Journal of Combinatorial Optimization (2019) 38:224–253
https://doi.org/10.1007/s10878-019-00381-6

Online interval scheduling on two related machines: the power of lookahead

Nicolas Pinson[1] · **Frits C. R. Spieksma**[2]

Published online: 9 January 2019

Abstract
We consider an online interval scheduling problem on two related machines. If one machine is at least as twice as fast as the other machine, we say the machines are *distinct*; otherwise the machines are said to be *similar*. Each job $j \in J$ is characterized by a length p_j, and an arrival time t_j; the question is to determine whether there exists a feasible schedule such that each job starts processing at its arrival time. For the case of unit-length jobs, we prove that when the two machines are distinct, there is an amount of lookahead allowing an online algorithm to solve the problem. When the two machines are similar, we show that no finite amount of lookahead is sufficient to solve the problem in an online fashion. We extend these results to jobs having arbitrary lengths, and consider an extension focused on minimizing total waiting time.

Keywords Online algorithms · Interval scheduling · Lookahead · Competitive ratio

1 Introduction

Consider the following problem. We are given a finite set of jobs $J = \{1, 2, \ldots, n\}$, and two related machines called M_1 and M_2. Each job $j \in J$ must start at a given *arrival time* t_j, where $0 \equiv t_1 \leq t_2 \leq \cdots \leq t_n$, and each job $j \in J$ has *length* p_j. The two machines have respective *speeds* s_1 and s_2, with the convention that the first machine is faster than the second one (i.e., $s_1 > s_2$). Since, in our context, it is more intuitive to consider times rather than inverses of speeds, we use so-called *standard processing times* $T_i = \frac{1}{s_i}$ for $i = 1, 2$. Thus, it requires $p_j \times T_i$ time-units to process

✉ Frits C. R. Spieksma
f.c.r.spieksma@tue.nl

Nicolas Pinson
nicolas.pinson@ens-lyon.fr

[1] ENS-Lyon, Lyon, France

[2] Department of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands

job $j \in J$ on machine i ($i = 1, 2$). Each job $j \in J$ must be assigned to either machine M_1 or M_2; the resulting *schedule* is feasible if and only if there is no overlap between any pair of jobs assigned to the same machine; in other words, for each pair of distinct jobs $j_1, j_2 \in J$, with $j_1 \leq j_2$, assigned to a same machine i, $t_{j_2} \geq t_{j_1} + p_{j_1} T_i$ ($i = 1, 2$). An instance of this problem is called *feasible* if a feasible schedule exists, otherwise the instance is called *infeasible*.

Our focus is on the existence of online algorithms that decide whether a given instance is feasible. In a classical online algorithm, the jobs and their lengths are not known in advance, and the existence of job j and its length p_j is only revealed at time t_j. At that moment, the online algorithm has to assign the job to either machine M_1, or machine M_2, or report failure. Here, we want to understand to what extent partial knowledge of the future can help an online algorithm in order to decide feasibility. Thus, we employ a parameter τ, called the *look-ahead*, so that at time t all arrival times of jobs arriving in $[t, t + \tau]$, and their corresponding lengths, are known to the algorithm. More formally, for any duration $\tau \geq 0$, the phrase *an online algorithm with lookahead time* τ refers to an algorithm that has to decide at the arrival time of job j, i.e., at time t_j, to which machine job j should be assigned, while knowing only what happened before t_j, as well as the arrival times and the lengths of the jobs arriving in the interval $[t_j, t_j + \tau]$. We say that an online algorithm with look-ahead τ *exists* if the algorithm constructs, in an online fashion, a feasible schedule whenever one exists. The interval $[t_j, t_j + \tau]$ is called the *lookahead interval* with respect to t_j.

This section is structured as follows. We give a concise overview of the literature on online interval scheduling in Sect. 1.1. Subsection 1.2 describes the practical application motivating this work, and Sect. 1.3 summarizes our results.

1.1 Literature

A defining characteristic of interval scheduling problems is that the starting times of the jobs are given, see Kolen et al. (2007) for a general survey. Thus, jobs can be represented by intervals and they require uninterrupted processing, so that two intervals that overlap cannot be assigned to a same machine. An interval is said to be accepted if it is entirely processed by a machine, while an interval that is not entirely processed, is lost. It is usually allowed to interrupt the processing of an interval to process another one. In that case, the interrupted interval is lost: its processing can not be resumed later on. One often wants to maximize the number of intervals accepted, or if jobs have weights, one wants to maximize the sum of the weights of accepted jobs. Online algorithms for interval scheduling problems have been studied since Lipton and Tomkins (1994); as described above the jobs and their lengths are not known in advance, and a decision about a job needs to be made at the instant it arrives. We refer to Sgall (1998) for a survey on online algorithms for scheduling problems, and for definitions of relevant terminology.

For a single machine, Woeginger (1994) presents an online algorithm that outputs a solution with a value at least $\frac{1}{4}$ of the optimal weight under various conditions (including the case of unit-length jobs with arbitrary weights), i.e., he gives an algorithm achieving a competitive ratio of 4. Fung et al. (2014) achieve a competitive ratio

of 2 using randomization, see also earlier work of Fung et al. (2008), Fung et al. (2012) and Epstein and Levin (2010). Recent work on online interval scheduling on two and three machines can be found in Yu and Jacobson (2018). In the context of a single machine, jobs of unit length and arbitrary weights, Zheng et al. (2013) investigate the impact of lookahead. They find that a lookahead of one time-unit serves as a threshold: a lookahead of less than one time-unit does not lead to the existence of algorithms with a better competitive ratio, while a lookahead of at least one time-unit does.

For a fixed number k of identical independent machines, Faigle and Nawijn (1995) and Carlisle and Lloyd (1995) present an online algorithm that maximizes the number of jobs accepted (even if jobs have arbitrary length). Krumke et al. (2011) allow the machines to be distinct and show that the decision problem derived from maximizing the number of jobs accepted (with machines having arbitrary speeds) is strongly NP-complete. Dosa et al. (1994) consider an online scheduling problem with two related machines where the goal is to minimize makespan allowing rearrangement of jobs.

In a recent paper, Epstein et al. (2016) study online interval scheduling with related machines, where the jobs (or intervals) have a length, and a weight, and they present lower and upper bounds on competitive ratios of algorithms that aim at maximizing total weight of accepted intervals. In particular, for the case of two related machines, where jobs have unit length, as well as unit weight, they give an online algorithm achieving a competitive ratio of $\frac{4}{3}$, which they show to be best-possible. Other results in Epstein et al. (2016) include a matching lower and upper bound of k (where k is the number of machines) for the case of arbitrary lengths, and unit weights [correcting a claim in Krumke et al. (2011)].

Notice that, in contrast to the objective functions considered in those papers, we focus on a more modest question, namely the decision problem. Indeed, instead of maximizing total weight, we are only interested in the question whether there exists an online algorithm (with a certain amount of lookahead) that is able to accept all intervals if the instance allows so. The results of Epstein et al. (2016) imply that, without any additional 'power' for the online algorithm (such as lookahead), the answer to this question is no. To the best of our knowledge, the setting with lookahead, i.e., the setting where at a decision moment, some future jobs are known, has not been studied in the context of more than one machine in the field of interval scheduling.

However, when considering scheduling problems where jobs do not have fixed starting times, a sizable literature on the impact of lookahead exists. For instance, Schwarz (2008) considers a setting where advance warnings considering the (non-)availability of a machine are given. Further, in Li et al. (2009) lookahead is considered for a problem involving a (parallel) batching machine. We also mention Erlebach and Spieksma (2003) and Miyazawa and Erlebach (2004) who consider an online interval scheduling problem where the intervals are revealed in the order of their right endpoints (which can be interpreted as a particular form of lookahead).

The off-line version of a special case of our problem (namely, the setting with unit-length jobs) is studied in Passchyn et al. (2016). They provide necessary and sufficient conditions for the feasibility of a given instance. Based on this characteriza-

tion (which we describe in Sect. 2.2), an $O(n)$ algorithm is given to solve the problem with unit-length jobs. In addition, they describe an $O(n^2)$ algorithm to solve the so-called bidirectional case with two machines, and also provide an $O(mn^m)$ dynamic programming algorithm to solve the bidirectional problem with m machines; all these results pertain to the off-line problem.

1.2 Motivation

Our problem is relevant in the context of handling ship traffic in inland waterways. In such waterways, locks are very often used to allow ships to overcome changes in the water level; more often than not, locks have multiple chambers that operate independently, and each chamber is able to transfer ships to the other side. Lock scheduling is receiving an increasing amount of attention, especially due to the growing relevance of inland waterway transport as a sustainable, cheap, emission-friendly, and safe alternative to transport over land. We refer to Hermans (2014), Smith et al. (2011) and Passchyn et al. (2016) who study single-chamber locks, and to Prandtstetter et al. (2015), Disser et al. (2015) and Passchyn et al. (2016), where series of locks are studied.

We claim that operating a single lock with two distinct chambers and identical ships can be modeled as a scheduling problem by seeing ships as jobs and chambers as machines. In that case, we set $p_j = 1$ for every job (ship) $j \in J$, and T_1 and T_2 represent the so-called *lockage times* of the chambers (the *lockage time* is the time needed by a chamber to let a ship enter, change the water level, and let the ship exit). We are interested in the question whether a schedule exists in which no job (ship) has to wait.

Online algorithms with lookahead are especially relevant in this setting, since the person responsible for operating a lock (the *lockmaster*) may, on the one hand, know some time in advance that a ship is going to arrive, but, on the other hand, does not know all arrival times that will realize during a day of operation. In particular, we are aware of a situation (along the river Main, Würzburg, Germany) where the lockmaster has access to cameras that describe the situation one lock upstream, as well as one lock downstream. Clearly, such a situation can be modelled by choosing an appropriate value for the look-ahead τ.

1.3 Our results

We focus on the power of lookahead for this interval scheduling problem.

(i) For the case of unit-length jobs, we show that there exists an online algorithm with lookahead $2T_1$ if and only if the ratio between the two standard processing times is at least 2 (Sect. 3); in addition, we show that there cannot exist an online algorithm with lookahead less than $2T_1$.
(ii) We generalize these results to the case where jobs have arbitrary lengths (Sect. 4).
(iii) Then, in Sect. 5, we investigate whether our results extend to a situation where the objective is to minimize total waiting time.

2 Preliminaries

2.1 A graph reformulation

A key tool in our analysis is the following undirected graph that we build from a given instance $\mathcal{I}$ defined by arrival times $(t_j)_{1\leq j\leq n}$, T_1, and T_2 as follows. Let $G(\mathcal{I}) = (V, E)$ where there is a node in V for each job $j \in J$, i.e., $V = J$. Observe that the sequence of the jobs implied by sorting their arrival times in nondecreasing order, and breaking ties arbitrarily, gives an order of the nodes of V. The edgeset E is the disjoint union of two sets E_1 and E_2 as follows. Let $j_1, j_2 \in V$ with $j_1 < j_2$: $(j_1, j_2) \in E_1 \Leftrightarrow t_{j_2} - t_{j_1} < p_{j_1} \times T_1$ and $(j_1, j_2) \in E_2 \Leftrightarrow t_{j_2} - t_{j_1} < p_{j_1} \times T_2$. We call an edge in E_1 (E_2) a 1-edge (2-edge). Hence, a 1-edge implies that the two corresponding jobs cannot both be assigned to machine M_1, while a 2-edge means that the two corresponding jobs cannot both be assigned to machine M_2. Moreover, a *1-triangle* will stand for a triangle of 1-edges in $G(\mathcal{I})$. In terms of graphs, deciding whether an instance $\mathcal{I}$ is feasible is equivalent to deciding whether the corresponding graph $G(\mathcal{I}) = (V, E)$ allows a partition of V into two sets V_1 and V_2 such that V_1 is an independent set in (V, E_1) and V_2 is an independent set in (V, E_2).

