Online interval scheduling to maximize total satisfaction

Koji M. Kobayashi

Abstract

The interval scheduling problem is one variant of the scheduling problem. In this paper, we propose a novel variant of the interval scheduling problem, whose definition is as follows: given jobs are specified by their release times, deadlines and profits. An algorithm must start a job at its release time on one of \( m \) identical machines, and continue processing until its deadline on the machine to complete the job. All the jobs must be completed and the algorithm can obtain the profit of a completed job as a user’s satisfaction. It is possible to process more than one job at a time on one machine. The profit of a job is distributed uniformly between its release time and deadline, that is its interval, and the profit gained from a subinterval of a job decreases in reverse proportion to the number of jobs whose intervals intersect with the subinterval on the same machine. The objective of our variant is to maximize the total profit of completed jobs.

This formulation is naturally motivated by best-effort requests and responses to them, which appear in many situations. In best-effort requests and responses, the total amount of available resources for users is always invariant and the resources are equally shared with every user. We study online algorithms for this problem. Specifically, we show that for the case where the profits of jobs are arbitrary, there does not exist an algorithm whose competitive ratio is bounded. Then, we consider the case in which the profit of each job is equal to its length, that is, the time interval between its release time and deadline. For this case, we prove that for \( m = 2 \) and \( m \geq 3 \), the competitive ratios of a greedy algorithm are at most \( 4/3 \) and at most \( 3 \), respectively. Also, for each \( m \geq 2 \), we show a lower bound on the competitive ratio of any deterministic algorithm.

1 Introduction

The interval scheduling problem is one of the variants of the scheduling problem, which has been widely studied. One of the most basic definitions is as follows: We have \( m \geq 1 \) identical machines and jobs are given. A job is characterized by the release time, deadline and weight (or value). To complete a job, we must start to process it at its release time on a machine of the \( m \) machines, and continue processing it until its deadline on that machine. That is, the processing time (or length) of the job is the time interval between its release time and deadline. The number of jobs which can be processed on one machine at a time is at most one. The objective of an algorithm is to maximize the total weight of completed jobs. There are many applications of the interval scheduling problem, such as bandwidth allocation and vehicle assignment (see e.g., [13, 14]). Many variants of this problem have been proposed and extensively studied. Furthermore, research on online settings has also been considered. In an online variant of the interval scheduling problem, a job arrives at its release time and an online algorithm must decide whether it processes the job before the next job arrives. The performance of online algorithms is evaluated using competitive analysis [3, 19]. For any input, if the total weight gained by an optimal offline algorithm is at most \( c \) times that gained by an online algorithm, the online algorithm is \( c \)-competitive.

In this paper, we introduce a novel variant of the interval scheduling problem. In many existing variants of the interval scheduling problem, jobs (or users) require resources for an algorithm, and
the algorithm assigns the required resources of a machine to the job. Thus, the number of jobs assigned to one machine at a time is subject to the maximum amount of resources of the machine. The amount is generally one; that is, at most one job can be processed at a time on one machine in most variants. Therefore, we can regard such existing variants as formulating resource reservation requests by users, who designate the amount of resources they want to use in advance and the responses to them. However, it is not always possible for users to designate the amount of resources they want when they issue requests. Additionally, there are not necessarily sufficient resources of a machine to meet users’ requests. Thus, we focus on a best-effort method to manage situations, which is often considered paired with resource reservation methods. In this method, the amount of resources of a machine is always invariant and the resources are equally shared by users who want to use the resources at the same time. Then, we formulate best-effort requests and responses to them as a variant of the interval scheduling problem. Specifically, we remove the capacity constraints from machines in our variant, which makes it possible to assign jobs unlimitedly on one machine at a time. To the best of our knowledge, this is the first such formulation of the interval scheduling problem. This is the first formulation as the interval scheduling problem. A suggestion “the first such formulation of the interval scheduling problem.” Consider a given job as a user’s request. If a machine processes the request using sufficient resources, the user is sufficiently satisfied with the result obtained from the process. Conversely, if there are not sufficient resources to process the request, the user is less satisfied with the result than usual. Then, the objective of our variant is to maximize the total satisfaction gained by users. Bandwidth allocation in networks is one of the most suitable examples for best-effort requests and responses. In this example, the total bandwidth which may be supplied to users on the same communication link is fixed in advance, and all users share the bandwidth. Hence, the fewer users which use the communication link at a time, the greater the bandwidth which each one can use, which means that the effective speed of the communication link is higher for the users. Conversely, the more people there are using link simultaneously, the lower the effective speed for each user. As a result, if the bandwidth for a user is high, then the user’s satisfaction is high. Otherwise, it is low. Best-effort requests and responses such as bandwidth allocation could happen in many cases, for example, the use of facilities, such as swimming pools and gyms, passenger trains without reservations, and buffet style meals. Therefore, we have sufficient incentives to study our variant.

**Our Results.** In this paper, we propose and analyze a novel variant of the interval scheduling problem. We study online algorithms for this problem. Specifically, in the case where the profits of jobs are arbitrary: that is, the profits are not relevant to the lengths of jobs, we show that the competitive ratio of any deterministic algorithm is unbounded. Then, we introduce the profits of jobs are equal to their lengths, which is a more natural case, called the uniform profit case. In this case, the total amount of time during which at least one job is scheduled on a machine is equal to the total amount of the satisfaction gained on the machine. That is, the objective of this case can be regarded as maximizing the working hours of all the machines. We analyze the performance of a greedy algorithm $GR$ in this case. Since $GR$ is a significant algorithm from a practical point of view, it is worthwhile to evaluate its performance. When $m = 2$ and $m \geq 3$, we show that the competitive ratios of $GR$ are at most $4/3$ and at most 3, respectively. When $m = 2$, we prove that a lower bound on the competitive ratio of $GR$ is $4/3$. That is, for $m = 2$, our analysis of $GR$ is tight. Also, we show lower bounds of any deterministic online algorithms for each $m \geq 2$, which are summarized in Table 1 and Table 2 in Sec. 5.

