorem shows that an optimal schedule for the PFF instance is determined by the values $c^*_{ij}, i \in [s], j \in [n]$, from (3).

Theorem 8 *Consider a PFFB instance with a regular objective function for which permutation schedules are optimal. Then the values c^*_{ij}, $i \in [s]$, $j \in [n]$, are the completion times of an optimal schedule of the corresponding PFF instance without batching.*

Proof Since there are no batching machines involved in a PFF, there is no need to wait for the arrival of other jobs in order to achieve a fuller batch. Therefore, an optimal permutation schedule for the PFF can be achieved by starting each job J_j at each stage S_i as soon as the following two conditions are satisfied:

– The job J_j has finished stage S_{i-1} (or has been released, if $i = 1$).
– There is a machine available for processing job J_j at stage S_i and all jobs $J_1, J_2, \ldots, J_{j-1}$ have already been started at that stage. Due to the fixed permutation and the number $m_i b_i$ of machines at stage S_i, this is the case as soon as job $J_{j-m_i b_i}$ has finished stage S_i (or immediately, if $j \leq m_i b_i$).

Putting these conditions together, the completion time of job J_j at stage S_i in an optimal permutation schedule can be calculated recursively by

$$c_{ij} = \max\{c_{(i-1)j}, c_{i(j-m_i b_i)}\} + p_i.$$

This is exactly the same formula as in the definition (3) of the values c^*_{ij}, $i \in [s]$, $j \in [n]$. □

5.3 Comparison with the bound of Sung et al. (2000)

As mentioned in the literature review, Sung et al. (2000) propose heuristic algorithms to schedule a PFB, that is, a PFFB with only one machine at each stage, without release dates. In order to evaluate their experiments, they provide a lower bound for the makespan. Transferred to our notation, it reads as follows:

$$\max\left\{\left\lceil \frac{j}{b_1} \right\rceil p_1 + \sum_{i=2}^{k-1} p_i + \left\lceil \frac{n-j+1}{b_k} \right\rceil p_k + \sum_{i=k+1}^{s} p_i \;\middle|\; j \in [n], k \in [s]\right\}. \tag{4}$$

We show that our bound is an improvement, i.e., that c^*_{sn} is at least as large as the bound given by (4) and that for at least one instance, our bound is strictly larger.

Fix $j \in [n]$ and $k \in [s]$ as the maximizers in (4). Applying the recursive definition (3) with $m_i = 1$ for all $i \in [s]$ and $r_j = 0$ for all $j \in [n]$, we obtain the following four inequalities.

$$\begin{aligned} c^*_{1j} &\geq \left\lceil \frac{j}{b_1} \right\rceil p_1, \\ c^*_{kj} &\geq c^*_{1j} + \sum_{i=2}^{k} p_i, \\ c^*_{kn} &\geq c^*_{kj} + \left\lfloor \frac{n-j}{b_k} \right\rfloor p_k \\ &= c^*_{kj} + \left(\left\lceil \frac{n-j+1}{b_k} \right\rceil - 1\right) p_k, \\ c^*_{sn} &\geq c^*_{kn} + \sum_{i=k+1}^{s} p_i. \end{aligned}$$

Summing up these four inequalities yields that c^*_{sn} is at least as large as the bound (4) by Sung et al. (2000). Moreover, the following example shows that c^*_{sn} can be strictly larger than (4). Consider a PFB (i.e., only one machine per stage) instance with $s = 3$ stages. Let $n = 6$ be the number of jobs and let $p_1 = b_1 = 1$, $p_2 = 3$, $b_2 = 2$, $p_3 = 5$, and $b_3 = 3$. One can easily check that for this instance the maximum in (4) is 16, which is attained either for $j = 2$ and $k = 2$ or for $j = 3$ and $k = 3$. However, for our bound it holds that

$$c^*_{36} \geq 5 + c^*_{33} \geq 10 + c^*_{23} \geq 13 + c^*_{21} \geq 16 + c^*_{11} \geq 17.$$

Hence, c^*_{sn} is a strict improvement upon the bound (4) of Sung et al. (2000).

We can compute c^*_{sn} in time $\mathcal{O}(ns)$ by recursively using (3). Since the computation of (4) takes the same asymptotic runtime, our improvement of the bound incurs no increase in computational cost.

6 The Never-Wait algorithm

This section is devoted to establishing a simple, yet reasonably effective online scheduling rule for a PFFB. Again we focus on permutation schedules with job order $J_1, J_2, \dots, J_n$. Hence, on each stage, when a machine is idle and jobs are available, the main scheduling decision is how long to wait until starting a new batch. One obvious strategy is to always immediately start a batch when jobs are present and a machine is idle. We call this strategy the Never-Wait algorithm. Another strategy, in some ways the opposite to the Never-Wait algorithm, is to always wait until a full batch can be started. This strategy is called the Full-Batch algorithm. In this section, we show that the Never-Wait algorithm is a 2-approximation with respect to various objective functions and, hence, 2-competitive when seen as an online algorithm. Furthermore, we show that the Full-Batch algorithm admits no constant approximation guarantee at all.

Definition 9 The *Never-Wait algorithm* for scheduling a PFFB is defined as follows: A batch is started at a stage whenever at least one job is available for processing at that stage and at least one machine is idle at that stage. The size of the new batch is chosen as large as possible, i.e., the minimum of the batch capacity at the stage and the number of available jobs at the stage when the batch is started.

Note that, if at stage S_i several machines are idle but there are not more than b_i jobs available, only one machine is started. On the other hand, if there are more than b_i jobs available at stage S_i, then more than one machine can be started at the same time.