The previous definitions have several consequences for the structure of the graph. A first one is that $E_1 \subseteq E_2$. A second one is that the existence of an edge between j_1 and j_2 (with $j_1 < j_2$) crucially depends on the length p_{j_1}. Indeed, if there are three nodes j_1, j_2, j_3 such that $j_1 < j_2 < j_3$, the existence of $(j_1, j_3) \in E_1$ (E_2), implies that $(j_1, j_2) \in E_1$ (E_2). Notice however, that it does not imply that $(j_2, j_3) \in E_1$ (E_2).

For the sake of readability of our figures, we will not represent all the edges of the graph but only the *maximal* edges, i.e., those which are not implied by another represented edge. For example, keep in mind that if two nodes are connected by a 1-edge, they are also connected by a 2-edge. In the sequel, we will represent 2-edges by segments in the form of arcs above the line of the nodes (⌒) whereas we will represent 1-edges either by straight line segments (—), or by segments in the form of arcs below the line of the nodes (⌣).

2.2 Characterizing the off-line case for unit-length jobs

In this subsection, we recall the off-line characterization of Passchyn et al. (2016) for the case of unit-length jobs, i.e., for the case where $p_j = 1$ for all $j \in J$. This assumption imposes additional structure on the graph $G(\mathcal{I})$: if two nodes j_1 and j_2 are connected by an edge of any kind, then any node whose arrival times lies in $[t_{j_1}, t_{j_2}]$ is connected to j_1 as well as j_2 by an edge of the same kind. Now, let us recall the characterization of feasible instances in this setting.

Definition 1 Given is an instance $\mathcal{I}$ and its graph $G(\mathcal{I}) = (V, E)$. A *bad path* is any sequence of distinct nodes $(j_1, j_2, \ldots, j_k)$ with $k \geq 4$ and k even, satisfying:

1. The nodes in the sequence appear in the order defined on V, with exception of j_1 and j_k, which satisfy $j_2 < j_1 < j_3$ and $j_{k-2} < j_k < j_{k-1}$. More formally: $j_x < j_{x+1}$ for all $x \in \{2, \ldots, k-2\}$, $j_2 < j_1 < j_3$ and $j_{k-2} < j_k < j_{k-1}$.

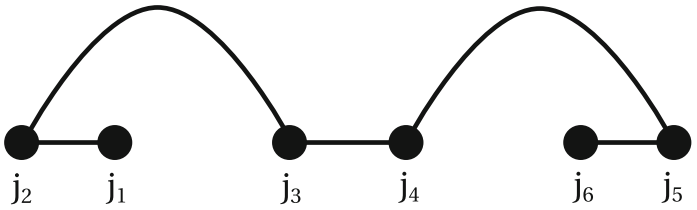
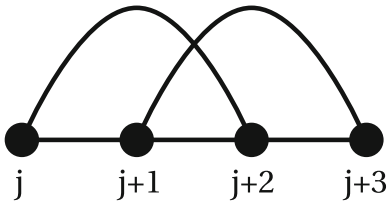


Fig. 1 A bad path with $k = 6$

Fig. 2 The pattern described in Observation 2



2. The pairs of consecutive nodes in the sequence are alternately connected by a 1-edge and a 2-edge, with the first and last edges in the sequence being both 1-edges. More formally: $(j_x, j_{x+1}) \in E_1$ for all odd $x \in \{1, \ldots, k-1\}$, $(j_x, j_{x+1}) \in E_2$ for all even $x \in \{1, \ldots, k-1\}$.

A bad path with $k = 6$ can be seen in Fig. 1.

Observation 1 *Given is an instance $\mathcal{I}$ and its graph $G(\mathcal{I}) = (V, E)$. If there exists a node $j \in V$ such that $(j, j+2) \in E_1$, i.e., if there exists a 1-triangle, then the instance is not feasible.*

Observation 2 *Given is an instance $\mathcal{I}$ and its graph $G(\mathcal{I}) = (V, E)$. If there exists a node $j \in V$ such that $(j, j+1)$, $(j+1, j+2)$ and $(j+2, j+3)$ are 1-edges and $(j, j+2)$ and $(j+1, j+3)$ are 2-edges, then the instance is not feasible.*

See Fig. 2 for the structure described in Observation 2.

Theorem 1 (Passchyn et al. (2016)) *An instance $\mathcal{I}$ is feasible if and only if its corresponding graph $G(\mathcal{I})$ does not contain a bad path, nor any of the patterns described in Observations 1 and 2.*

2.3 A lemma to prove the non-existence of online algorithms with lookahead

We use the following lemma to prove the non-existence of online algorithms with lookahead for our problem. The phrase "the first job" used below refers to the job with minimum arrival time, i.e, the job arriving at t_1.

Lemma 1 *Consider an interval scheduling problem with two machines where the goal is to decide feasibility. For any $\tau \geq 0$, if there exists two feasible instances $\mathcal{I}_1$ and $\mathcal{I}_2$ such that:*

- *the arrival times, as well as the corresponding lengths of jobs arriving in the interval $[0, \tau]$ are the same in both instances,*

- *in every feasible schedule for $\mathcal{I}_1$, the first job is assigned to M_1, whereas in every feasible schedule for $\mathcal{I}_2$, the first job is assigned to M_2,*

then there does not exist a deterministic online algorithm with lookahead time τ for this problem.

Proof We use contradiction. Suppose that there exists a deterministic online algorithm $\mathcal{A}$ with lookahead time τ for our problem. As the arrival times in the two instances restricted to the interval $[0, \tau]$ are the same, algorithm $\mathcal{A}$ cannot distinguish the two instances because its only knowledge of the input is the sequence of arrival times in the interval $[0, \tau]$. Since $\mathcal{A}$ is deterministic, it has to assign the first job of both instances to the same machine. But if $\mathcal{A}$ assigns the first job to M_1, then it does not output a feasible schedule for $\mathcal{I}_2$ (whereas there is one), and if $\mathcal{A}$ does not assign the first job to M_1, then it does not output a no-wait schedule for $\mathcal{I}_1$ (whereas there is one). Hence, we have arrived at a contradiction, and conclude that there does not exist an online algorithm with lookahead τ for our problem. □

3 Online algorithms with lookahead for unit-length jobs

In this section we deal with jobs of unit length, i.e., we assume $p_j = 1$ for all $j \in J$. We distinguish two situations: one where the speed of machine M_1 is at least as fast as twice the speed of machine M_2 (distinct machines), and one where this is not the case (similar machines). Thus, since $s_1 < 2s_2$ implies $T_2 < 2T_1$, we consider the case of distinct machines (where $T_2 \geq 2T_1$) in Sect. 3.1, and we consider the case of similar machines (where $T_2 < 2T_1$) in Sect. 3.2. The results in these subsections jointly imply the following statement:

Theorem 2 (i) *If $T_2 \geq 2T_1$, then there exists an online algorithm with lookahead time τ if and only if $\tau \geq 2T_1$.*
(ii) *If $T_2 < 2T_1$, then, for any fixed $\tau \geq 0$, there does not exist an online algorithm with lookahead time τ.*

3.1 Distinct machines: the case where $T_2 \geq 2T_1$

Here we show that for the case $T_2 \geq 2T_1$, there exists an online algorithm solving our problem if and only if the lookahead equals at least $2T_1$.

First, we prove that a lookahead time of $2T_1$ is sufficient for an online algorithm to exist. Let us first outline the algorithm, called Algorithm 1. We assume that we have access at any time t to a dynamic queue called *times* which contains all the arrival times in the lookahead interval $[t, t + \tau]$ sorted in nondecreasing order. Given an arrival time $t = \text{times}[0]$, we call, for convenience, the first job to arrive at this time, job 0; we will call the next job arriving after job 0, job 1, and so on, until no more jobs exist in $[t, t + \tau]$. At any time, the algorithm knows whether the machines are available, thanks to the boolean variables av_1 and av_2.

times: queue containing the arrival times in the lookahead interval in increasing order
clock: time running ▷ *times and clock are updated apart from this algorithm*
av_1: boolean variable indicating whether M_1 is currently available, initialized with **true**
av_2: boolean variable indicating whether M_2 is currently available, initialized with **true**
end_1: time at which M_1 will be available if it currently is not, initialized with clock-1
end_2: time at which M_2 will be available it it currently is not, initialized with clock-1
assignments: list of assignments, initialized with $\emptyset$

In the description of Algorithm 1, we model the difference between arrival times by the presence or the absence of an edge in E_1 or E_2 (recall from Sect. 2.1 that $(j_1, j_2) \in E_1\ (E_2) \Leftrightarrow t_{j_2} - t_{j_1} < T_1\ (T_2)$). Notice that, even when two jobs have the same arrival time, the algorithm handles them sequentially. Clearly, when a new job arrives, and only one machine is available, the algorithm has no other choice than assigning the arriving job to this machine. If both machines are available, the algorithm chooses the machine. The following three cases explain how our algorithm makes this choice; in each of these cases we assume that the lookahead interval is nonempty, i.e., we assume that the next job after job 0 arrives after at most τ time-units. (Indeed, if the lookahead interval is empty, we simply assign the job to M_1, and we know that both machines are available for the next job).

Case 1: $(0, 1) \notin E_1$. The algorithm assigns job 0 to M_1.

This ensures that job 0 is processed while guaranteeing that both machines are available for the next job.

Case 2: $(0, 1) \in E_1$ and $(1, 2) \notin E_1$. The algorithm assigns job 0 to M_2.

Observe that, in any feasible schedule, jobs 0 and 1 must be assigned to different machines. No matter which of these jobs goes where, we know that at time t_2, machine M_1 is available (since $(1, 2) \notin E_1$). Knowing this, it can only be beneficial to have machine M_2 available as soon as possible, i.e., to assign job 0 to M_2.

Case 3: $(0, 1) \in E_1$ and $(1, 2) \in E_1$ (which implies, since $T_2 \geq 2T_1$, $(0, 2) \in E_2$). The algorithm assigns job 0 to M_1.

Indeed, there is really no choice: job 0 needs to be assigned to M_1, job 1 to M_2 and job 2 to M_1; otherwise, there is no feasible assignment.

The pseudo-code of the algorithm we described above is given in Algorithm 1. The choice of a machine when both machines are available is implemented in the function CHOICE1. Notice that the length of the lookahead interval satisfies the condition $\tau \geq 2T_1$, which allows us to check whether $(0, 1) \in E_1$ and $(1, 2) \in E_1$ at line 4. In this algorithm, we assume the existence of a function POP that, given a queue, removes the head of the queue, and a function PUSH that, given an element and a list, pushes the element to the end of the list.