**Related Results.** Much research on the interval scheduling problem has been conducted. Arkin and Silverberg [1] and Bouzina and Emmons [4] provided polynomial time algorithms to solve the interval scheduling problem.
There is also much research on online interval scheduling problems. If an online algorithm aborts a job \( J \) which was placed on a machine, then we say that the algorithm *preempts* \( J \). In the case in which preemption is allowed, Faigle and Nawijn [9] designed a 1-competitive algorithm to maximize the number of completed jobs. This algorithm was independently discovered by Carlisle and Lloyd [6] but used only for the offline setting. Moreover, for the variant in which the objective is to maximize the total weight of completed jobs, Woeginger [20] showed that no any competitive deterministic algorithm exists (even) for \( m = 1 \). Canetti and Irani [5] provided a randomized online algorithm whose competitive ratio is \( O(\log \Delta) \) and proved that a lower bound on the competitive ratio of any randomized algorithm is \( \Omega(\sqrt{\log \Delta / \log \log \Delta}) \), where \( \Delta \) is the ratio of the longest length to the shortest length. This result indicates that the competitive ratio of an online algorithm may become worse depending on a given input even if it is supported by randomization. Additionally, the setting in which the jobs are unit length has been extensively studied. For the one machine setting, Woeginger [20] designed a deterministic algorithm whose competitive ratio is at most 4 and showed that this is the best possible ratio. There has also been much work regarding randomized algorithms (e.g. [18, 16, 10, 8, 11, 12]). When \( m = 1 \), the current best upper and lower bounds on the competitive ratios of randomized algorithms are 2 by Fung et al. [12] and \( 1 + \ln 2 \geq 1.693 \) by Epstein and Levin [8], respectively. For \( m \geq 2 \), Fung et al. [11] proved that, if \( m \) is even, an upper bound is 2, and otherwise \( 2 + 2/(2m - 1) \). However, for \( m = 2 \), the current best lower bound is 2 by Fung et al. [10]. When each \( m \geq 3 \), Fung et al. [11] indicated that we can obtain a lower bound of \( 1 + \ln 2 \geq 1.693 \) in a similar manner to the lower bound of Epstein and Levin [8]. If preemption is not allowed, Lipton and Tomkins [15] proposed a randomized algorithm whose competitive ratio is \( O((\log \Delta)^{1+\epsilon}) \) and proved that a lower bound of any randomized algorithms is \( \Omega(\log \Delta) \).

For a job given in the interval scheduling problem, its length is equal to the length of the time between its release time and deadline. On the other hand, a variant in which the job length is generalized has also been studied. Specifically, a parameter *slack* \( \varepsilon > 0 \) is introduced, whose value is known to an algorithm in advance, and the length of a job is at most \( x \) times as long as the length of the time between its release time and deadline, in which \( x = 1/(1 + \varepsilon) \). In this variant, preemption is allowed and to complete a job, an algorithm must process it during its length by its deadline after its release time. For several \( m \), optimal online algorithms were designed [2, 7, 17], whose competitive ratios are \( 1 + 1/\varepsilon \).
2 Model Description

We have $m (\geq 2)$ identical machines. A list consisting of $n (\geq 1)$ jobs is provided as an input. A job $J$ is specified by a triplet $(r, d, v)$, where $r(J)$ is the release time of $J$, $d(J)$ is the deadline of $J$, and $v(J)$ is the profit of $J$. An algorithm $ALG$ must place each job onto one of the $m$ machines. It is possible to place more than one job at a time on one machine. The profit of a job is distributed uniformly between its release time and deadline, that is its interval, and the profit gained from a subinterval of a job decreases in reverse proportion to the number of jobs whose intervals intersect with the subinterval on the same machine. Specifically, the profit from the subinterval is defined as follows: For an algorithm $ALG$, if the numbers of jobs placed at any two points in an interval $(x, y)$ ($x < y$) are equal on $ALG$’s $a (\in [1, m])$th machine and $(x, y)$ does not contain any endpoint of the interval of a job placed on the machine after processing of the input, then we call the interval a $P$-interval on $ALG$’s $a$th machine. Also, let $k_{ALG}(a, x, y)$ denote the number of the jobs. If an algorithm $ALG$ places a job $J$ onto the $a$th machine, then we define $m_{ALG}(J) = a$. For an algorithm $ALG$ and a job $J$, suppose that the interval $(r(J), d(J))$ consists of $b (\geq 1)$ $P$-intervals $(x_i, x_{i+1})$ ($i = 1, \ldots, b - 1$) on $ALG$’s $m_{ALG}(J)$th machine such that $r(J) = x_1 < x_2 < \cdots < x_b = d(J)$. Then, we define the satisfaction (profit) which is yielded from $[x_i, x_{i+1}]$ of $J$ and $ALG$ gains as

$$V_{ALG}(J, i) = \frac{x_{i+1} - x_i}{d(J) - r(J)} \frac{v(J)}{k_{ALG}(m_{ALG}(J), x_i, x_{i+1})}$$

(see an example in Fig. 1). We define the satisfaction (profit) of $J$ gained by $ALG$ as

$$V_{ALG}(J) = \sum_{i=1}^{b-1} V_{ALG}(J, i).$$

The profit of $ALG$ for an input $\sigma$ is defined as

$$V_{ALG}(\sigma) = \sum_{J \in \mathcal{L}} V_{ALG}(J),$$

where $\mathcal{L}$ is a list consisting of the $n$ given jobs. The objective is to maximize the total satisfaction of the $n$ jobs.