In the following, let c_{ij}, $i \in [s]$, $j \in [n]$, be the completion times resulting from the Never-Wait algorithm and c^*_{ij} be the bound defined in Sect. 5. We also write $c_{0j} = c^*_{0j} = r_j$ for the time at which J_j becomes available at S_1.

Since the Never-Wait algorithm greedily starts as many available jobs as possible, the following property holds.

Lemma 10 *Suppose that, for some $j \in [n]$ and some $i \in [s]$, it holds that $c_{ij} > c_{(i-1)j} + 2p_i$. Then $j > m_i b_i$ and $c_{i(j-m_i b_i)} \geq c_{ij} - p_i$.*

Proof Notice that $c_{ij} > c_{(i-1)j} + 2p_i$ implies that J_j is available but not started at stage S_i during the complete interval $\lambda = [c_{ij} - 2p_i, c_{ij} - p_i[$. Since interval λ has length p_i and at least one job is available during all of interval λ, each of the m_i machines starts processing exactly one batch during λ. Moreover, since J_j is already available, but not included in one of these batches, all these batches must be full batches. Hence, we obtain that at least $m_i b_i$ jobs complete stage S_i in the time interval $[c_{ij} - p_i, c_{ij}[$. In particular, using the fixed job permutation, this implies that $j > m_i b_i$ and that $J_{j-m_i b_i}$ is completed not before time $c_{ij} - p_i$. □

Now we are ready to show that the completion times produced by the Never-Wait algorithm can be bounded from above in terms of the lower bound c^*_{ij} of the previous section.

Theorem 11 *For all $i \in [s]$ and $j \in [n]$, the completion time c_{ij} in the Never-Wait algorithm satisfies*

$$c_{ij} \leq c^*_{ij} + \sum_{i'=1}^{i} p_{i'}.$$

Proof We use a simultaneous induction on i and j. Using $c_{0j} = r_j = c^*_{0j}$, the following arguments settle induction start ($i = 1$) and induction step ($i > 1$) at the same time.

Let $i \in [s]$ be a stage index and let $j \in [n]$ be a job index. Suppose the claim is already proven for all pairs of indices $i' \leq i$ and $j' \leq j$ with either $i' < i$ or $j' < j$. We distinguish two cases.

Case 1: J_j waits for at most p_i time units at stage S_i, i.e., $c_{ij} \leq 2p_i + c_{(i-1)j}$. In particular, by Lemma 10, this always holds if $j \leq m_i b_i$. We obtain

$$\begin{aligned} c_{ij} &\leq 2p_i + c_{(i-1)j} \\ &\overset{\text{ind.}}{\leq} p_i + p_i + c^*_{(i-1)j} + \sum_{i'=1}^{i-1} p_{i'} \\ &\overset{(3)}{\leq} c^*_{ij} + \sum_{i'=1}^{i} p_{i'}. \end{aligned}$$

Case 2: J_j waits for more than p_i time units at stage S_i, i.e., $c_{ij} > c_{(i-1)j} + 2p_i$. By Lemma 10, this implies that $j > m_i b_i$ and that $J_{j-m_i b_i}$ is completed not before time $c_{ij} - p_i$. Then we conclude

$$\begin{aligned} c_{ij} &\leq c_{i(j-m_i b_i)} + p_i \\ &\overset{\text{ind.}}{\leq} c^*_{i(j-m_i b_i)} + p_i + \sum_{i'=1}^{i} p_{i'} \\ &\overset{(3)}{\leq} c^*_{ij} + \sum_{i'=1}^{i} p_{i'}. \end{aligned}$$

□

Using this, we obtain the desired competitiveness result.

Corollary 12 *With respect to makespan, total completion time, maximum flow time and total flow time, the Never-Wait algorithm to schedule a PFFB is a* 2*-competitive online algorithm.*

Proof Using Theorem 5, we obtain that there exists an optimal schedule ς^* that is a permutation schedule with job order $J_1, J_2, \dots, J_n$. Let ς be the schedule produced by the Never-Wait algorithm. Using Theorem 11 and Lemma 6, it follows for a fixed job index $j \in [n]$ that

$$\begin{aligned} C_j(\varsigma) &\le c^*_{sj} + \sum_{i=1}^{s} p_i \\ &\le C_j(\varsigma^*) + C_j(\varsigma^*) - r_j = 2C_j(\varsigma^*) - r_j, \end{aligned}$$

where the term $-r_j$ stems from $C_j(\varsigma^*) \ge c^*_{sj} \ge r_j + \sum_{i=1}^{s} p_i$. Thus, we have $C_j(\varsigma) \le 2C_j(\varsigma^*) - r_j \le 2C_j(\varsigma^*)$ and $F_j(\varsigma) = C_j(\varsigma) - r_j \le 2C_j(\varsigma^*) - 2r_j = 2F_j(\varsigma^*)$, which proves the desired statement. □

Corollary 12 and Theorem 4 imply the following corollary.

Corollary 13 *For the total flow time objective, there is no deterministic online algorithm which, in general, has a better competitive ratio than the Never-Wait algorithm.*

Next, we show by an example that the competitive ratio of the Never-Wait algorithm is not smaller than 2 with respect to any of the considered objectives.

Example 14 Consider the PFFB instance with only a single stage, with a fixed number $m_1 \ge 1$ of machines, a batch capacity $b_1 \ge 1$ to be chosen later, and processing time $p_1 = 1$. For some small $\varepsilon > 0$, suppose further there are $n = m_1 b_1$ jobs, with release dates $r_j = (j-1)\varepsilon$ for $j \le m_1$ and $r_j = m_1\varepsilon$ for $j \ge m_1 + 1$.