Algorithm 1 An online algorithm with lookahead time $\tau \geq 2T_1$

```
1: function CHOICE1(times)
2:     if #times = 1 or times[1] ≥ times[0] + T_1 then                ▷ Case 1
3:         return M_1
4:     else if #times = 2 or times[2] ≥ times[1] + T_1 then           ▷ Case 2
5:         return M_2
6:     else                                                           ▷ Case 3
7:         return M_1
8:
9: repeat
10:    if clock = end_1 then
11:        av_1 ← true
12:    if clock = end_2 then
13:        av_2 ← true
14:    if times ≠ ∅ and clock = times[0] then
15:        if not av_1 and not av_2 then
16:            return 'Infeasible'
17:        else if av_1 and not av_2 then
18:            machine ← M_1
19:        else if av_2 and not av_1 then
20:            machine ← M_2
21:        else
22:            machine ← CHOICE1(times)
23:        PUSH(machine, assignments)
24:        if machine = M_1 then
25:            av_1 ← false
26:            end_1 ← times[0] + T_1
27:        else
28:            av_2 ← false
29:            end_2 ← times[0] + T_2
30:        POP(times)
31: until the end
32: return assignments
```

We are now in a position to state and prove that Algorithm 1 is correct, i.e., that Algorithm 1 is indeed an online algorithm for our problem.

Lemma 2 *If $T_2 \geq 2T_1$, then, for any fixed $\tau \geq 2T_1$, Algorithm 1 is an online algorithm with lookahead time τ.*

Proof We prove that Algorithm 1 is an online algorithm for our problem. First of all, observe that if the algorithm outputs an assignment of all jobs, it is necessarily a feasible schedule because each job is assigned to a machine that is available with respect to previous assignments. Thus, on infeasible instances, the algorithm returns 'Infeasible', which is correct. It remains to show that the algorithm returns a feasible assignment if the instance is feasible. We do so by assuming that the algorithm outputs 'Infeasible', and next show that the corresponding instance is indeed infeasible.

Thus, suppose that Algorithm 1 tries to assign job j_1 at time t_{j_1}, and finds that both machines are unavailable. Then there must be two previous jobs, say j_2 and j_3 such that $(j_3, j_1) \in E_2$ and $(j_2, j_1) \in E_1$, and the algorithm has previously assigned job j_3 to M_2 and j_2 to M_1. We know that $j_3 < j_2$ (since otherwise, if $j_3 > j_2$, then

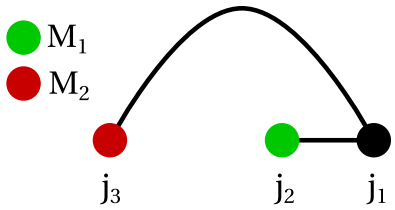


Fig. 3 The jobs j_2 and j_3 that need to exist given the impossibility to assign job j_1

Fig. 4 A bad path

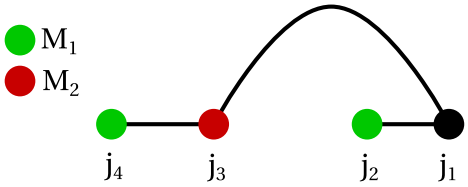


Fig. 5 The job j_4

(j_2, j_3, j_1) is a 1-triangle which makes the instance infeasible by Observation 1), as illustrated by Fig. 3.

Let us now focus on the assignment of job j_3. We know that

(1) Algorithm 1 assigned job j_3 to M_2, and
(2) job j_3 has a successor j_1 such that $(j_3, j_1) \in E_2$.

We will refer to (1) and (2) as the *properties* of job j_3, and we will use these properties to argue the existence of a previous job with the same properties.

If both M_1 and M_2 were available at time t_{j_3}, then we were in Case 2 at time t_{j_3} because j_3 has been assigned to M_2. But then there must exist another job, say j_4, such that $j_3 \prec j_4 \prec j_2$, with $(j_3, j_4) \in E_1$ and $(j_4, j_2) \notin E_1$, giving rise to the structure depicted in Fig. 4, which is a bad path with 4 nodes.

It follows that at time t_{j_3} not both machines were available, and hence, job j_3 was assigned to M_2 only because M_1 was not available. Thus, there exists a job, say job j_4, such that $(j_4, j_3) \in E_1$ and job j_4 was assigned to M_1 (see Fig. 5).

We now focus on the assignment of job j_4. Again, suppose that both machines were available at time t_{j_4}. First, observe that Algorithm 1 was not in Case 1 at time t_{j_4} because $(j_4, j_3) \in E_1$. The algorithm was not in Case 2 either (since job j_4 was assigned to M_1). Thus, Algorithm 1 was in Case 3, which means that either (i) there is another job, say job j_5, such that $j_3 \prec j_5 \prec j_2$ and $(j_3, j_5) \in E_1$, or (ii) $(j_3, j_2) \in E_1$. However, both cases are impossible since in case of (i) the instance is infeasible because of the bad path (see Fig. 4), and in case of (ii), $(j_4, j_3) \in E_1$ and $(j_3, j_2) \in E_1$ imply that $(j_4, j_2) \in E_2$ (because $2T_1 \leq T_2$) and the instance is infeasible because of the structure in Observation 2 (Fig. 2).

It follows that not both machines were available at time t_{j_4}. Thus, the only reason why job j_4 was assigned to M_1 is the existence of a job, say job j_5, with $j_5 \prec j_4$ such that job j_5 was assigned to M_2, and such that $(j_5, j_4) \in E_2$ (see Fig. 6).

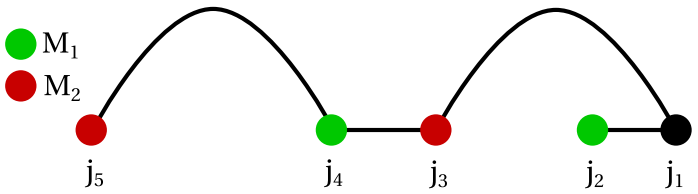
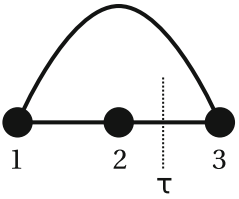


Fig. 6 The necessary existence of job j_5

Fig. 7 The graph $G(\mathcal{I}_1)$



Now, if there was a job j_6 arriving between j_5 and j_4 such that $(j_5, j_6) \in E_1$, then there would be a bad path and the instance would not be feasible. Thus, we have identified a job j_5 for which we know that

(1) Algorithm 1 assigned job j_5 to M_2, and
(2) job j_5 has a successor j_4 such that $(j_5, j_4) \in E_2$.

As announced above, observe that the properties of job j_3 allowed us to prove the existence of job j_5 that has the same properties as job j_3. Thus, applying recursively the reasoning to job j_5 that we applied to job j_3 leads to an arbitrarily long path in the graph. As the number of jobs in any instance of our problem is finite, the instance is necessarily infeasible.

As a result, Algorithm 1 outputs a feasible schedule whenever the instance is feasible, and we conclude that Algorithm 1 is correct. □

Finally, we formulate in a lemma that a lookahead time of at least $2T_1$ is necessary for an online algorithm to exist.

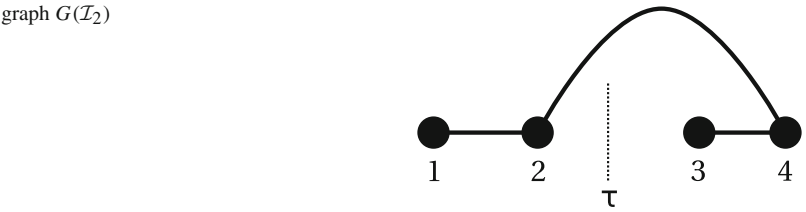
Lemma 3 *If $T_2 \geq 2T_1$, then, for any fixed $\tau < 2T_1$, there is no online algorithm with lookahead time τ.*

Proof Let $\tau < 2T_1$, and pick some ϵ such that $0 < \epsilon < \min(2T_1 - \tau, T_1)$. Consider the arrival times in the two instances $\mathcal{I}_1$ and $\mathcal{I}_2$ in the table below.

	t_1	t_2	t_3	t_4
$\mathcal{I}_1$	0	$\frac{2T_1-\epsilon}{2}$	$2T_1 - \epsilon$	
$\mathcal{I}_2$	0	$\frac{2T_1-\epsilon}{2}$	$\frac{2T_1-\epsilon}{2} + T_2 - \epsilon$	$\frac{2T_1-\epsilon}{2} + T_2 - \frac{\epsilon}{2}$

The graphs corresponding to these two instances are represented in Figs. 7 and 8.

Fig. 8 The graph $G(\mathcal{I}_2)$



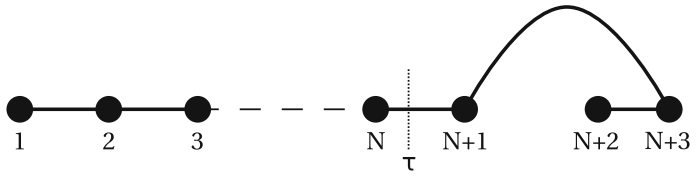


Fig. 9 The graph $G(\mathcal{I}_1)$

Notice that for both instances, the two first arrivals are identical and $\tau < t_3$. Moreover, there is only one feasible assignment for $\mathcal{I}_1$, where job 1 is assigned to M_1, whereas every feasible assignment for $\mathcal{I}_2$ assigns job 1 to M_2. Thus Lemma 1 applies to $\mathcal{I}_1$ and $\mathcal{I}_2$, which proves the result. □

3.2 Similar machines: the case where $T_2 < 2T_1$

We repeat part (ii) of Theorem 2.

Lemma 4 *If $T_2 < 2T_1$, then, for any fixed $\tau \geq 0$, there does not exist an online algorithm with lookahead time τ.*

Proof We construct two instances $\mathcal{I}_1$ and $\mathcal{I}_2$ as follows. Let $T := \dfrac{T_1 + \frac{T_2}{2}}{2}$ (this means $\frac{T_2}{2} \leq T < T_1$) and $N := \left\lceil \dfrac{\tau}{T} \right\rceil$. We have $N + 3$ jobs in $\mathcal{I}_1$, $N + 4$ jobs in $\mathcal{I}_2$, and we choose the following arrival times for these jobs.

	t_1	t_2	t_3	...	t_{N+1}	t_{N+2}	t_{N+3}	t_{N+4}
$\mathcal{I}_1$	0	T	$2T$	...	NT	$NT + T_1$	$NT + \dfrac{T_1 + T_2}{2}$	
$\mathcal{I}_2$	0	T	$2T$	...	NT	$(N+1)T$	$(N+1)T + T_1$	$(N+1)T + \dfrac{T_1 + T_2}{2}$

Notice that $t_N \leq \tau \leq t_{N+1}$. The corresponding graphs are given in Fig. 9 and 10 respectively.