In this paper, we consider an online variant of this problem. Specifically, $n$ jobs are given one by one. The jobs are not necessarily given in order of release time. An online algorithm must place a given job to a machine before the next job is given. Once a job is placed on a machine, it cannot be removed later. That is, preemption is not allowed. The total number $n$ of given jobs is not known to the online algorithm, and it does not require this information until after all the jobs arrive. We say that the competitive ratio of an online algorithm $A$ is at most $c$ or $A$ is $c$-competitive if, for any input, the profit gained by an offline optimal algorithm $OPT$ is at most $c$ times the profit gained by $A$.

3 General Profit Case

In this section, we consider the case in which the profits of jobs are arbitrary. First, we consider the case $m = 2$ for better understanding of any $m \geq 3$. 
Execute one of the following two cases:

**Step 5:**

whose release time is the closest to \( z \) onto one machine and can obtain the profit of \( \sigma \). The following routine attempts to force each \( \sigma \) and \( \sigma' \)’s machine to place at least \( z \) jobs during an interval. \( OPT \) places only \( J \) onto one machine and can obtain the profit of \( v(J) \).

**Theorem 3.1** When \( m = 2 \), there does not exist any deterministic online algorithm whose competitive ratio is bounded.

**Proof.** Consider a deterministic online algorithm \( ON \). Let the profits of all the given jobs in this proof be one. First, we outline the routine to provide \( ON \) with an input. If there exist at least \( z(>0) \) jobs placed during a time interval on each machine, and a new job \( J \) is included in the interval, then \( ON \) can obtain a profit of at most \( v(J)/(z+1) \) from \( J \). The following routine attempts to force each \( ON \)’s machine to place at least \( z \) jobs during an interval. \( OPT \) places only \( J \) onto one machine.

| Step | Description |
|------|-------------|
| **Step 1:** | Let \( a_1 > 0 \) and \( \text{MaxC} > 0 \) be sufficiently large integers. \( \ell := 1 \) and \( s_1 := 0 \). |
| **Step 2:** | Give 2 \( \cdot \) \( \text{MaxC} \) jobs \( J \) such that \( r(J) = 0 \) and \( d(J) = 1 \). Let \( x_1 \) \((x_2)\) denote the number of the 2 \( \cdot \) \( \text{MaxC} \) jobs which \( ON \) places onto the first (second) machine. Without loss of generality, we assume that \( x_1 \geq x_2 \). |
| **Step 3:** | If \( \ell = \text{MaxC} + 1 \), then finish. |
| **Step 4:** | Give \( a_\ell \) jobs \( J_{\ell,i} \) \((i = 0, 1, \ldots, a_\ell-1)\) such that \( r(J_{\ell,i}) = s_\ell + p_\ell i/a_\ell \), \( d(J_{\ell,i}) = s_\ell + p_\ell (i+1)/a_\ell \), in which \( p_1 = 1 \) and for any \( \ell \geq 2 \), \( p_\ell = d(J_{\ell-1,1}) - r(J_{\ell-1,1}) \). |
| **Step 5:** | Execute one of the following two cases: |
| **Case 5.1** \((ON \ does \ not \ place \ any \ of \ J_{\ell,i} \ onto \ the \ second \ machine)\): | Finish. |
| **Case 5.2** \( (\text{Otherwise}) \): | Let \( J_{\ell,i'} \) be the job which placed onto the second machine by \( ON \) and whose release time is the closest to \( s_\ell \). \( s_{\ell+1} := s_\ell + p_\ell i'/a_\ell \). Let \( a_{\ell+1} > 0 \) be an integer such that \((2 + \sum_{j=1}^{\ell} \frac{1}{x_{j+1}}} \cdot \frac{1}{x_{j+2}}) a_j / a_{\ell+1} \) is sufficiently small. \( \ell := \ell + 1 \) and go to Step 3. |

Let \( \sigma_{\text{FinC}} \) denote the value of \( \ell \) at a time when the routine finishes. Let \( \sigma_{\text{FinC}} \) denote the input given to \( ON \). For a given input \( \sigma_{\text{FinC}} \), an offline algorithm \( OFF \) places \( a_{\text{FinC}} \) jobs \( J_{\text{FinC},i} \) given at the time of the final execution, that is, the \( \text{FinC} \)th execution of Step 4, onto the second machine, and places the other jobs onto the first machine. Thus, for any \( \text{FinC} \leq \text{MaxC} \),

\[
V_{\text{OPT}}(\sigma_{\text{FinC}}) \geq V_{\text{OFF}}(\sigma_{\text{FinC}}) \geq a_{\text{FinC}}
\]

and

\[
V_{\text{OPT}}(\sigma_{\text{MaxC}+1}) \geq V_{\text{OFF}}(\sigma_{\text{MaxC}+1}) \geq a_{\text{MaxC}+1}
\]
Conversely, we consider the profit gained by ON. The number of jobs which are placed by ON on the $j$th machine for each $j \in \{1, 2\}$ immediately after Step 2 is $x_j$, and ON can gain profit $1/x_j$ per job. Since the profit of a job does not increase, the total profit which ON gains from all the jobs given in Step 2 is at most two at the end of the input.

By the definition of Case 5.2, jobs in the $\ell (\in [2, \text{FinC} - 1])$th execution of Step 4 are given in an interval where ON places jobs given in the $\ell - 1$st execution of Step 4 (Fig. 2). Thus, there exist $x_2 + \ell - 1$ jobs during the interval $[s_\ell, s_\ell + p_\ell]$ on ON’s second machine immediately before the $\ell$th execution of Step 4. Hence, in the case where Case 5.2 is executed after the $\ell$th Step 4, the profit of a job given in the $\ell$th Step 4 is at most $\max\{1/x_{\ell+1}, 1/x_{\ell+2}\}$ immediately after the $\ell$th execution of Step 4. Thus, the total profit of the jobs given in the $\ell$th Step 4 is at most $\max(1/x_{\ell+1}, 1/x_{\ell+2}) a_\ell$.