The Never-Wait algorithm schedules the first m_1 jobs as singleton batches as soon as they arrive, filling all machines. All other jobs are not started before time 1, when the first machine becomes idle again. Thus, none of the $n - m_1$ jobs J_j, $j \ge m_1 + 1$, can be finished before time 2. Hence, for the schedule ς produced by the Never-Wait algorithm, we obtain

$$\begin{aligned} C_{\max}(\varsigma) &\ge 2, \\ \sum C_j(\varsigma) &\ge 2n - m_1 = 2m_1 b_1 - m_1, \\ F_{\max}(\varsigma) &\ge 2 - m_1\varepsilon, \\ \sum F_j(\varsigma) &\ge 2n - m_1 - nm_1\varepsilon = 2m_1b_1 - m_1 - m_1^2 b_1 \varepsilon. \end{aligned}$$

In contrast, consider the feasible schedule in which all machines remain idle until time $m_1\varepsilon$, when the last job becomes available. At this point in time, all the $n = m_1 b_1$ jobs are partitioned into m_1 batches and started. For the resulting schedule ς', it follows that

$$\begin{aligned} C_{\max}(\varsigma') &\le 1 + m_1\varepsilon, \\ \sum C_j(\varsigma') &\le n + nm_1\varepsilon = m_1b_1 + m_1^2 b_1\varepsilon, \\ F_{\max}(\varsigma') &\le 1 + m_1\varepsilon, \\ \sum F_j(\varsigma') &\le n + nm_1\varepsilon = m_1b_1 + m_1^2 b_1\varepsilon. \end{aligned}$$

Now, given any $\alpha > 0$, we show that ε and b_1 can be chosen such that the ratio of the objective values is at least $2 - \alpha$. Without loss of generality, assume that $\frac{4-\alpha}{\alpha}$ is an integer. If this is not the case, make α continuously smaller, until it is. Let $b_1 = \frac{4-\alpha}{\alpha}$ and $\varepsilon = \frac{1}{m_1 b_1}$. We obtain

$$\begin{aligned} \frac{C_{\max}(\varsigma)}{C_{\max}(\varsigma')} &\ge \frac{2}{1+m_1\varepsilon} = \frac{2}{1+\frac{1}{b_1}} = \frac{2b_1}{b_1+1} \\ &= \frac{8-2\alpha}{4} = 2 - \frac{1}{2}\alpha > 2 - \alpha; \\ \frac{\sum C_j(\varsigma)}{\sum C_j(\varsigma')} &\ge \frac{2b_1 - 1}{b_1 + m_1 b_1 \varepsilon} = \frac{2b_1-1}{b_1+1} = \frac{8-3\alpha}{4} > 2-\alpha; \\ \frac{F_{\max}(\varsigma)}{F_{\max}(\varsigma')} &\ge \frac{2 - m_1\varepsilon}{1+m_1\varepsilon} = \frac{2-\frac{1}{b_1}}{1+\frac{1}{b_1}} = \frac{2b_1-1}{b_1+1} > 2-\alpha; \\ \frac{\sum F_j(\varsigma)}{\sum F_j(\varsigma')} &\ge \frac{2b_1 - 1 - m_1 b_1\varepsilon}{b_1 + m_1 b_1 \varepsilon} = \frac{2b_1-2}{b_1+1} \\ &= \frac{8-4\alpha}{4} = 2-\alpha. \end{aligned}$$

Hence, the Never-Wait algorithm does not have a competitive ratio less than 2 with respect to any of the considered objectives. Note that, while this example uses only a single stage, it can easily be extended to arbitrary many stages by using negligible processing times on all stages but the first one. Moreover, this example works for arbitrary values of m_1, which implies that the competitive ratio of the Never-Wait algorithm cannot be better than 2, no matter how many parallel machines per stage there are.

Remark 15 Even though the Never-Wait algorithm is in general not better than 2-competitive, concerning the makespan and total completion time objectives, Theorem 11 actually delivers a much stronger result than 2-competitiveness if the number of jobs is large.

To see this, we first show that for any $i \in [s]$, we have $c^*_{ij} \ge \lceil \frac{j}{m_i b_i} \rceil p_i =: \mathrm{LB}_{i,j}$. Indeed, with $k := \lceil \frac{j}{m_i b_i} \rceil - 1$ and $j' := j - k m_i b_i \ge 1$, recursive application of (3) yields

$$c^*_{ij} \overset{(3)}{\ge} kp_i + c^*_{ij'} \overset{(3)}{\ge} \lceil \tfrac{j}{m_i b_i} \rceil p_i + c^*_{(i-1)j'} \ge \mathrm{LB}_{i,j},$$

where the last inequality follows from $c^*_{(i-1)j'} \ge 0$.

Furthermore, $\mathrm{LB}_{i,j}$ tends to infinity for $j \to \infty$. Thus, in particular, if n tends to infinity, also c^*_{sn} tends to infinity, which is a lower bound for the makespan.

In contrast, note that the difference between the makespan of the schedule produced by the Never-Wait algorithm and the optimal makespan is at most $\sum_{i=1}^{s} p_i$ by Theorem 11. This difference stays constant if n tends to infinity.

Putting these things together, we obtain that, with respect to the makespan, the competitive ratio of the Never-Wait algorithm tends to 1, if n tends to infinity.