Consider job $N + 1$ in $\mathcal{I}_1$. In any feasible schedule for $\mathcal{I}_1$ job $N + 1$ is assigned to M_1. Indeed, if job $N + 1$ would be assigned to M_2, then neither job $N + 2$ nor job $N + 3$ can be assigned to M_2 (since $(N+1, N+3) \in E_2$), and they also cannot both be assigned to M_1 either (since $(N+2, N+3) \in E_1$). Similarly, in any feasible schedule

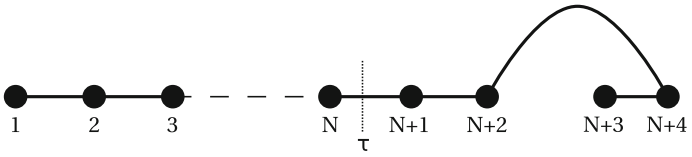


Fig. 10 The graph $G(\mathcal{I}_2)$

for $\mathcal{I}_2$ job $N+2$ is assigned to M_1. Notice that if there is a chain of consecutive 1-edges, such as the $N+1$ earliest jobs of $\mathcal{I}_1$, or the $N+2$ earliest jobs of $\mathcal{I}_2$, the assignment of one job of the chain implies those of all the other jobs, because there must be an alternation of M_1 and M_2. Thus, if N is even (odd), then $\mathcal{I}_1$ has two feasible schedules that both assign the first job to M_1 (M_2), whereas $\mathcal{I}_2$ has two feasible schedules that both assign the first job to M_2 (M_1). Notice also that the arrival times in the interval $[0, \tau]$ are the same for $\mathcal{I}_1$ and $\mathcal{I}_2$ because $NT \geq \tau$. Thus, we can apply Lemma 1, and conclude that there does not exist an online algorithm with lookahead time τ. □

Notice that the proof of Lemma 4 does not apply when $T_2 \geq 2T_1$ because in that case the existence of two consecutive 1-edges implies the existence of a 2-edge. Indeed, if $t_{j+1} - t_j < T_1$ and $t_{j+2} - t_{j+1} < T_1$, then $t_{j+2} - t_j < 2T_1 \leq T_2$. Therefore, if $T_2 \geq 2T_1$, the graphs drawn above do not correspond to any sequence of arrival times.

Clearly, Lemmata 4, 3 and 2 imply Theorem 2.

4 Online algorithms with lookahead: jobs with arbitrary length

In this section, we consider jobs of arbitrary length. Now, it becomes relevant whether we are given an upperbound, called P, on the length of the longest job in the instance. In case such a bound is given, and if the two machines are distinct, there exists an online algorithm with lookahead (see Sect. 4.1); and otherwise, there does not exist an online algorithm with any amount of lookahead (Sect. 4.2). More formally, we prove in these sections the following.

Theorem 3 (i) *If $T_2 > 2T_1$ and $p_j \leq P$ for $j \in J$, then for any fixed $\tau \geq \frac{(T_2 - T_1)^2}{T_2 - 2T_1} P$, there exists a no-wait online algorithm with lookahead time τ.*

(ii) *If $T_2 \leq 2T_1$ or if job lengths can be arbitrarily large, then for any fixed $\tau \geq 0$, there does not exist a no-wait online algorithm with lookahead time τ.*

4.1 Distinct machines and upperbounded job lengths

In this section, we first prove a lemma bounding the maximum difference in arrival times of consecutive jobs connected by 1-edges (Sect. 4.1.1), state the online algorithm (Sect. 4.1.2), and finally prove its correctness (Sect. 4.1.3).

4.1.1 Bounding arrival times of consecutive jobs

Thus, suppose that $T_2 > 2T_1$ and that all lengths are known to be smaller than P. We prove the following lemma that bounds the maximum difference in arrival times of consecutive jobs that are connected by 1-edges.

Lemma 5 *If $(j_1, j_2, \ldots, j_m)$ is a sequence of consecutive nodes such that*

(i) $(j_x, j_{x+1}) \in E_1$ *for each* $x \in \{1, \ldots, m-1\}$, *and*
(ii) $(j_x, j_{x+2}) \notin E_2$ *for each* $x \in \{1, \ldots, m-2\}$,

then $t_{j_m} - t_{j_1} < \dfrac{PT_1^2}{T_2 - 2T_1}$.

Proof Let us consider three consecutive nodes of the sequence: $j_x < j_{x+1} < j_{x+2}$ for some $x \in \{1, \ldots, m-2\}$. Clearly, since $(j_x, j_{x+1}) \in E_1$ and $(j_x, j_{x+2}) \notin E_2$ ($x \in \{1, \ldots, m-2\}$), we have

$$t_{j_{x+1}} - t_{j_x} < p_{j_x} T_1, \tag{1}$$

$$t_{j_{x+2}} - t_{j_{x+1}} < p_{j_{x+1}} T_1, \tag{2}$$

$$t_{j_{x+2}} - t_{j_x} \geq p_{j_x} T_2. \tag{3}$$

By adding (1) and (2), and using (3), we deduce:

$$p_{j_{x+1}} T_1 + p_{j_x} T_1 > t_{j_{x+2}} - t_{j_x} \geq p_{j_x} T_2, \text{ which implies:}$$

$$p_{j_x} < \frac{p_{j_{x+1}}}{\frac{T_2}{T_1} - 1}. \tag{4}$$

Using (1) and (4), we derive:

$$\begin{aligned}
t_{j_m} - t_{j_1} &= \sum_{x=1}^{m-1} (t_{j_{x+1}} - t_{j_x}) < \sum_{x=1}^{m-1} p_{j_x} T_1 \\
&< T_1 \times \sum_{x=1}^{m-1} \frac{P}{\left(\frac{T_2}{T_1} - 1\right)^{m-x}} \\
&= P \times T_1 \times \sum_{y=1}^{m-1} \left(\frac{1}{\frac{T_2}{T_1} - 1}\right)^y \\
&< P \times T_1 \times \left(\sum_{y=0}^{\infty} \left(\frac{1}{\frac{T_2}{T_1} - 1}\right)^y - 1\right) \\
&< P \times T_1 \times \left(\frac{1}{1 - \frac{1}{\frac{T_2}{T_1} - 1}} - 1\right) = P \times T_1 \times \frac{T_1}{T_2 - 2T_1}.
\end{aligned}$$

□

4.1.2 The online algorithm

Let us now explain our algorithm. To simulate an online behavior, we assume that we are given two dynamic queues t and p such that at every moment, t contains all the arrival times in the lookahead interval in increasing order and p the corresponding lengths, i.e. p[j] is the length of the job arriving at time t[j]. In the pseudo-code, t[j :] (resp. p[j :]) refers to the queue obtained by removing the j first elements of t (resp. p). We also define the same variables $clock$, av_1, av_2, end_1 and end_2 as in Algorithm 1. In the following, the current job to be assigned is always called 0 because its arrival time is t[0]. Then the next job is 1, etc. When a machine is chosen for 0, it is added at the end of the list called *assignments* thanks to a function PUSH and 0 is removed from t and p by a function POP. Both functions are assumed to be already existent.

t: queue containing the arrival times of jobs in the lookahead interval in increasing order
p: queue containing the lengths of jobs in the lookahead interval
clock: time running ▷ *t, p and clock are updated apart from this algorithm*
av_1: boolean variable indicating whether M_1 is currently available, initialized with **true**
av_2: boolean variable indicating whether M_2 is currently available, initialized with **true**
end_1: time at which M_1 will be available if it is not, initialized with clock−1
end_2: time at which M_2 will be available if it is not, initialized with clock−1
assignments: list of assignments, initialized with $\emptyset$

Moreover, when we write $(j_1, j_2) \in E_1$ (resp. $(j_1, j_2) \in E_2$) in the pseudo-code, it can be replaced by t[j_2]−t[j_1] < p[j_1]T_1 (resp. t[j_2]−t[j_1] < p[j_1]T_2). When a new job arrives, the algorithm looks at the machines available to process this job. If there is no machine available, as it is supposed to output a feasible assignment whenever such an assignment exists, the algorithm claims that the instance is infeasible. If only one machine is available, the algorithm has no other choice than assigning the job to this machine. Finally, the most decisive part of the algorithm is when both machines are available. Then, using the lookahead, we have to decide which is the best choice. We will say that a node j is *dependent* if $(j, j+1) \in E_1$, $(j+1, j+2) \in E_1$ and $(j, j+2) \notin E_2$, and *independent* otherwise. We will see that if a node 0 is independent, then a lookahead of $P \times T_2$ suffices to take a good decision on the assignment of 0. Otherwise, if 0 is dependent, we identify the first node j which is independent. This is done by the function NEXTINDEPENDENT in the pseudo-code. We take a decision on the assignment of j and we deduce the assignment of 0 by looking back and alternating M_1 and M_2 from j to 0, i.e. if j is even, we keep the assignment of j for 0, whereas if j is odd, we take the other machine. This is the reason why we need a lookahead of $\frac{T_1}{T_2 - 2T_1} PT_1 + PT_2$ (which can also be written as $\frac{(T_2 - T_1)^2}{T_2 - 2T_1} P$): $\frac{T_1}{T_2 - 2T_1} PT_1$ is an upper bound for $t_j - t_0$, by Lemma 5, and PT_2 is the remaining lookahead needed to assign correctly j. The assignment of such a node j is decided according to 5 cases. Note that a case is checked only if the conditions of the previous cases were not satisfied.

Case 1: $(0, 1) \notin E_1$. The algorithm assigns job 0 to M_1.

The Case 1 is the same as in Algorithm 1: if the next job 1 arrives after the completion of job 0 in case this job is performed by M_1, i.e. if $(0, 1) \notin E_1$, then it is always better to assign job 0 to M_1 so that both machines available for job 1.

Case 2: $\exists j \geq 1$, $(0, j+1) \in E_1$ and $(j, j+1) \in E_2$. The algorithm assigns job 0 to M_2.
This condition is checked in the function CASE2. If we are in this case and if job 0 is assigned to M_1, it is clear that the algorithm will fail to assign each job because jobs j and $j+1$ cannot use M_1 which is used by job 0 and they cannot use both M_2 because they are connected by a 2-edge. Thus, job 0 is forced to be assigned to M_2.

Case 3: $\exists j \geq 1$, $(0, j+1) \in E_2$ and $(j, j+1) \in E_1$. The algorithm assigns job 0 to M_1.
This condition is checked in the function CASE3. It is the same as the condition of Case 2 after exchanging M_1 and M_2. Hence, if this condition is satisfied, job 0 is forced to be assigned to M_1. It is because of this case that a lookahead time PT_2 is needed to assign job j.

Case 4: The job j^* defined by $j^* = \max\{j|(0, j) \in E_1\}$ satisfies $t_{j^*} + p_{j^*}T_2 \leq t_0 + p_0T_2$. The algorithm assigns job 0 to M_1
This condition is checked in the function CASE4, where j^* is computed. Note that the set $\{j|(0, j) \in E_1\}$ is not empty if we are not in Case 1. In this case, job 0 is assigned to M_1 for the following reason. If job 0 is assigned to M_2, then all jobs arriving in $[t_0, t_0 + p_0T_2]$ must be assigned to M_1. On the contrary, if job 0 is assigned to M_1, then all jobs between 1 and j^* must be assigned to M_2 (which is not a problem because we are not in the Case 2). In particular, when job j^* is assigned to M_2, then all jobs arriving in $[t_{j^*}, t_{j^*} + p_{j^*}T_2]$ must be assigned to M_1. But the assignment of jobs arriving in $[t_{j^*} + p_{j^*}T_2, t_0 + p_0T_2]$ is not restricted a priori, whereas it is forced to be M_1 if job 0 is assigned to M_2. Said differently, assigning job 0 to M_2 is more restrictive for further assignments than assigning it to M_1. Then M_1 is a better option.