Conversely, if Case 5.1 is executed, the profit of a job given in the $\ell$th Step 4 is at most $1/a_{\text{FinC}}$. Thus, the total profit is at most $\sum_j a_j$. Additionally, $x_1 \geq \text{MaxC}$ because both $x_1 \geq x_2$ and $x_1 + x_2 = 2 \cdot \text{MaxC}$ by definition. Therefore, if the routine finishes in Case 5.1 when $\text{FinC} \leq \text{MaxC}$, we have

$$V_{\text{ON}}(\sigma_{\text{FinC}}) \leq 2 + \sum_{j=1}^{\text{FinC}-1} \max \left\{ \frac{1}{x_1 + 1}, \frac{1}{x_2 + j} \right\} a_j + \frac{a_{\text{FinC}}}{x_1 + 1}$$

where the second inequality follows from the fact that $x_1 \geq \text{MaxC}$. In addition, if the routine finishes in Step 3, that is, $\text{FinC} = \text{MaxC} + 1$, we have

$$V_{\text{ON}}(\sigma_{\text{MaxC}+1}) \leq 2 + \sum_{j=1}^{\text{MaxC}} \max \left\{ \frac{1}{x_1 + 1}, \frac{1}{x_2 + j} \right\} a_j$$

By the above argument, if $\text{FinC} \leq \text{MaxC}$,

$$\frac{V_{\text{OPT}}(\sigma_{\text{FinC}})}{V_{\text{ON}}(\sigma_{\text{FinC}})} \geq \frac{a_{\text{FinC}}}{2 + \text{FinC}} \frac{\max \left\{ \frac{1}{x_1 + 1}, \frac{1}{x_2 + j} \right\} a_j + a_{\text{FinC}}/\text{MaxC}}{a_{\text{FinC}} \delta_{\text{FinC}} + a_{\text{FinC}}/\text{MaxC}} \geq \frac{1}{\delta_{\text{FinC}} + 1/\text{MaxC}},$$

where $\delta_{\text{FinC}} = (2 + \sum_{j=1}^{\text{FinC}-1} \max \left\{ \frac{1}{x_1 + 1}, \frac{1}{x_2 + j} \right\})/a_{\text{FinC}}$. Note that as both $\delta_{\text{FinC}}$ and $1/\text{MaxC}$ tend to zero, $1/(\delta_{\text{FinC}} + 1/\text{MaxC})$ tends to infinity. In a similar manner,

$$\frac{V_{\text{OPT}}(\sigma_{\text{MaxC}+1})}{V_{\text{ON}}(\sigma_{\text{MaxC}+1})} \geq \text{MaxC}.$$

\[\square\]

**Theorem 3.2** For any $m$, there does not exist a competitive deterministic algorithm.

**Proof (sketch).** In Theorem 3.1, we use the routine to force the numbers of jobs existing during an interval on both machines of an online algorithm ON to approach a sufficiently large integer MaxC.
After the end of the input, we need to evaluate the profit from each job by OPT using the profits yielded from intervals of jobs scheduled by GR to analyze the performance of GR. Then, we classify intervals (or points) in a job J by GR or OPT into the following four categories depending on the behaviors of GR and OPT for J.

For any two intervals $I = [t_1, t_2]$ and $I' = [t'_1, t'_2]$, we say that $I$ intersects with $I'$ if $t'_1 < t_2$ and $t_1 < t'_2$. For any job $J$, we call the interval $[r(J), d(J)]$ the interval of $J$. If an algorithm ALG
places two jobs onto the same machine and they intersect, then we say that they overlap. For any interval $I = [t, t']$, we call the value of $t' - t$ the length of $I$, written as $|I|$.

We give the definition of a greedy algorithm $GR$ and analyze its performance in this section. $GR$ places a given job $J$ onto the machine on which $GR$ gains the largest profit from $J$. The tie-breaking rule selects the minimum indexed machine.

For ease of analyzing, we introduce the following idea. Suppose that two jobs $J_1$ and $J_2$ are placed onto the same machine, and they overlap in an interval $I$. Also, suppose that $J_1$ is the first job placed in $I$ on the machine. Then, pretend that the profits from $I$ of $J_1$ and $J_2$ are $|I|$ and zero, respectively. That is, we pretend that a job which is placed chronologically first in an interval on a machine monopolizes the machine power in the interval. Note that in the uniform profit case, the total profit gained from an interval of jobs placed on a machine depends not on how large the number of the jobs in the interval is but on whether there exists at least one job placed in the interval. That is why this assumption does not affect the profit of any algorithm.

### 4.2 Overview of Analysis

To evaluate the performance of $GR$, that is, its competitive ratio, we bound the profit of $OPT$ at the end of the input using that of $GR$. Then, we classify intervals of jobs placed by either $GR$ or $OPT$ into four categories.