The same holds for the total completion time objective, as we argue now. Note that, with the same arguments as before, the total completion time of any feasible schedule is lower bounded by

$$\sum_{j=1}^{n} c^*_{sj} \geq \sum_{j=1}^{n} \mathrm{LB}_{s,j} \geq \sum_{j=1}^{n} \left\lceil \frac{j}{m_s b_s} \right\rceil p_s \in \Omega(n^2).$$

On the other hand, the difference between the total completion time of the schedule produced by the Never-Wait algorithm and the objective value of any optimal schedule is at most $n \sum_{i'=1}^{s} p_{i'} \in \mathcal{O}(n)$ by Theorem 11. Thus, also for the total completion time, we obtain that the competitive ratio of the Never-Wait algorithm tends to 1, if n tends to infinity.

However, a similar result cannot be achieved for our two flow time related objectives. To see this, note that Example 14 can be kind of "copied" arbitrarily often: given an instance with n jobs for which the Never-Wait algorithm achieves a competitive ratio of at least $2-\alpha$ for some $\alpha > 0$, introduce n more jobs J_{n+1} to J_{2n} with release dates $r_j = r_{j-n} + M$, $j = n+1, n+2, \dots, 2n$, for some large constant M. Then, both, the Never-Wait algorithm and an optimal offline algorithm, process the jobs J_{n+1} to J_{2n} in exactly the same way as they did process jobs J_1 to J_n, just shifted by M time steps. By the definition of flow times (in contrast to completion times), this keeps the maximum flow time constant, while the total flow time is doubled for both algorithms. Thus, with respect to these two objectives, the competitive ratio is still at least $2-\alpha$. This procedure can be repeated arbitrarily often. Hence, the competitive ratio of the Never-Wait algorithm does not tend to 1 for $n \to \infty$ with respect to flow time related objectives.

6.1 The Full-Batch algorithm

The Never-Wait algorithm can be seen as an extreme strategy, where all waiting time other than what is mandated by the scheduling constraints is avoided. Note that, for regular objective functions, it makes no sense to wait with starting a batch at stage S_i if a machine is available and already b_i jobs are waiting at stage S_i. Thus, it can be viewed as the opposite extreme to always wait until a full batch can be started.

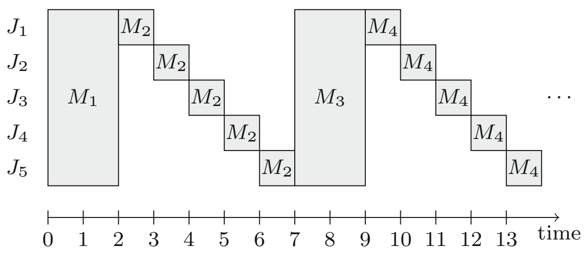

Fig. 4 Schedule ς up to stage S_4 produced by the Full-Batch algorithm for the case $n = 5$

Definition 16 The *Full-Batch algorithm* for scheduling a PFFB is defined by the following rule: At each stage, use only full batches (the last batch at stage S_i may be less than full, if the batch capacity b_i is not a divisor of the total number of jobs n).

Note that the Full-Batch algorithm is not actually an online algorithm in the strict sense, because in order to know when to start the last batch one needs to know the number of jobs in advance. This is not the case in the standard online setting. Therefore, we view the Full-Batch algorithm primarily as an offline approximation algorithm. It can also be seen as an online algorithm in a relaxed online setting, where the scheduler receives the information that no more jobs are coming when the last job is released.

Theorem 17 *There exists no $\alpha \geq 1$ such that the Full-Batch algorithm is an α-approximation for minimizing the makespan in a PFFB.*

Proof The statement already holds for usual PFBs, i.e., PFFBs where at each stage there is only one machine. Therefore, in the following, we assume $m_i = 1$ for all stages S_i, $i \in [s]$. For ease of notation, we identify each stage S_i with its single associated machine M_i.

Without loss of generality let $\alpha \geq 1$ be integer. Construct a PFB instance without release dates as follows. Set $s = 10\alpha$, $n = 5\alpha$, $p_{2i} = b_{2i} = 1$, $p_{2i-1} = 2$ and $b_{2i-1} = n$ for $i \in [5\alpha]$.

Let ς be the schedule produced by the Full-Batch algorithm. Fig. 4 illustrates ς on the first four machines for the case $n = 5$. An odd machine waits until all jobs have been finished on the previous even machine, before it processes all of them in a single batch. This way, we obtain $C_{\max}(\varsigma) = 10\alpha + 5\alpha n = 10\alpha + 25\alpha^2$, where the first term stems from the 5α odd machines and the second term from the 5α even machines.

In contrast, let ς' be the schedule where all batches consist of a single job only. On the first machine, each job is started two time steps after the previous job, i.e., J_j is started at time $2j - 2$. On all remaining machines, jobs can be started immediately upon arrival, since no processing time is larger than two. Fig. 5 illustrates ς' on the first four machines for

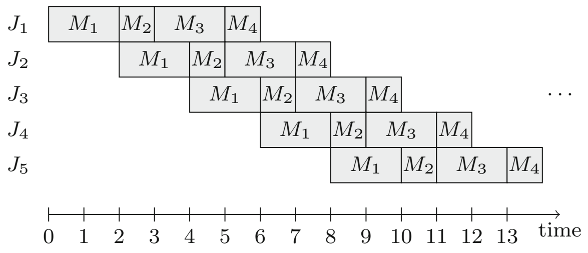

Fig. 5 Schedule ς' up to stage S_4 produced by using only batches of size one for the case $n = 5$

the case $n = 5$. We obtain

$$C_{\max}(\varsigma') = 2n - 2 + 15\alpha = 25\alpha - 2$$

because it takes $2n - 2$ time steps until the last job is started on the first machine, and 15α more time steps for processing the last job on all machines.