Case 5: $t_{j^*} + p_{j^*}T_2 > t_0 + p_0T_2$ (negation of Case 4). The algorithm assigns job 0 to M_2.
In this case, job 0 is assigned to M_2 for the following reason. If job 0 is assigned to M_1, then all jobs between 1 and j^* must be assigned to M_2 and then all jobs arriving in $[t_{j^*}, t_{j^*} + p_{j^*}T_2]$ must be assigned to M_1. However, we have $t_{j^*} + p_{j^*}T_2 > t_0 + p_0T_2$, and thus, if job 0 is assigned to M_2, only jobs in $[t_0, t_0 + p_0T_2]$ are forced to be assigned to M_1 (which is not a problem because we are not in Case 3). Again, assigning job 0 to M_1 is more restrictive for further assignments than assigning it to M_2. Then M_2 is a better option.

Finding in which case we are is done in the function CHOICE in the pseudo-code. We are now in a position to state and prove that Algorithm 2 is correct, i.e., that Algorithm 2 is indeed an online algorithm for our problem.

4.1.3 Correctness

We rephrase part (i) of Theorem 3 as a lemma.

Algorithm 2 An online algorithm with lookahead $\tau \geq \dfrac{T_1}{T_2 - 2T_1} PT_1 + PT_2$

```
1: function NEXTINDEPENDENT(t, p)
2:     m ← #t        j ← 0
3:     while j + 2 ≤ m − 1 and (j, j + 1) ∈ E1 and (j + 1, j + 2) ∈ E1 and (j, j + 2) ∉ E2 do
4:         j ← j + 1
5:     return j
6:
7: function CASE2(t, p)
8:     m ← #t        j ← 1        found ← false
9:     while not found and j + 1 ≤ m − 1 and (0, j + 1) ∈ E1 do
10:        if (j, j + 1) ∈ E2 then
11:            found ← true
12:        j ← j + 1
13:    return found
14:
15: function CASE3(t, p)
16:    m ← #t        j ← 0        found ← false
17:    while not found and j + 1 ≤ m − 1 and (0, j + 1) ∈ E2 do
18:        if (j, j + 1) ∈ E1 then
19:            found ← true
20:        j ← j + 1
21:    return found
22:
23: function CASE4(t, p)
24:    m ← #t        j ← 0
25:    while j + 1 ≤ m − 1 and (0, j + 1) ∈ E1 do
26:        j ← j + 1
27:    if t[j]+p[j]T2 ≤ t[0]+ p[0]T2 then
28:        return true
29:    else
30:        return false
```

Lemma 6 *If $T_2 > 2T_1$ and if $p_j \leq P$ for all $j \in J$, then, for any fixed $\tau \geq \dfrac{(T_2 - T_1)^2}{T_2 - 2T_1} P$, Algorithm 2 is an online algorithm with lookahead time τ.*

Proof We prove that Algorithm 2 is indeed an online algorithm for our problem. Because each assignment of a job is compatible with the previous ones, every assignment returned by the algorithm is a feasible assignment. Then, on infeasible instances, the algorithm returns 'Infeasible' because there is no feasible schedule. Thus, the algorithm is correct on infeasible instances. Let us show that it returns a feasible schedule on feasible instances. To do so, we prove that if the algorithm returns 'Infeasible', the instance is really infeasible. The proof works by contradiction. Suppose $\mathcal{I}$ is a feasible instance on which 'Infeasible' is returned at a time t_{j_1} when the algorithm tries to assign job j_1. Then, at time t_{j_1}, both machines are unavailable because of two jobs j_2 and j_3, with $j_3 < j_2 < j_1$. There are two possible situations according to the assignments of these jobs, as shown in Fig. 11.

1. If j_2 has been assigned to M_1 and j_3 to M_2, as in Fig. 11a, j_3 is independent because $j_1 \geq j_3 + 2$ and $(j_3, j_1) \in E_2$, and then $(j_3, j_3 + 2) \in E_2$. Suppose that

```
1: function CHOICE(t, p)
2:     if #t = 1 or (0, 1) ∉ E_1 then                          ▷ Case 1
3:         return M_1
4:     else if CASE2(t, p) then                                ▷ Case 2
5:         return M_2
6:     else if CASE3(t, p) then                                ▷ Case 3
7:         return M_1
8:     else if CASE4(t, p) then                                ▷ Case 4
9:         return M_1
10:    else                                                    ▷ Case 5
11:        return M_2
12:
13: function SCHEDULE( )
14:    repeat
15:        if clock = end_1 then av_1 ← true
16:        if clock = end_2 then av_2 ← true
17:        if t ≠ ∅ and clock = t[0] then
18:            if not av_1 and not av_2 then
19:                return "Infeasible"
20:            else if av_1 and not av_2 then
21:                machine ← M_1
22:            else if av_2 and not av_1 then
23:                machine ← M_2
24:            else
25:                i ← NEXTINDEPENDENT(t, p)
26:                C ← CHOICE(t[j :], p[j :])
27:                if i mod 2 = 0 then
28:                    machine ← C
29:                else
30:                    if C = M_1 then
31:                        machine ← M_2
32:                    else
33:                        machine ← M_1
34:            PUSH(machine, assignments)
35:            if machine = M_1 then
36:                av_1 ← false
37:                end_1 ← t[0]+ p[0]T_1
38:            else
39:                av_2 ← false
40:                end_2 ← t[0]+ p[0]T_2
41:            POP(t)
42:            POP(p)
43:    until the end
44:    return assignments
```

both machines were available at time t_{j_3}. At that time, the algorithm was not in Case 1 because job j_3 has been assigned to M_2. If j_3 was in Case 2, we have seen that there is no feasible assignment where it is assigned to M_1. But there is no feasible assignment where it is assigned to M_2 either because of j_2 and j_1, then if j_3 was in Case 2, $\mathcal{I}$ would be infeasible. Therefore, j_3 was in Case 3 because $(j_3, j_2 + 1) \in E_2$ and $(j_2, j_2 + 1) \in E_1$, but then, j_3 would have been assigned to M_1. Actually, j_3 was assigned to M_2 because M_1 was not available. Therefore,

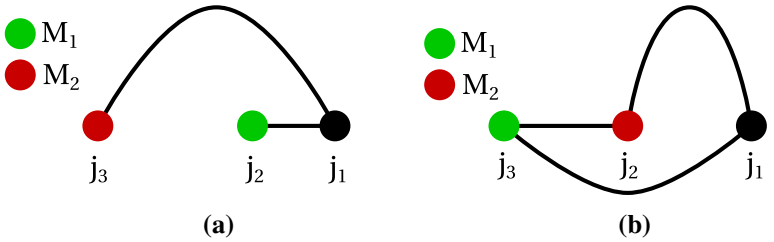
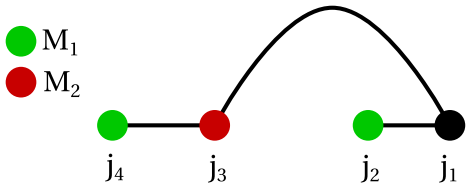


Fig. 11 The nodes j_2 and j_3 when **a** j_2 is assigned to M_1 and j_3 to M_2 **b** j_2 is assigned to M_2 and j_3 to M_1

Fig. 12 The node j_4



there exists a previous job j_4 such that $(j_4, j_3) \in E_1$ and j_4 had been assigned to M_1, as shown in Fig. 12.

Hence j_2, j_3 and j_4 satisfy the following properties:

(P1) $(j_3, j_2) \in E_2$ and $(j_4, j_3) \in E_1$,
(P2) j_2, j_3 and j_4 were assigned to M_1, M_2 and M_1 respectively,
(P3) There is no feasible assignment where j_2 is assigned to M_1,
(P4) There is no feasible assignment where j_3 is assigned to M_2,
(P5) j_3 is independent,
(P6) A call of CHOICE with j_3 as first node would not have returned M_2.

Let us prove the existence of some previous nodes j_5 and j_6. First, observe that there is no feasible assignment where j_4 is assigned to M_1 because there is no feasible assignment where j_3 is assigned to M_2 (P4) and $(j_4, j_3) \in E_1$ (P1). If j_4 was dependent, then $j_3 = j_4 + 1$ would hold but (P5) and (P6) imply that the next independent node after j_4 is j_3, and if one could have chosen the assignment of j_3 between M_1 and M_2, CHOICE would have returned M_1 and then j_4 would have been assigned to M_2. Then j_4 is independent. Now suppose that both machines were available at time t_{j_4}. Let us show that j_4 was neither in Case 1, nor in Case 3, nor in Case 4. j_4 was not in Case 1 because $(j_4, j_3) \in E_1$. If j_4 was in Case 3, then we have seen in the description of this case that, on the one hand, there would be no feasible assignment where j_4 is assigned to M_2. On the other hand, if j_4 is assigned to M_1, then j_3 is forced to be assigned to M_2. However, there is no feasible assignment where j_3 is assigned to M_2 (P4). Thus $\mathcal{I}$ can not be feasible if j_4 was in Case 3. If j_4 was in Case 4, then the condition of Case 4 implies that $(j_4, j_2) \in E_2$, because $j_3 \preceq j_4^*$ and $(j_3, j_2) \in E_2$. The pattern of Fig. 13 is present. Then, whatever the assignment of j_4, j_2 is forced to be assigned to M_2. This contradicts (P3) because $\mathcal{I}$ is supposed to be feasible.

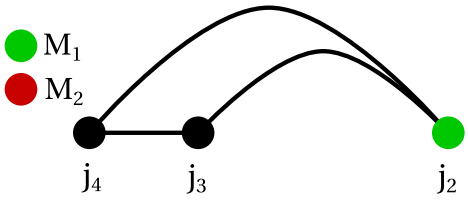


Fig. 13 j_2 is forced to be assigned to M_1

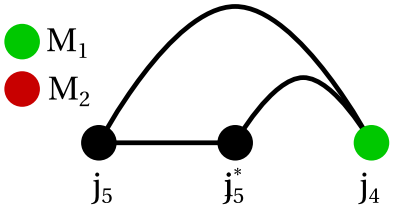


Fig. 14 The node j_5^*

Then we have proved that, if CHOICE had been called with j_4 as first node, it could not have returned M_1, then j_4 was assigned to M_1 because M_2 was not available. Therefore, there exists a previous node j_5 assigned to M_2 such that $(j_5, j_4) \in E_2$.
Now, we prove the existence of another previous node j_6. First, observe that there is no feasible assignment where j_5 is assigned to M_2 because there is no feasible assignment where j_4 is assigned to M_1 and $(j_5, j_4) \in E_2$ (P1). If j_5 was dependent, then $j_4 = j_5 + 1$ would hold, but we have just proved that j_4 is independent and that if one could have chosen its assignment between M_1 and M_2, CHOICE would have returned M_2 and then j_5 would have been assigned to M_1. Then j_5 is independent. Suppose that both machines were available at time t_{j_5}. Let us show that j_5 was neither in Case 2, nor in Case 5. If j_5 was in Case 2, then we have seen in the description of this case that, on the one hand, there would be no feasible assignment where j_5 is assigned to M_1. On the other hand, if j_5 is assigned to M_2, then j_4 is forced to be assigned to M_1. However, there is no feasible assignment where j_4 is assigned to M_1, thus $\mathcal{I}$ can not be feasible if j_5 was in Case 2. If j_5 was in Case 5, let us assume towards contradiction that $j_5^* \geq j_4$. Then $(j_5, j_4) \in E_1$, by definition of j_5^*, and the condition of Case 5 entails that $(j_5, j_3) \in E_2$ because $(j_4, j_3) \in E_1$. But, if $(j_5, j_3) \in E_2$ and $(j_4, j_3) \in E_1$, we were not in Case 5 but in Case 3. Then $j_5^* < j_4$, and the condition of Case 5 implies $(j_5^*, j_4) \in E_2$ (see Fig. 14) because $(j_5, j_4) \in E_2$. Hence, whatever the assignment of j_5, j_4 is forced to be assigned to M_1, which contradicts the feasibility of $\mathcal{I}$.
Therefore, if CHOICE had been called with j_5 as first node, it could not have returned M_2, then j_5 was assigned to M_2 because M_1 was not available. Then there exists a previous node j_6 such that $(j_6, j_5) \in E_1$ and j_6 had been assigned to M_1. Then j_4, j_5 and j_6 satisfy the same properties as j_2, j_3 and j_4, namely:

(P1) $(j_5, j_4) \in E_2$ and $(j_6, j_5) \in E_1$,
(P2) j_4, j_5 and j_6 were assigned to M_1, M_2 and M_1 respectively,
(P3) There is no feasible assignment where j_4 is assigned to M_1,
(P4) There is no feasible assignment where j_5 is assigned to M_2,
(P5) j_5 is independent,

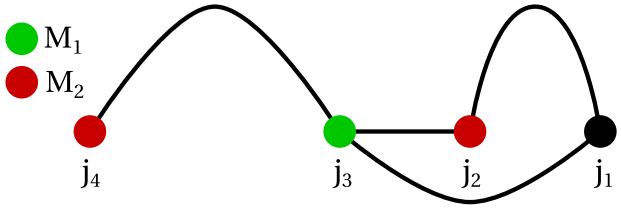


Fig. 15 The node j_4

(P6) A call of CHOICE with j_5 as first node would not have returned M_2.

Observe that the properties of jobs j_2, j_3 and j_4 are sufficient to prove the existence of jobs j_5 and j_6 such that jobs j_4, j_5 and j_6 have the same properties. Thus, applying recursively the reasoning to jobs j_4, j_5 and j_6 that was applied to j_2, j_3 and j_4 leads to an arbitrary long path in the graph. As the number of jobs is finite, the instance $\mathcal{I}$ is necessarily infeasible.

2. We deal here with the second configuration where j_2 was assigned to M_2 and j_3 to M_1, as in Fig. 11b. Then there is also a 1-edge (j_2, j_3) implied by the 1-edge (j_2, j_1). j_3 is independent because it is involved in two 1-edges. Suppose that both machines were available at time t_{j_3}. j_3 was not in Case 1 because $(j_3, j_2) \in E_1$. Then j_3 was in Case 2 because $(j_3, j_2 + 1) \in E_1$ and $(j_2, j_2 + 1) \in E_2$. Thus, if CHOICE had been called with j_3 as first job, j_3 would have been assigned to M_2. Hence, j_3 was assigned to M_1 because M_2 was not available. Therefore, there exists a previous job j_4 assigned to M_2 such that $(j_4, j_3) \in E_2$ (see Fig. 15).

Hence j_2, j_3 and j_4 satisfy the following properties:

(P1) $(j_3, j_2) \in E_1$ and $(j_4, j_3) \in E_2$,
(P2) j_2, j_3 and j_4 were assigned to M_2, M_1 and M_2 respectively,
(P3) There is no feasible assignment where j_2 is assigned to M_2,
(P4) There is no feasible assignment where j_3 is assigned to M_1,
(P5) j_3 is independent,
(P6) A call of CHOICE with j_3 as first node would not have returned M_1.

Let us prove the existence of some previous nodes j_5 and j_6. First, observe that there is no feasible assignment where j_4 is assigned to M_2 because there is no feasible assignment where j_3 is assigned to M_1 (P4) and $(j_4, j_3) \in E_2$ (P1). If j_4 was dependent, then $j_3 = j_4 + 1$ would hold but (P5) and (P6) imply that the next independent node after j_4 is j_3, and if j_3 could have chosen between M_1 and M_2, it would have chosen M_2 and then j_4 would have been assigned to M_1. Then j_4 is independent. Suppose that both machines were available at time t_{j_4}. Let us show that j_4 was neither in Case 2, nor in Case 5. If j_4 was in Case 2, then we have seen in the description of this case that, on the one hand, there would be no feasible assignment where j_4 is assigned to M_1. On the other hand, if j_4 is assigned to M_2, then j_3 is forced to be assigned to M_1. However, there is no feasible assignment where j_3 is assigned to M_1, thus $\mathcal{I}$ can not be feasible if j_4 was in case 2. If j_4 was in Case 5, let us assume towards

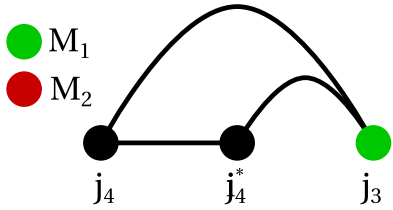


Fig. 16 The node j_4^*

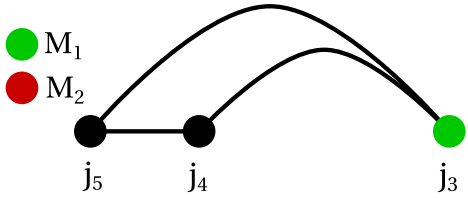


Fig. 17 j_3 is forced to be assigned to M_1

contradiction that $j_4^* \geq j_3$. Then $(j_4, j_3) \in E_1$, by definition of j_4^*, and the condition of Case 5 entails that $(j_4, j_2) \in E_2$ because $(j_3, j_2) \in E_1$. But, if $(j_4, j_2) \in E_2$ and $(j_3, j_2) \in E_1$, we were not in Case 5 but in Case 3. Then $j_4^* < j_3$, and the condition of the Case 5 implies that $(j_4^*, j_3) \in E_2$ (see Fig. 14). Hence, whatever the assignment of j_4, j_3 is forced to be assigned to M_1, and j_3 is forced to be assigned to M_1. This contradicts (P3) because $\mathcal{I}$ is supposed to be feasible (Fig. 16).

Therefore, if CHOICE had been called with j_4 as first node, it could not have returned M_2, then j_4 was assigned to M_2 because M_1 was not available. Then there exists a previous node j_5 such that $(j_5, j_4) \in E_1$ and j_5 had been assigned to M_1.

Now, we prove the existence of another previous node j_6. First, observe that there is no feasible assignment where j_5 is assigned to M_2 because there is no feasible assignment where j_4 is assigned to M_1 and $(j_5, j_4) \in E_2$ (P1). If j_5 was dependent, then $j_4 = j_5 + 1$ would hold but we have just proved that j_4 is independent and that if one could have chosen between M_1 and M_2, CHOICE would have returned M_1 and then j_5 would have been assigned to M_2. Then j_5 is independent. Now suppose that both machines were available at time t_{j_5}. Let us show that j_5 was neither in Case 1, nor in Case 3, nor in Case 4. j_5 was not in Case 1 because $(j_5, j_4) \in E_1$. If j_5 was in Case 3, then we have seen in the description of this case that, on the one hand, there would be no feasible assignment where j_5 is assigned to M_2. On the other hand, if j_5 is assigned to M_1, then j_4 is forced to be assigned to M_2. However, there is no feasible assignment where j_4 is assigned to M_2, thus $\mathcal{I}$ can not be feasible if j_5 was in Case 3. If j_5 was in Case 4, the condition of Case 4 implies that $(j_5, j_3) \in E_2$. Then whatever the assignment of j_5, j_3 is forced to be assigned to M_1 because of the pattern in Fig. 17, which contradicts the feasibility of $\mathcal{I}$.

Then we have proved that, if CHOICE had been called with j_5 as first node, it could not have returned M_1, then j_5 was assigned to M_1 because M_2 was not available. Therefore, there exists a previous node j_6 assigned to M_2 such that $(j_6, j_5) \in E_2$. Then j_4, j_5 and j_6 satisfies the same properties as j_3 and j_4, namely:

(P1) $(j_5, j_4) \in E_1$ and $(j_6, j_5) \in E_2$,
(P2) j_4, j_5 and j_6 were assigned to M_2, M_1 and M_2 respectively,

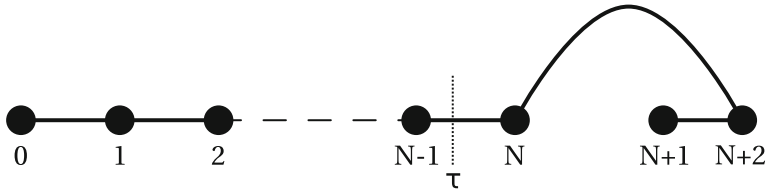


Fig. 18 The graph $G(\mathcal{I}_1)$

(P3) There is no feasible assignment where j_4 is assigned to M_2,
(P4) There is no feasible assignment where j_5 is assigned to M_1,
(P5) j_5 is independent,
(P6) A call of CHOICE with j_5 as first node would not have returned M_1.

Observe that the properties of job j_2, j_3 and j_4 were sufficient to prove the existence of jobs j_5 and j_6 such that j_4, j_5 and j_6 have the same properties. Thus, applying recursively the reasoning to jobs j_4, j_5 and j_6 that was applied to j_2, j_3 and j_4 leads to an arbitrary long path in the graph. As the number of jobs is finite, the instance $\mathcal{I}$ is necessarily infeasible.

Thus we have proved that if the algorithm returns "Infeasible", then the instance is infeasible. Then, on feasible instances, a feasible assignment is returned. □

Notice that the case where $P = 1$ is not identical to the special case of unit-length jobs.

4.2 Similar machines

Let us now address the case of similar machines and/or jobs whose length is not upperbounded. We first repeat part (ii) of Theorem 3, and formulate it as a lemma.

Lemma 7 *If $T_2 \leq 2T_1$ or if job lengths can be arbitrarily large, then for any fixed $\tau \geq 0$, there does not exist a no-wait online algorithm with lookahead time τ.*

Proof If $T_2 < 2T_1$, this is a consequence of the first part of Theorem 2 because the unit length setting is a particular case of the arbitrary length setting. The rest of the proof splits into two parts: the case of arbitrarily large job lengths and the special case with $T_2 = 2T_1$ and bounded job lengths.

First, let us suppose that the lengths can be arbitrarily large, i.e., we do not know a priori an upper bound on them. We build two instances $\mathcal{I}_1$ and $\mathcal{I}_2$ such that $G(\mathcal{I}_1)$ is the graph shown in Fig. 18 and $G(\mathcal{I}_2)$ is the graph shown in Fig. 19.