For any job $J$ and any interval $I \subseteq [r(J), d(J)]$, if the profit gained from $I$ of $J$ by $GR$ is zero and that by $OPT$ is $|I|$, then we call $I$ of $J$ an $OPT$ extra interval of $J$ (denoted as an $oe$-interval, for short) (see Fig. 3). Also, if the profit gained from $I$ of $J$ by $OPT$ is zero and that by $GR$ is $|I|$, then we call $I$ of $J$ a $GR$ extra interval of $J$ (a $ge$-interval, for short). If the profits gained from $I$ of $J$ by $GR$ and $OPT$ are both $|I|$, we call $I$ of $J$ a common interval of $J$ (a $c$-interval, for short). For ease of presentation, we call an interval which is a $c$-interval or a $ge$-interval a profit interval (a $p$-interval, for short). If the profits gained from $I$ of $J$ by $GR$ and $OPT$ are both zero, we call $I$ of $J$ a non-profit interval of $J$ (an $n$-interval, for short). Further, we call a point in an $oe$-interval (a $ge$-interval, a $c$-interval, and a $p$-interval, respectively) of $J$ an $oe$-fraction (a $ge$-fraction, a $c$-fraction, and a $p$-fraction, respectively) of $J$.

We evaluate the competitive ratio of $GR$ by “assigning” $p$-fractions (i.e., $p$-intervals) to all $oe$-fractions (i.e., $oe$-intervals) according to a routine, which is defined later. This “assignment” is realized by some functions. Let $V_{oe}(\sigma)$ be the total length of $oe$-intervals to which $c$-intervals are assigned. Let $V_{ge}(\sigma)$ be the total length of $oe$-intervals to which $ge$-intervals are assigned. Also, let $V_c(\sigma)$ be the total length of $c$-intervals and $V_{ge}(\sigma)$ be the total length of $ge$-intervals. Then, we have by definition,

$$V_{GR}(\sigma) = V_c(\sigma) + V_{ge}(\sigma)$$ \hspace{1cm} (1)

and

$$V_{OPT}(\sigma) = V_c(\sigma) + V_{oe}(\sigma) + V_{oe}(\sigma).$$ \hspace{1cm} (2)

We will show the following three properties of the assignments by the routine:

1. Each $oe$-fraction is assigned a $p$-fraction,
2. a $c$-fraction of a job given to $GR$ is assigned at most twice, and
3. a $ge$-fraction is assigned at most three times.

To show these, we will construct sequentially three functions $M_1, M_2$ and $M_3$ from $oe$-intervals to $p$-intervals satisfying the following properties: Initially, for any $oe$-fraction $f$ and any $i \in \{1, 2, 3\}$,
$M_i(f) = \emptyset$. At the end of the input, for any $oe$-fraction $f$, $M_1(f) \cup M_2(f) \cup M_3(f) \neq \emptyset$. There exists a $p$-fraction $f'$ such that $M_1(f) = f'$ if $M_1(f) \neq \emptyset$. There exists a $ge$-fraction $f'$ such that $M_2(f) = f'$ if $M_2(f) \neq \emptyset$. There exists a $p$-fraction $f''$ such that $M_3(f) = f''$ if $M_3(f) \neq \emptyset$. For any $oe$-fractions $f$ and $f'(\neq f)$ and any $i \in \{1, 2, 3\}$, $M_i(f) \cap M_i(f') = \emptyset$. Then, we have by these functions,

$$V_{oe}(\sigma) \leq 2V_c(\sigma)$$

and

$$V_{oe'}(\sigma) \leq 3V_{ge}(\sigma).$$

By Eq. (2), we have

$$V_{OPT}(\sigma) = V_c(\sigma) + V_{oe}(\sigma) + V_{oe'}(\sigma) \leq V_c(\sigma) + 2V_c(\sigma) + 3V_{ge}(\sigma) \quad \text{(by Eqs. (3) and (4))}$$

$$= 3(V_c(\sigma) + V_{ge}(\sigma)) = 3V_{GR}(\sigma), \quad \text{(by Eq. (1))}$$

which leads to the following theorem:

Figure 3: An example in which five jobs $J_i (i \in [1, 5])$ are given when $m = 2$. The top (bottom) figure presents jobs placed by $GR$ ($OPT$). $c$-intervals ($ge$-intervals, $oe$-intervals and $n$-interval, respectively) are shown in blue (green, red and yellow, respectively) squares. The intervals $[1, 2]$ of $J_4$ and $J_2$ by $GR$ overlap but the intervals $[1, 2]$ of $J_4$ by $OPT$ do not. Thus, this interval of $J_4$ is an $oe$-interval. Also, neither $GR$ nor $OPT$ gains the profit from the interval $[0, 1]$ of $J_4$ and thus this interval is an $n$-interval.

**Theorem 4.1** For any $m \geq 2$, the competitive ratio of $GR$ is at most three.
4.3 Analysis of GR

For any job $J$ and any point $t \in [r(J), d(J)]$, let $E(J, t)$ denote the total length of oe-intervals of $J$ in the interval $[r(J), t]$. For any job $J$, any job $J’$ given before $J$, any interval $[t_1, t_2]$ and any $a(\in [1, m])$, let $P_a(J, J’, t_1, t_2)$ denote the total length of p-intervals of GR’s jobs placed on the ath machine which are in $[t_1, t_2]$ immediately after $J$ is placed and are not intersecting with any n-interval of $J$. For any $a(\in [1, m])$, any job $J$, any job $J’$ given before $J$, and any point $t \in [r(J’), d(J’)]$, define $h_a(J, J’, t) = t’$ in which $t’$ is the point such that $P_a(J, J’, r(J’), t’) = E(J’, t)$ and $t’ \in [r(J’), d(J’)]$ immediately after $J$ is placed onto the machine. ($t’$ exists by Lemma 4.2, which is shown later.) For any $i \in \{1, 2, 3\}$ and any p-fraction $f’$, define $M_i^{-1}(f’) = \{ f \mid M_i(f) = f’ \}$. We say that a $t$-fraction $f’$ such that $M_1^{-1}(f’) = \emptyset$ is 1-assignable. We say that a ge-fraction $f’$ such that $M_2^{-1}(f’) = \emptyset$ is 2-assignable. We say that a ge-fraction $f’$ such that $M_2^{-1}(f’) \neq \emptyset$ and $M_1^{-1}(f’) = \emptyset$ is 1-assignable. If a p-fraction is 1-assignable or 2-assignable, we say that it is assignable. Now we give the definition of the routine mentioned in the previous section. For better understanding assignments, we give examples in Appendix A.