Hence, in total, we obtain

$$\frac{C_{\max}(\varsigma)}{C_{\max}(\varsigma')} = \frac{10\alpha + 25\alpha^2}{25\alpha - 2} > \frac{25\alpha^2}{25\alpha} = \alpha.$$

Therefore, the Full-Batch algorithm cannot be an α-approximation for any constant $\alpha \geq 1$. □

Remark 18 Note that Hertrich (2018) shows for the case of PFBs that the example in the proof of Theorem 17 is no longer valid if either the number of jobs n or the number of stages s is fixed and no longer depends on α. In these cases, the Full-Batch algorithm becomes a constant factor approximation. Analyzing the proofs of Hertrich (2018), one can see that these results carry over to PFFBs.

7 Optimal online algorithm for two stages

We have seen that the competitive ratio of the Never-Wait algorithm is 2 with respect to all four objective functions considered in this paper. Comparing with the lower bounds of Sect. 3, this is best possible for the total flow time objective. However, for the other three objectives, there is a gap between the lower bound of the golden ratio φ and the upper bound of 2. In this section, we close this gap in the special case of $s \leq 2$ for makespan and total completion time by presenting a specialized φ-competitive algorithm for this case. This extends the result of Zhang et al. (2003), who provide a φ-competitive algorithm for makespan minimization with $s = 1$, i.e., on identical, parallel batching machines.

Let $t = \varphi p_1 + (\varphi - 1)p_2$. This is the latest possible time at which the first batch must be started at the second stage if we want that a job released at time zero is completed at time $\varphi(p_1 + p_2)$, which is the minimal completion time of such a job multiplied with φ. The idea of the following algorithm is to schedule S_1 in a way such that as many jobs as possible have completed S_1 at time t, while the machines of S_2 stay idle until time t and are scheduled according to the Never-Wait algorithm afterwards.

Definition 19 For a 2-stage PFFB the *t-Switch* algorithm is defined as follows. Let the set I of *starting instants* consist of those points in time $\tau \geq 0$, for which $\tau + \ell p_1 = t$ for some integer $\ell \in \mathbb{Z}$, i.e.,

$$I = \left\{ t - \left\lfloor \frac{t}{p_1} \right\rfloor p_1,\ t - \left(\left\lfloor \frac{t}{p_1} \right\rfloor - 1 \right) p_1, \ldots, \right.$$
$$\left. t - p_1, t, t + p_1, \ldots \right\}.$$

At stage S_1, jobs are started only at starting instants. At each starting instant, as many jobs are started as possible, i.e., the minimum of $m_1 b_1$ and the number of available jobs. The machines of S_2 stay idle until time t and are scheduled according to the Never-Wait algorithm afterwards.

Next, we prove three lemmas that help to show φ-competitiveness. In the following, let c_{1j} and c_{2j}, $j \in [n]$, be the completion times produced by the t-Switch algorithm and let c^*_{1j}, c^*_{2j}, $j \in [n]$, be the lower bounds of Sect. 5.

Lemma 20 *For all $j \in [n]$, it holds that $c_{1j} \leq c^*_{1j} + p_1$.*

Proof We use induction on j. Fix a job index $j \in [n]$. Note that J_j is started at the starting instant $\tau = c_{1j} - p_1$. Let $\tau' = c_{1j} - 2p_1$ be the previous starting instant. First, suppose that $r_j > \tau' = c_{1j} - 2p_1$, i.e., τ is the first starting instant after r_j. Note, in particular, that since $m_1 b_1$ jobs can be started at each starting instant, this is always the case if $j \leq m_1 b_1$. In this case, we directly obtain $c_{1j} \leq r_j + 2p_1 \leq c^*_{1j} + p_1$.

Otherwise, if $r_j \leq \tau' = c_{1j} - 2p_1$, then J_j was already released at starting instant τ', but has not been started at this time. Hence, $j > m_1 b_1$ and exactly $m_1 b_1$ other jobs must have been started at τ'. This implies $c_{1(j-m_1 b_1)} \geq c_{1j} - p_1$ and, hence,

$$\begin{aligned} c_{1j} &\leq c_{1(j-m_1 b_1)} + p_1 \\ &\overset{\text{ind.}}{\leq} c^*_{1(j-m_1 b_1)} + 2p_1 \\ &\overset{(3.)}{\leq} c^*_{1j} + p_1 \end{aligned}$$

□

Lemma 21 *For all $j \in [n]$ with $c_{1j} \leq t$, it holds that $c_{2j} \leq c^*_{2j} + (\varphi - 1)(p_1 + p_2)$.*

Proof We use induction on j. Suppose first that J_j is started at stage S_2 exactly at time t. Note that this is always the case if $j \leq m_2 b_2$, as stage S_2 is idle before time t and thus up to $m_2 b_2$ jobs can be started at time t (recall that it is assumed that J_j is finished at S_1 before time t). We obtain

$$\begin{aligned} c_{2j} &= t + p_2 = \varphi(p_1 + p_2) \\ &= (p_1 + p_2) + (\varphi - 1)(p_1 + p_2) \\ &\leq c^*_{2j} + (\varphi - 1)(p_1 + p_2). \end{aligned}$$