Let N be an integer, we will see how to fix it conveniently. $\mathcal{I}_1$ has $N + 3$ jobs numbered from 0 to $N + 2$, and is defined as follows:

$$t_0 = 0, \qquad p_0 = 1,$$
$$\forall j \in \{1, \ldots, N\}: \quad t_j = t_{j-1} + \left(\frac{T_2}{T_1}\right)^{j-1} T_1 - \frac{T_1}{2}, \qquad p_j = \left(\frac{T_2}{T_1}\right)^{j},$$

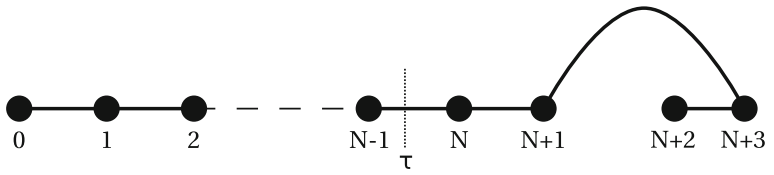


Fig. 19 The graph $G(\mathcal{I}_2)$

$$t_{N+1} = t_N + \left(\frac{T_2}{T_1}\right)^N T_2 - T_1, \qquad p_{N+1} = 1,$$
$$t_{N+2} = t_N + \left(\frac{T_2}{T_1}\right)^N T_2 - \frac{T_1}{2}, \qquad p_{N+2} = 1.$$

$\mathcal{I}_2$ has $N + 4$jobs numbered from 0 to $N + 3$, and is defined as follows:

$$\begin{aligned}
& t_0 = 0, && p_0 = 1,\\
\forall j \in \{1, \ldots, N+1\}: \quad & t_j = t_{j-1} + \left(\frac{T_2}{T_1}\right)^{j-1} T_1 - \frac{T_1}{2}, && p_j = \left(\frac{T_2}{T_1}\right)^j,\\
& t_{N+2} = t_{N+1} + \left(\frac{T_2}{T_1}\right)^{N+1} T_2 - T_1, && p_{N+2} = 1,\\
& t_{N+3} = t_{N+1} + \left(\frac{T_2}{T_1}\right)^{N+1} T_2 - \frac{T_1}{2}, && p_{N+3} = 1.
\end{aligned}$$

We claim that the resulting instances $\mathcal{I}_1$ and $\mathcal{I}_2$ are depicted by Figs. 18 and 19. To argue this claim, we need to show that $(j, j+1), (j+1, j+2) \in E_1$, while $(j, j+2) \notin E_2$ for each $j \in \{0, \ldots, N-2\}$. Using the arrival times defined above, we see that:

$$t_{j+1} - t_j = \left(\frac{T_2}{T_1}\right)^j T_1 - \frac{T_1}{2} < p_j T_1, \tag{5}$$
$$t_{j+2} - t_{j+1} = \left(\frac{T_2}{T_1}\right)^{j+1} T_1 - \frac{T_1}{2} < p_{j+1} T_1. \tag{6}$$

This shows $(j, j+1), (j+1, j+2) \in E_1$ for each $j \in \{0, \ldots, N-2\}$. Summing (5) and (6) gives

$$t_{j+2} - t_j = \left(\frac{T_2}{T_1}\right)^{j+1} T_1 + \left(\frac{T_2}{T_1}\right)^j T_1 - 2\frac{T_1}{2} = \left(\frac{T_2}{T_1}\right)^j (T_2 + T_1) - T_1 \geq p_j T_2,$$

implying $(j, j+2) \notin E_2$ for each $j \in \{0, \ldots, N-2\}$.

Consider now the value of t_N: $t_N = \sum_{j=0}^{N-1} \left(\left(\frac{T_2}{T_1} \right)^j T_1 - \frac{T_1}{2} \right) = \frac{\left(\frac{T_2}{T_1} \right)^N - 1}{\frac{T_2}{T_1} - 1} T_1 - N\frac{T_1}{2}$. As this quantity becomes arbitrarily large as N grows, we define N as the smallest integer such that $t_N \geq \tau$. Thus, for each value of $\tau > 0$, it is possible to build the instances of Figs. 18 and 19. Since these instances satisfy the assumptions of Lemma 1, it follows that there does not exist an online algorithm with lookahead time τ if lengths can be arbitrarily large.

Second, we deal with the case where $T_2 = 2T_1$ and $p_j \leq P$ for $j \in J$. It is clear that as soon as we are able to build two infinite sequences of jobs such that two consecutive jobs are linked by a 1-edge but no two jobs are linked by a 2-edge, then for any $\tau > 0$, it is possible to use this sequence to build two instances like those of Figs. 18 and 19 proving that there is no online algorithm with lookahead τ. Thus we have to prove that for every $P > 0$ and every $T_1 > 0$, there exists a sequence $(t_n)_{n\in\mathbb{N}}$ of arrival times and a sequence $(p_n)_{n\in\mathbb{N}}$ of job lengths satisfying the following properties:

(a) $\forall n \geq 0,\ t_{n+1} - t_n < p_n T_1$,
(b) $\forall n \geq 0,\ t_{n+2} - t_n \geq 2p_n T_1$ (since $T_2 = 2T_1$),
(c) $\forall n \geq 0,\ p_n \leq P$.

Let us construct such a sequence in the following way. Let α be a positive real number sufficiently small such that $\alpha < \min(\frac{3}{2}, \frac{1}{4}P)$ and β a positive real number sufficiently large such that $4\alpha + \frac{2}{\beta-1} \leq P$. Take as initial values $t_0 = 0$, $p_0 = 3\alpha$ and $t_1 = 2\alpha T_1$. Then, we define t_{n+2} and p_{n+1} by:

- $t_{n+2} = t_n + 2p_n T_1$ for $n = 0, 1, \ldots,$
- $p_n = \frac{t_{n+1}-t_n}{T_1} + \beta^{-n}$ for $n = 1, 2, \ldots$.

Now we check that these sequences satisfy (a), (b) and (c). We can see that $t_1 - t_0 = 2\alpha T_1 < p_0 T_1$, and moreover, $\forall n \geq 1, t_{n+1} - t_n = p_n T_1 - \beta^{-n} T_1 < p_n T_1$; it follows that (a) holds. Furthermore, by construction, (b) holds and is even always tight. Finally, we prove that all job lengths are bounded by P. By the choice of α, $p_0 = 3\alpha < P$. Then, because $t_2 = 6\alpha T_1$, $p_1 = 4\alpha + \frac{1}{\beta}$ which is less than $4\alpha + \frac{2}{\beta-1}$, which is less than P by the choice of β. As regards the following job lengths, for any $n \geq 1$, on the one hand, $t_{n+2} = t_n + 2p_n T_1$, and on the other hand, $t_{n+2} - t_n = (t_{n+2} - t_{n+1}) + (t_{n+1} - t_n) = (p_{n+1} T_1 - \beta^{-(n+1)} T_1) + (p_n T_1 - \beta^{-n} T_1)$. Thus $p_{n+1} = p_n + (\beta + 1)\beta^{-(n+1)}$. This implies for any $n \geq 1$:

$$
\begin{aligned}
p_n &= p_1 + \sum_{i=2}^{n} (\beta + 1)\beta^{-i} \\
&= 4\alpha + \frac{1}{\beta} + \frac{\beta + 1}{\beta^2} \sum_{i=0}^{n-2} \beta^{-i} \\
&\leq 4\alpha + \frac{1}{\beta} + \frac{\beta + 1}{\beta^2} \frac{1}{1 - \beta^{-1}}
\end{aligned}
$$

$$= 4\alpha + \frac{\beta - 1}{\beta(\beta - 1)} + \frac{\beta + 1}{\beta(\beta - 1)}$$
$$= 4\alpha + \frac{2}{\beta - 1} \leq P \text{ (by choice of } \beta)$$

Then (c) holds, which concludes the proof of part (i). □

Clearly, Lemmata 6 and 7 imply Theorem 3.

5 Extension: minimizing total waiting time

In this section, we deviate from the problem stated in Sect. 1, and allow that a job starts later than its arrival time. Hence, a job may have to wait before the machine starts processing it, and we are interested in obtaining solutions with minimum total waiting time. From this point of view, Theorem 2 shows that Algorithm 1 is capable of finding a solution with total waiting time 0 if one exists (provided that $T_2 \geq 2T_1$, and provided that there is a certain amount of lookahead), and it is conceivable that extensions of Algorithm 1 exist that would find solutions with minimum total waiting time. Unfortunately, the following theorem shows that it is not the case. Let us recall that an online algorithm whose goal is to minimize a quantity is said to be *k-competitive* if it always outputs a solution whose value is at most k times the optimal value.

Theorem 4 *For any fixed $\tau \geq 0$, there is no online algorithm with lookahead time τ which always outputs a schedule minimizing the total waiting time. Furthermore, for any $k \geq 1$, there is no online algorithm with lookahead time τ which is k-competitive in terms of total waiting time.*

Proof First of all, observe that if $T_2 < 2T_1$, this result is a corollary of Theorem 2. Indeed, the first part of the theorem is true since there is even no online algorithm that always outputs a no-wait schedule whenever such a schedule exists, then there is no online algorithm that minimizes the waiting time. The second part of the theorem comes from the fact that a k-approximation of a no-wait schedule is also a no-wait schedule. Therefore, let us now restrict ourselves to the case where $T_2 \geq 2T_1$. Let $\tau \geq 0$ be an arbitrary lookahead time and let $N = \left\lceil \frac{\tau}{T_2} \right\rceil$ so that $NT_2 \geq \tau$. Let also $K > 1$ be an arbitrary number (we will see later for what it is useful). We will use an instance $\mathcal{I}_N$ having $2N + 4$ jobs. Its graph is represented in Fig. 20 and its arrival times are the following:

$$t_1 = 0 \quad t_2 = \frac{T_1}{2} \quad t_3 = T_2 - \frac{T_1}{4NK} \quad t_4 = T_2 + \frac{3T_1}{4} - \frac{T_1}{4NK} \quad \ldots$$
$$t_{2i+1} = iT_2 - \frac{T_1}{4NK} \quad t_{2i+2} = iT_2 + \frac{3T_1}{4} - \frac{T_1}{4NK} \quad (\text{for } 2 \leq i \leq N) \quad \ldots$$
$$t_{2N+3} = (N+1)T_2 + \frac{T_1}{4} - \frac{T_1}{4NK} \quad t_{2N+4} = (N+1)T_2 + \frac{T_1}{2} - \frac{T_1}{4NK}$$

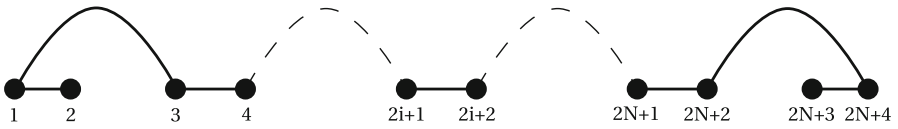
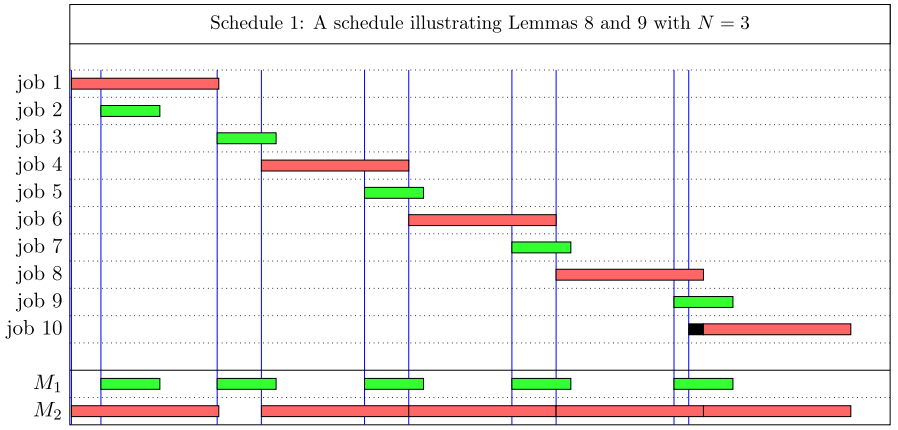


Fig. 20 The graph $G(\mathcal{I}_N)$

Schedule 1 can be helpful to visualize the instance $\mathcal{I}_N$ (here with $N = 3$) and the arguments of the proofs of the following lemmas. The arrival times are represented by vertical blue lines.