**AssignmentRoutine**

Consider a moment immediately after the jth job $J_j$ is placed. $\mathcal{J} := \{ J_j \}$ (the set of $J_j$ plus each job $J_j’$ ($j’ \leq j - 1$) whose interval intersects with the interval of $J_j$). For any oe-fraction $f$ of each $J \in \mathcal{J}$, execute the following.

**Step 1:** For each $i \in \{1, 2, 3\}$, $M_i(f) := \emptyset$. $t_1 := h_1(J_j, J, t)$, in which $f$ exists at a point $t$.

**Step 2:** Execute one of the following two cases.

- **Case 2.1 (An assignable p-fraction $f_1$ exists at $t_1$):** If $f_1$ is 1-assignable, $M_1(f) := f_1$.
- Otherwise, if $f_1$ is 2-assignable, $M_2(f) := f_1$.

- **Case 2.2 (No assignable p-fraction exists at $t_1$):** By Lemma 4.3, there exists a p-fraction $f_a$ at the point $t_a$ on some $a(\in \{1, m\})$th machine such that $M_3^{-1}(f_a) = \emptyset$, in which $t_a = h_a(J_j, J, t)$.

In the following, we first show the existence of $t_a$ in Case 2.2. Next, we show that there exists $p_a$ in Case 2.2. That is, we prove that the routine can assign a p-fraction to each oe-fraction.

**Lemma 4.2** For any $a(\in [1, m])$, any job $J$, any job $J’$ which is given before $J$, and any point $t \in [r(J’), d(J’)]$, there exists the point $t’$ such that $h_a(J, J’, t) = t’$ and $t’ \in [r(J’), d(J’)]$ immediately after $J$ is placed.

**Proof.** Suppose that $J, J’$ and $t$ satisfy the conditions of the statement of this lemma. By the definition of GR, GR chooses the machine $m_{GR}(J’)$ when placing $J’$ so that it gains the largest profit from $J’$. That is, GR chooses the machine so that the total length of the intervals of jobs which were already placed before placing $J’$ and which are intersecting with the interval of $J’$ is minimized. Hence, the total length of p-intervals on the $a(\in [1, m])$th machine which are intersecting with the interval of $J’$ is at least the length of oe-intervals or n-intervals of $J’$. Namely, the total length of p-intervals on the ath machine which are not in n-intervals of $J’$ and are intersecting with the interval of $J’$ is at least the length of oe-intervals of $J’$. Therefore, there exists the point $t’$ such that $P_a(J, J’, r(J’), t’) = E(J’, t)$ and $t’ \in [r(J’), d(J’)]$.

**Lemma 4.3** Case 2.2 is executable. That is, when Case 2.2 is executed for an oe-fraction $f$, $f$ can be assigned a p-fraction $f_a$ such that $M_3^{-1}(f_a) = \emptyset$ immediately before executing Case 2.2.
Proof. Suppose that the routine executes Step 2 for a placed job $J'$, a job $J$ given before $J'$ and an oe-fraction $f$ of $J$. Also, suppose that $f$ exists at a point $t \in [r(J), d(J)]$ and $t_1 = h_1(J', J, t)$ holds.

First, we evaluate the number of $p$-fractions located at $t_1$ which are required for the execution of Step 2. Let $x (y, z)$ denote the number of jobs whose $c$-fractions (oe-fractions, ge-fractions) located at $t_1$ immediately after $J'$ is placed. Since $OPT$ places jobs whose $c$-fractions or oe-fractions are located at $t_1$, the number of machines is at least $x + y$. The number of machines on which $GR$'s jobs are placed at $t_1$ is $x + z$. Let $w$ denote the number of machines on which $GR$ places no jobs at $t_1$. Then,

$$w \geq \max\{(x + y) - (x + z), 0\} = \max\{y - z, 0\}. \quad (5)$$

By definition, the value of $h_1$ for an oe-fraction of a job is different from that for another oe-fraction of the job (i.e., the two oe-fractions are located at distinct points). Hence, there is a one-to-one correspondence between oe-fractions of a job and the values of $h_1$ for the job. That is, the number of executions of Step 2 for a job with $t_1$ is exactly one. Further, the interval of such job must include $t_1$ and $t_1$ is not in an $n$-interval of the job by the definition of $h_1$. Therefore, the number of $p$-fractions which are located at $t_1$ and required for the assignments is at most the number of jobs such that the interval of each of the jobs includes $t_1$ and $t_1$ is not in an $n$-interval of each of them, that is, at most $x + y + z$.