On the other hand, consider the case in which J_j is started at stage S_2 later than time t. This can only happen if all machines of S_2 continuously process full batches between time t and time $c_{2j} - p_2$. In particular, this implies $j > m_2 b_2$ and $c_{2(j-m_2b_2)} \geq c_{2j} - p_2$. Hence,

$$\begin{aligned} c_{2j} &\leq c_{2(j-m_2b_2)} + p_2 \\ &\overset{\text{ind.}}{\leq} c^*_{2(j-m_2b_2)} + p_2 + (\varphi - 1)(p_1 + p_2) \\ &\overset{(3)}{\leq} c^*_{2j} + (\varphi - 1)(p_1 + p_2). \end{aligned}$$

□

Lemma 22 *For all $j \in [n]$, it holds $c_{2j} < c^*_{2j} + p_1 + p_2$.*

Proof If $c_{1j} \leq t$, then the claim for this index j follows by Lemma 21. If $c_{1j} > t$, then S_2 is already scheduled according to the Never-Wait algorithm when J_j arrives. Hence, the claim can be proven analogously to Theorem 11, making use of Lemma 20. □

Now we are ready to prove φ-competitiveness of the t-Switch algorithm.

Theorem 23 *For a two-stage PFFB, the t-Switch algorithm is φ-competitive with respect to the two objective functions $C_{\max}$ and $\sum C_j$.*

Proof Consider a job J_j, $j \in [n]$. We distinguish two cases:
Case 1: $c^*_{1j} < t$. Using Lemma 20, it follows that $c_{1j} < t + p_1$. Since c_{1j} must be a starting instant, we even obtain $c_{1j} \leq t$. Now Lemma 21 yields

$$c_{2j} \leq c^*_{2j} + (\varphi - 1)(p_1 + p_2) \leq c^*_{2j} + (\varphi - 1)c^*_{2j} = \varphi c^*_{2j}.$$

Case 2: $c^*_{1j} \geq t$. Then it follows that $c^*_{2j} \geq t + p_2 = \varphi(p_1 + p_2)$. Using Lemma 22, we obtain

$$c_{2j} \leq c^*_{2j} + p_1 + p_2 \leq c^*_{2j} + \frac{1}{\varphi} c^*_{2j} = \varphi c^*_{2j}.$$

Having proven $c_{2j} \leq \varphi c^*_{2j}$ for all $j \in [n]$, the φ-competitiveness follows for $C_{\max}$ and $\sum C_j$. □

Theorem 23 in combination with Theorems 2 and 3 implies the following corollary.

Corollary 24 *For PFFBs with $s = 2$ stages and the makespan or total completion time objective, there is no deterministic online algorithm which, in general, has a better competitive ratio than the t-Switch algorithm.*

8 Conclusion

In this paper, we consider proportionate flexible flow shops with batching machines (PFFBs). We put a special focus on the online version of the problem, which is highly relevant for applications in the production of modern, individualized medicaments. To the best of our knowledge, the online version has not been studied before, not even in the special case of proportionate (non-flexible) flow shops with batching machines (PFBs).

We describe and analyze two algorithms: the very general Never-Wait algorithm and the more specialized t-Switch algorithm. The Never-Wait algorithm works for an arbitrary number of stages and machines. What is more, its description is relatively simple and therefore it is easy to implement in practice. We show that, despite its simplicity, the Never-Wait algorithm is 2-competitive for minimizing the makespan, total completion time, maximum flow time, and total flow time. Furthermore, we show that for the total flow time objective, no deterministic online algorithm can, in general, do better than the Never-Wait algorithm.

Note that the total flow time, which is equivalent with the average flow time by dividing by the constant number of jobs, is particularly important for our application: it measures the average time patients have to wait for their medicament after production is ordered. Obviously, a low average waiting time is necessary for patients to benefit from the medicine as quickly as possible. Interestingly, studying a particular industrial instance, Ackermann et al. (2020) have also done some initial work to confirm that the theoretical usefulness of the Never-Wait algorithm is also coherent with its practical behavior.

The t-Switch algorithm is specialized for PFFBs with only two stages and the makespan or total completion time objective. For these versions of the online problem, the t-Switch algorithm is a φ-competitive algorithm, with $\varphi = \frac{1+\sqrt{5}}{2}$, the golden ratio. By using and extending lower bounds known from the literature, we show that no deterministic online algorithm to minimize the makespan or total completion time in PFFBs with two stages can, in general, be better than the t-Switch algorithm.

Both results are based on the observation that for all objectives we consider, there exists an optimal permutation schedule with start times of jobs ordered by a non-decreasing release date ordering on all stages. We show this as an extension to the theorem proved by Hertrich et al. (2020).

Notice that for the offline version, our results imply that the Never-Wait algorithm is a 2-approximation algorithm for PFFBs to minimize the makespan, total completion time, maximum flow time or total flow time. This is interesting as so far, for PFFBs with arbitrarily many stages, no exact polynomial algorithm has been found even in the case where all stages consist of only one machine (see, e.g., Hertrich et al. (2020)).

As this is the first study of online PFFBs, it is natural that some open questions remain. Most importantly, there remains a gap between the competitiveness of the Never-Wait algorithm and the lower bound of competitiveness in the case of three or more stages for makespan and total completion time, and even in the case of two stages, for the maximum flow time. Of course, it would be desirable to close this gap, either by proving a larger lower bound of competitiveness or by finding a better algorithm than the Never-Wait algorithm. As we have shown in Example 14, the Never-Wait algorithm itself cannot be better than 2-competitive in general. One way to improve the Never-Wait algorithm could be to better forecast what happens on the later stages of the PFFB. Observe that, despite the online situation, we can forecast job arrivals on later stages once the jobs become available at the first stage. Indeed, at any time step t, the full schedule (and thus arrival forecast) at each stage can be computed for all jobs which become available before t. Thus, for the later stages in the PFFB, the Never-Wait algorithm might be improved by using this additional knowledge of future job arrivals. Note that in the practical study mentioned above, such an improvement has been successfully attempted for a specific problem instance (see Ackermann et al. (2020)).