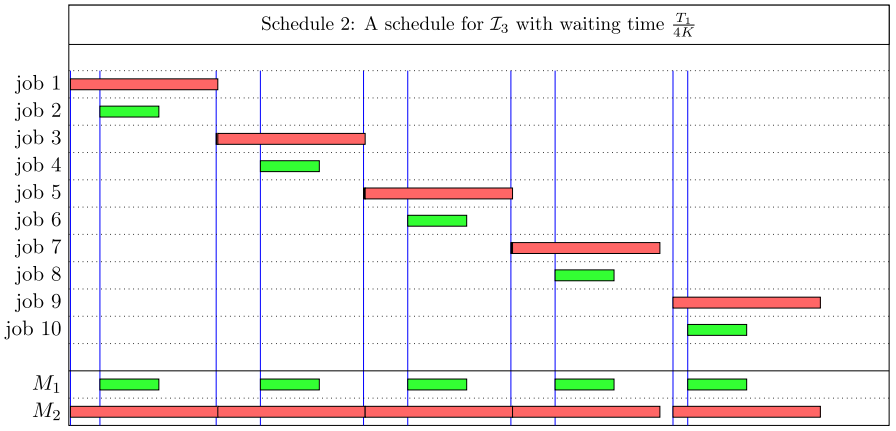
Lemma 8 *Let $\mathcal{A}$ be a no-wait online algorithm with lookahead time $\tau \leq NT_2$. Taking $\mathcal{I}_N$ as input, $\mathcal{A}$ schedules job 3 at time t_3 on M_1.*

Proof Since $\mathcal{I}_N$ contains a bad path, there does not exist a no-wait schedule for this instance. However, if we remove job $2N + 4$, the instance becomes feasible because it no longer contains a bad path. Therefore, as long as the algorithm $\mathcal{A}$ (which is a correct algorithm for feasible instances), does not see job $2N + 4$ in its lookahead interval, it "does not know" that the instance is infeasible and is forced to do as if it was feasible. Otherwise, it would fail on some feasible instances. Notice that $t_{2N+4} > t_3 + NT_2$, because $t_5 - t_3 = T_2, t_7 - t_5 = T_2, \ldots, t_{2N+1} - t_{2N-1} = T_2$ and $t_{2N+4} - t_{2N+1} = T_2 + \frac{T_1}{2} > T_2$. Then $\mathcal{A}$ schedules the three earliest jobs as soon as they arrive to avoid any waiting time. Obviously, jobs 1 and 2 are assigned to different machines because they are connected by a 1-edge. With respect to job 3, whatever the assignments of jobs 1 and 2, M_1 is available at time t_3 whereas M_2 is not. Thus, job 3 is assigned to M_1 at time t_3 to incur no waiting time. □

Lemma 9 *Any schedule for the instance $\mathcal{I}_N$ where job 3 is scheduled at time t_3 on M_1 has a total waiting time greater than or equal to $\frac{T_1}{4}$.*

Proof Consider a schedule where job 3 is scheduled at time t_3 on M_1. If an algorithm waits in order to assign job 4 to M_1 too, it incurs a waiting time of at least $(t_3+T_1)-t_4 = \frac{T_1}{4}$, which satisfies the lemma. Thus, suppose that job 4 is assigned to M_2. Then, if an

algorithm waits for M_2 to be released to assign job 5 to it, it incurs a waiting time of at least $(t_4 + T_2) - t_5 = \frac{3T_1}{4} > \frac{T_1}{4}$, which satisfies the lemma. Thus, suppose that job 5 is assigned to M_1. By repeating this reasoning for the following jobs, we prove that if not all even jobs between 4 and $2N + 2$ are assigned to M_2 or if not all odd jobs between 5 and $2N + 3$ are assigned to M_1, then the statement of the lemma is true because for every $i \in \{2, \ldots, N\}$, $(t_{2i+1} + T_1) - t_{2i+2} = \frac{T_1}{4}$, for every $i \in \{2, \ldots, N-1\}$, $(t_{2i+2} + T_2) - t_{2i+3} = \frac{3T_1}{4} > \frac{T_1}{4}$ and $(t_{2N+2} + T_1) - t_{2N+3} = \frac{T_1}{2} > \frac{T_1}{4}$. But if the converse is true, then $2N + 2$ is assigned to M_2 and $2N + 3$ is assigned to M_1. Therefore, $2N + 4$ has to wait at least $(t_{2N+2} + T_2) - t_{2N+4} = \frac{T_1}{4}$ for M_2, or at least $(t_{2N+3} + T_1) - t_{2N+4} = \frac{3T_1}{4} > \frac{T_1}{4}$ for M_1. In all cases, the total waiting time is greater than or equal to $\frac{T_1}{4}$.



However, a waiting time of $\frac{T_1}{4}$ or more is not optimal: there is a better schedule which consists in assigning odd jobs to M_2 and even jobs to M_1. Its waiting time is $\frac{T_1}{4K}$ because the N odd jobs $3, 5, \ldots, 2N + 1$ all have to wait $\frac{T_1}{4NK}$ before using M_2. Schedule 2 represents such a schedule with $N = 3$. We can now check the statements of the theorem. On the one hand, if $\mathcal{A}$ always minimizes the waiting time, then in particular, it is optimal on instances having a no-wait schedule. Then, Lemmas 8 and 9 imply that $\mathcal{A}$ does not return an optimal schedule on $\mathcal{I}_N$, which is a contradiction. On the other hand, if $\mathcal{A}$ is only k-competitive, for some $k > 1$, it must also be optimal on instances having a no-wait schedule, because a k-approximation of a no-wait schedule is also a no-wait schedule. Then, Lemma 8 applies also to $\mathcal{A}$, and Lemma 9 implies that $\mathcal{A}$ returns a schedule whose waiting time is K times worse than the optimal waiting time. By choosing $K > k$, it contradicts the hypothesis according to which $\mathcal{A}$ is k-competitive. □

6 Conclusion

We have focussed on the potential that lookahead offers for online algorithms to solve an interval scheduling problem with two related machines. We showed that, in case of

unit-length jobs, if the ratio between the speeds of the machines is at least 2, then there exists an online algorithm with lookahead that finds a feasible schedule whenever one exists. If this ratio is less than 2, no online algorithm exists for the resulting instances. This result can be extended to jobs with arbitrary lengths provided an upperbound on the maximum job length is given. We also showed that this result cannot be extended when the goal is to minimize total waiting time. Summarizing: there are situations where an amount of lookahead allows to obtain results that cannot be obtained without lookahead.

References

Carlisle MC, Lloyd EL (1995) On the k-coloring of intervals. Discrete Appl Math 59(3):225–235
Disser Y, Klimm M, Lübbecke E (2015) Scheduling bidirectional traffic on a path. arXiv preprint arXiv:1504.07129
Dosa G, Wang Y, Han X, Guo H (1994) Online scheduling with rearrangement on two related machines. Theor Comput Sci 130(1):5–16
Epstein L, Jeż Ł, Sgall J, van Stee R (2016) Online scheduling of jobs with fixed start times on related machines. Algorithmica 74(1):156–176
Epstein L, Levin A (2010) Improved randomized results for the interval selection problem. Theor Comput Sci 411:3129–3135
Erlebach T, Spieksma F (2003) Interval selection: applications, algorithms, and lower bounds. J Algorithms 46(1):27–53
Faigle U, Nawijn WM (1995) Note on scheduling intervals on-line. Discrete Appl Math 58(1):13–17
Fung SP, Poon CK, Yung DK (2012) On-line scheduling of equal-length intervals on parallel machines. Inf Process Lett 112:376–379
Fung SP, Poon CK, Zheng F (2008) Online interval scheduling: randomized and multiprocessor cases. J Comb Optim 16(3):248–262
Fung SP, Poon CK, Zheng F (2014) Improved randomized online scheduling of intervals and jobs. Theory Comput Syst 55(1):202–228
Hermans J (2014) Optimization of inland shipping. J Sched 17(4):305
Kolen AW, Lenstra JK, Papadimitriou CH, Spieksma FC (2007) Interval scheduling: a survey. Nav Res Logist 54(5):530–543
Krumke SO, Thielen C, Westphal S (2011) Interval scheduling on related machines. Comput Oper Res 38(12):1836–1844
Li W, Yuan J, Cao J, Bu H (2009) Online scheduling of unit length jobs on a batching machine to maximize the number of early jobs with lookahead. Theor Comput Sci 410:5182–5187
Lipton RJ, Tomkins A (1994) Online interval scheduling. SODA 94:302–311
Miyazawa H, Erlebach T (2004) An improved randomized on-line algorithm for a weighted interval selection problem. J Sched 7(4):293–311
Passchyn W, Briskorn D, Spieksma F (2016) No-wait scheduling for locks. INFORMS J Comput (to appear)
Passchyn W, Briskorn D, Spieksma FC (2016) Mathematical programming models for lock scheduling with an emission objective. Eur J Oper Res 248(3):802–814
Passchyn W, Coene S, Briskorn D, Hurink J L, Spieksma F C, Vanden Berghe G (2016) The lockmaster's problem. Eur J Oper Res 251(2):432–441
Prandtstetter M, Ritzinger U, Schmidt P, Ruthmair M (2015) A variable neighborhood search approach for the interdependent lock scheduling problem. In: European Conference on Evolutionary Computation in Combinatorial Optimization, pp. 36–47. Springer

Schwarz U (2008) Online scheduling on semi-related machines. Inf Process Lett 108:38–40
Sgall J (1998) On-line scheduling. In: Fiat A, Woeginger G (eds) Online algorithms: the state of the art. Springer, Berlin, pp 196–231
Smith LD, Nauss RM, Mattfeld DC, Li J, Ehmke JF, Reindl M (2011) Scheduling operations at system choke points with sequence-dependent delays and processing times. Transp Res Part E Logist Transp Rev 47(5):669–680
Woeginger GJ (1994) On-line scheduling of jobs with fixed start and end times. Theor Comput Sci 130(1):5–16
Yu G, Jacobson SH (2018) Online c-benevolent job scheduling on multiple machines. Optim Lett 12(2):251–263
Zheng F, Cheng Y, Liu M, Xu Y (2013) Online interval scheduling on a single machine with finite lookahead. Comput Oper Res 40(1):180–191