Second, we evaluate the number of $p$-fractions which can be used for the assignments at the execution of Step 2. The numbers of one $c$-fraction and one $ge$-fraction which can be assigned to oe-fractions are one and two, respectively (in Case 2.1). Hence, the number of $c$-fractions and $ge$-fractions which are located at $t_1$ and used for the assignments is at least $x + 2z$. Also, for the $a$th machine on which $GR$ places no jobs at the execution of Step 2, the routine assigns a $p$-fraction located at the point $t_a = h_a(J', J, t)$ (in Case 2.2). In the same way as the above argument, by the definition of $h_a$, there is a one-to-one correspondence between $p$-fractions of a job and the values of $t_a$ for the job and thus, there is a one-to-one correspondence between the oe-fractions of $J$ and the $p$-fractions of $J$. By summing up the above numbers, the number of $p$-fractions for the assignments at the execution of Step 2 is at least $x + 2z + w$. Thus, we have

$$x + 2z + w \geq x + 2z + y - z = x + y + z,$$

in which the inequality follows from Eq. (5). Therefore, when the routine executes Case 2.2 for $f$, it is executable. \hfill \square

### 4.4 Upper Bound for $m = 2$

When $m = 2$, we also evaluate the competitive ratio of $GR$ by assigning $p$-fractions to all oe-fractions. In this case, we obtain a better upper bound on the competitive ratio of $GR$ than one for general $m$ by implementing more detailed assignments. If the routine assigns one ge-fraction to one oe-fraction, we say that the routine ge-assigns the ge-fraction to the oe-fraction. Also, if the routine assigns three $p$-fractions to one oe-fraction, we say that the routine $3p$-assigns each of the $p$-fractions to the oe-fraction. We will show the following three properties by the assignments according to the routine defined later:

1. Each oe-fraction is ge-assigned or $3p$-assigned,
2. a $c$-fraction of a job given to $GR$ is $3p$-assigned at most once, and
3. a ge-fraction is ge-assigned at most once and is $3p$-assigned at most once.
We will show them by sequentially constructing two functions \(N_1\) and \(N_2\) from oe-intervals to \(p\)-intervals satisfying the following properties: Initially, for any oe-fraction \(f\) and any \(i \in \{1,2\}\), \(N_i(f) = \emptyset\). At the end of the input, for any oe-fraction \(f\), \(N_1(f) \cup N_2(f) \neq \emptyset\). There exist three distinct p-fractions \(f_1, f_2\) and \(f_3\) such that \(N_1(f) = \{f_1, f_2, f_3\}\) if \(N_1(f) \neq \emptyset\). There exists a ge-fraction \(f'\) such that \(N_2(f) = f' \) if \(N_2(f) \neq \emptyset\). For any oe-fractions \(f\) and \(f' (\neq f)\) and any \(i \in \{1,2\}\), \(N_i(f) \cap N_i(f') = \emptyset\). Let \(V_{oe}(\sigma)\) denote the total length of oe-intervals to which the routine 3p-assigns, and let \(V_{oe}(\sigma)\) denote the total length of oe-intervals to which the routine ge-assigns. Thus, 

\[
V_{oe}(\sigma) \leq V_{GR}(\sigma)/3
\]

and

\[
V_{oe}(\sigma) \leq V_{ge}(\sigma).
\]

Then, using these inequalities, we have

\[
V_{OPT}(\sigma) = V_{\epsilon}(\sigma) + V_{oe}(\sigma) + V_{oe}(\sigma) \\
\leq V_{\epsilon}(\sigma) + V_{GR}(\sigma)/3 + V_{ge}(\sigma) = \frac{4}{3}V_{GR}(\sigma).
\]

Therefore, we have the following theorem:

**Theorem 4.4** When \(m = 2\), the competitive ratio of \(GR\) is at most 4/3.

For any \(i \in \{1,2\}\) and any p-fraction \(f'\), define \(N_i^{-1}(f') = \{f \mid N_i(f) = f'\}\). We say that a p-fraction \(f'\) is 1-assignable if \(N_1^{-1}(f') = \emptyset\). Also, we say that a ge-fraction \(f'\) is 2-assignable if \(N_2^{-1}(f') = \emptyset\). Now we give the definition of the routine to construct the above two functions.

### AssignmentRoutine2

Consider a moment immediately after a job \(J\) is placed. For any oe-fraction \(f\) of \(J\), execute the following.

**Step 1:** \(m_2 := m_{GR}(J)\) and \(m_1 := \{1,2\} \setminus \{m_2\}\). \(t_1 := h_{m_1}(J,J,t)\), in which \(f\) exists at a point \(t\).

**Step 2:** Let \(f'\) be the p-fraction at \(t\) on the \(m_2\)th machine (\(f'\) exists by the definition of oe-fractions). Execute one of the following two cases.

- **Case 2.1 (\(f'\) is 2-assignable):** \(N_2(f) := f'\).
- **Case 2.2 (Otherwise):** \(N_1(f) := \{f', f_1, f_2\}\), in which \(f_1\) is the p-fraction at \(t_1\) on \(GR\)'s \(m_1\)th machine (\(f_1\) exists by Lemma 4.2), and \(f_2\) is the p-fraction at \(t_1\) on \(GR\)'s \(m_2\)th machine (\(f_2\) exists because the interval of \(J\) contains \(t_1\) by the definition of \(h_{m_1}\)). (By Lemma 4.5, \(f', f_1\) and \(f_2\) are 1-assignable.)

### Lemma 4.5

**Case 2.2 is executable.** That is, when Case 2.2 is executed for an oe-fraction \(f\), \(f\) can be assigned 3 p-fractions (i.e., 3p-assigned) each of which is 1-assignable immediately before executing Case 2.2.