In addition to closing these gaps, for the total completion time and total flow time objectives, it might be possible to achieve better competitive ratios for special cases where a certain minimum number of machines per stage is guaranteed. Observe that the lower bound constructions in Sect. 3 (Theorems 3 and 4) only use a single machine. It might be the case that the same lower bounds do no longer hold if each stage contains several parallel machines. Possibly, better competitive ratios dependent on $m := \min_{i\in[s]} m_i$ could be established. Cao et al. (2011) provide a result of this kind for parallel machines (only one stage) with unbounded batch capacity. For the makespan and maximum flow time objectives, however, the lower bounds by Zhang et al. (2003) and Jiao et al. (2014) involve arbitrarily many parallel machines. Hence, for these objectives, it is not possible to achieve better competitive ratios for high values of m.

Another open question concerns different objective functions. We have shown that in a schedule computed by the Never-Wait algorithm the finishing time of a job J_j is at most twice the least possible finishing time of J_j in any permutation schedule ordered by release dates. Unfortunately, for most traditional scheduling objectives beyond those studied in this paper, permutation schedules ordered by release dates are not, in general, optimal (see, e.g., Hertrich et al. (Hertrich et al. (2020), Example 4)). Still, the Never-Wait strategy may help with these objective functions, if jobs are prioritized differently. For example, instead of scheduling jobs in order of their release dates, each time a machine is started in the Never-Wait algorithm, one may instead pick the available jobs with the largest weight, if the objective involves job weights. It is, at the moment, unclear whether such an algorithm may be competitive and, if yes, what its competitiveness bound would be.

On the other hand, in the online scenario, it may be valid to restrict the study of other objectives to the case where jobs are ordered by their release dates on all stages. In other words, instead of searching for an optimal schedule amongst all possible schedules, we search for an optimal schedule amongst all permutation schedules with jobs ordered by release dates. Especially in the pharmacological application we consider, such a first-in-first-out approach may be mandated due to ethical and fairness considerations. For this type of restricted problem, the Never-Wait algorithm may well prove to be competitive for many objectives beyond the ones considered in this paper, as long as these objectives are regular. For example, using the same arguments as before, the Never-Wait algorithm is 2-competitive w.r.t. the weighted total completion time / flow time. Indeed, as for the non-weighted versions, the factor 2 from Theorem 11 can be moved in front of the sum objectives to immediately see 2-competitiveness.

Funding Open Access funding enabled and organized by Projekt DEAL. The research of Christoph Hertrich was supported by the Felix-Klein-Zentrum für Mathematik e.V. and by the DFK-GRK 2434 Facets of Complexity.

Appendix A: proof of theorem 5

We now show how to prove Theorem 5. First, analogous to Lemma 2 of Hertrich et al. (2020), we need the following lemma.

Lemma 25 *Let ς be a feasible schedule for a PFFB and let π be some earliest release date ordering of the jobs. Then* *there exists a feasible permutation schedule $\hat{\varsigma}$ in which the jobs are ordered by π and the multi-set of job completion times in $\hat{\varsigma}$ is the same as in ς.*

Proof The proof is completely analogous to the proof of Lemma 2 from Hertrich et al. (2020). To see this, one only needs to note that nowhere in the proof from Hertrich et al. (2020) it is actually needed that all batches $B_\ell^{(i)}$ at stage S_i are processed on the same machine. Numbering the batches at stage S_i in any start time order, the construction of the new schedule $\hat{\varsigma}$ as well as the proof of its feasibility work exactly as in the proof from Hertrich et al. (2020). □

Now we are ready to proof the main theorem.

Proof of Theorem 5 Let ς be an optimal PFFB schedule with respect to one of our four objective functions $C_{\max}$, $\sum C_j$, $F_{\max}$, and $\sum F_j$. Let π be an earliest release date ordering. Using Lemma 25, construct a new permutation schedule $\hat{\varsigma}$, with jobs ordered by π on all stages and with the same multi-set of job completion times.

For objective functions $C_{\max}$ and $\sum C_j$, clearly the new schedule $\hat{\varsigma}$ is optimal, since ς is optimal and $\hat{\varsigma}$ has the same multi-set of job completion times. Moreover, since $\sum F_j = \sum C_j - c$, where c is a constant given by $c = \sum_{i=1}^n r_i$, the same argument holds for objective function $\sum F_j$.

Finally, for objective function $F_{\max}$, suppose that jobs are indexed according to the earliest release date ordering π, i.e., $r_1 \leq r_2 \leq \cdots \leq r_n$ and $C_1(\hat{\varsigma}) \leq C_2(\hat{\varsigma}) \leq \cdots \leq C_n(\hat{\varsigma})$. Let j be the index of the job with maximum flow time in $\hat{\varsigma}$, i.e., $F_{\max}(\hat{\varsigma}) = C_j(\hat{\varsigma}) - r_j$. Since ς has the same multi-set of job completion times as $\hat{\varsigma}$, there exist at most $j-1$ jobs with a completion time strictly less than $C_j(\hat{\varsigma})$ in the original schedule ς. On the other hand, all the j jobs $J_1, J_2, \ldots, J_j$ have a release date of at most r_j. Hence, by pigeon-hole principle, there must exist a job $J_{j'}$ with $C_{j'}(\varsigma) \geq C_j(\hat{\varsigma})$ and $r_{j'} \leq r_j$. This implies

$$F_{\max}(\varsigma) \geq C_{j'}(\varsigma) - r_{j'} \geq C_j(\hat{\varsigma}) - r_j = F_{\max}(\hat{\varsigma}).$$

□

References

Ackermann, H., Heydrich, S., & Weiß, C. (2020). Analyzing and optimizing the throughput of a pharmaceutical production process. *Operations Research Proceedings, 2019*, 591–597.

Ahmadi, J. H., Ahmadi, R. H., Dasu, S., & Tang, C. S. (1992). Batching and scheduling jobs on batch and discrete processors. *Operations Research, 40*(4), 750–763.

Amin-Naseri, M. R., & Beheshti-Nia, M. A. (2009). Hybrid flow shop scheduling with parallel batching. *International Journal of Production Economics, 117*(1), 185–196.

Brucker, P. (2007). *Scheduling Algorithms* (5th ed.). Springer.

Cao, J., Yuan, J., Li, W., & Bu, H. (2011). Online scheduling on batching machines to minimise the total weighted completion time of jobs with precedence constraints and identical processing times. *International Journal of Systems Science, 42*(1), 51–55.

Chai, X., Li, W., Zhu, Y. (2019). Online scheduling to minimize maximum weighted flow-time on a bounded parallel-batch machine. *Annals of Operations Research.*

Chen, B., Deng, X., & Zang, W. (2004). On-line scheduling a batch processing system to minimize total weighted job completion time. *Journal of Combinatorial Optimization, 8*(1), 85–95.

Deng, X., Poon, C. K., & Zhang, Y. (2003). Approximation algorithms in batch processing. *Journal of Combinatorial Optimization, 7*(3), 247–257.

Fang, Y., Liu, P., & Lu, X. (2011). Optimal on-line algorithms for one batch machine with grouped processing times. *Journal of Combinatorial Optimization, 22*(4), 509–516.

Graham, R. L., Lawler, E. L., Lenstra, J. K., & Rinnooy Kan, A. H. G. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. In P. L. Hammer, E. L. Johnson, & B. H. Korte (Eds.), *Discrete Optimization II, Annals of Discrete Mathematics* (Vol. 5, pp. 287–326). Elsevier.

Hertrich C (2018) Scheduling a proportionate flow shop of batching machines. Master thesis, Technische Universität Kaiserslautern, http://nbn-resolving.de/urn:nbn:de:hbz:386-kluedo-54968

Hertrich, C., Weiß, C., Ackermann, H., Heydrich, S., & Krumke, S. O. (2020). Scheduling a proportionate flow shop of batching machines. *Journal of Scheduling, 23*(5), 575–593.

Jiao, C., Li, W., & Yuan, J. (2014). A best possible online algorithm for scheduling to minimize maximum flow-time on bounded batch machines. *Asia-Pacific Journal of Operational Research, 31*(04), 1450030.

Li, D., Meng, X., Liang, Q., & Zhao, J. (2015). A heuristic-search genetic algorithm for multi-stage hybrid flow shop scheduling with single processing machines and batch processing machines. *Journal of Intelligent Manufacturing, 26*(5), 873–890.

Li, W., & Yuan, J. (2011). Online scheduling on unbounded parallel-batch machines to minimize maximum flow-time. *Information Processing Letters, 111*(18), 907–911.

Li, W. H., & Chai, X. (2018). Online scheduling on bounded batch machines to minimize the maximum weighted completion time. *Journal of the Operations Research Society of China, 6*(3), 455–465.

Lin, R., Li, W., Chai, X. (2019). On-line scheduling with equal-length jobs on parallel-batch machines to minimise maximum flow-time with delivery times. *Journal of the Operational Research Society.*

Luo, H., Huang, G. Q., Zhang, Y. F., & Dai, Q. Y. (2011). Hybrid flowshop scheduling with batch-discrete processors and machine maintenance in time windows. *International Journal of Production Research, 49*(6), 1575–1603.

Ma, R., Wan, L., Wei, L., & Yuan, J. (2014). Online bounded-batch scheduling to minimize total weighted completion time on parallel machines. *International Journal of Production Economics, 156*, 31–38.

Panwalkar, S. S., Smith, M. L., & Koulamas, C. (2013). Review of the ordered and proportionate flow shop scheduling research. *Naval Research Logistics (NRL), 60*(1), 46–55.

Pinedo, M. (2012). *Scheduling* (5th ed.). Springer.

Sung, C. S., & Yoon, S. H. (1997). Minimizing maximum completion time in a two-batch-processing-machine flowshop with dynamic arrivals allowed. *Engineering Optimization, 28*(3), 231–243.

Sung, C. S., Kim, Y. H., & Yoon, S. H. (2000). A problem reduction and decomposition approach for scheduling for a flowshop of batch processing machines. *European Journal of Operational Research, 121*(1), 179–192.

Tan, Y., Mönch, L., & Fowler, J. W. (2018). A hybrid scheduling approach for a two-stage flexible flow shop with batch processing machines. *Journal of Scheduling, 21*(2), 209–226.

Tian, J., Fu, R., & Yuan, J. (2014). Online over time scheduling on parallel-batch machines: A survey. *Journal of the Operations Research Society of China, 2*(4), 445–454.

Zhang, G., Cai, X., & Wong, C. K. (2001). On-line algorithms for minimizing makespan on batch processing machines. *Naval Research Logistics, 48*(3), 241–258.

Zhang, G., Cai, X., & Wong, C. (2003). Optimal on-line algorithms for scheduling on parallel batch processing machines. *IIE Transactions, 35*(2), 175–181.