**Proof.** We prove the lemma by induction on the number of given jobs. The statement of the lemma is clearly true before the first job is given. We assume that Case 2.2 is executable immediately before a job \(J\) is given, and show that it is executable for \(J\) as well. Then, suppose that the routine executes Step 2 for an oe-fraction \(f\) of a job \(J\) at a point \(t\). Let \(f'\) be the p-fraction of a job \(J'\) which is given before \(J\) at \(t\) on the \(m_{GR}(J)\)th machine. First of all, consider an oe-fraction \(f''\) of a job \(J''\) to which a p-fraction at \(t\) is 3p-assigned. Suppose that \(f''\) is located at a point \(t''\).
(i) \((t'' \neq t)\): Since \(t'' \neq t\), \(t = h_{m_1}(J'', J''t'')\), in which \(m_1 = \{1, 2\} \setminus m_{GR}(J'')\). Suppose that \(\hat{J}(\neq J'')\) is a job given before \(J\) and after \(J''\). When Case 2.2 is executed for \(J''\), there exists jobs including \(t\) on the both machines, which are used for \(3p\)-assignments to \(J''\). It follows that if the interval of \(\hat{J}\) contains \(t\), then \(t\) is in an \(n\)-interval of \(\hat{J}\). Thus, an oe-fraction not at \(t\) of \(\hat{J}\) is not \(3p\)-assigned \(p\)-fractions at \(t\) by the definition of Case 2.2 of the routine. Hence, in this case, the number of \(p\)-fractions at \(t\) used for \(3p\)-assignments is exactly two by the definition of Case 2.2. Then, there exist at least three jobs (i.e., \(J, J''\) and \(J''t''\)) whose intervals contain \(t\) and \(m = 2\), which means that there exists at least one ge-fraction at \(t\).

(ii) \((t'' = t)\): Since both \(f\) and \(f''\) exist at \(t\) and \(m = 2\), at least one ge-fraction exists at \(t\). Also, all the oe-fractions at \(t\) are only \(f\) and \(f''\). Thus, the routine ge-assigns to \(f''\) by executing Case 2.1. That is, Case 2.2 is not executed. Hence, the routine does not \(3p\)-assign \(p\)-fractions at \(t\) to an oe-fraction of a job given before \(J\) at \(t\). Thus, \(f\) is 1-assignable.

Now we are ready to show that it is possible for the routine to execute Case 2.2 for \(f\). First, we discuss the case in which \(p\)-fractions at \(t\) are ge-assign. Since the number of jobs whose intervals contain \(t\) is at least three and \(m = 2\), the number of ge-fractions at \(t\) is equal to that of oe-fractions at \(t\). By the definition of Case 2.1, oe-fractions not at \(t\) are not ge-assign ge-fractions at \(t\). Hence, there exists at least one ge-fraction at \(t\) which is 2-assignable and Case 2.1 is executed for \(f\).

Second, we discuss the case in which \(p\)-fractions at \(t\) are not ge-assign. We first consider the case in which \(p\)-fractions at \(t\) are \(3p\)-assigned. By the above discussion (i), there exists at least one ge-fraction at \(t\). When \(J\) is given, the ge-fraction has not been used for ge-assignments yet and the routine ge-assigns to \(f\) in Case 2.1.

Finally, we consider the case in which \(p\)-fractions at \(t\) are not \(3p\)-assigned. In this case, \(f'\) is 1-assignable when \(J\) is given. Let \(t_1 = h_{m_1}(J, J, t)\), in which \(m_1 = \{1, 2\} \setminus m_{GR}(J)\). By the above (i), an oe-fraction which can be \(3p\)-assigned \(p\)-fractions at \(t_1\) is only \(f\) of \(J\) (i.e., even if the interval of a job with another oe-fraction contains \(t_1\), \(t_1\) is in an \(n\)-interval of the job). An oe-fraction at \(t_1\) is not \(3p\)-assigned a \(p\)-fraction at \(t_1\) by above (ii). Therefore, \(p\)-fractions at \(t_1\) are 1-assignable and Case 2.2 is executable for \(f\) together with the 1-assignability of \(f'\).

We show that our analysis of \(GR\) for \(m = 2\) is tight in the following theorem.

**Theorem 4.6** When \(m = 2\), for any \(\varepsilon > 0\), the competitive ratio of \(GR\) is at least \(4/3 - \varepsilon\).

**Proof.** Consider the following input \(\sigma\). The first job \(J_1\) such that \(r(J_1) = 0\) and \(d(J_1) = 1\) is given. The second job \(J_2\) such that \(r(J_2) = 2 - \varepsilon\) and \(d(J_2) = 2\) is given, where \(0 < \varepsilon < 1\). The third job \(J_3\) such that \(r(J_3) = 1\) and \(d(J_3) = 2\) and the fourth job \(J_4\) such that \(r(J_4) = 0\) and \(d(J_4) = 2\) are given. \(GR\) places \(J_1\) and \(J_2\) on the first machine, and then places \(J_3\) and \(J_4\) on the second machine. \(OPT\) places \(J_1, J_2, J_3\) and \(J_4\) on one machine and places \(J_4\) on the other machine. Thus,

\[
\frac{V_{OPT}(\sigma)}{V_{GR}(\sigma)} = \frac{4}{3 + \varepsilon} = \frac{4}{3} - \varepsilon,
\]

in which \(\varepsilon = 4\varepsilon/(9 + 3\varepsilon)\).

\[\square\]

5 **Lower Bounds for Uniform Profit Case**

In this section, we show lower bounds on the competitive ratios of online algorithms for the uniform profit case. For better understanding, we first consider the case of \(m = 2\).
Theorem 5.1 When $m = 2$, the competitive ratio of any deterministic online algorithm is at least $(10 - \sqrt{2})/7 \geq 1.226$.

Proof. Consider an online algorithm $ON$. The first given job is $J_1$ such that $r(J_1) = 0$ and $d(J_1) = 1$. The second job is $J_2$ such that $r(J_2) = 1 + x$ and $d(J_2) = 2 + x$. Note that $x$ is set later. Without loss of generality, we may assume that both $ON$ and $OPT$ place $J_1$ onto the first machine.

In the following, we use two inputs. First, we consider the case where $ON$ places $J_1$ and $J_2$ on two different machines. That is, suppose that $ON$ places $J_2$ on the second machine. Then, the third job $J_3$ such that $r(J_3) = 0$ and $d(J_3) = 2 + x$ is given, and no further job arrives. We call this input $σ_1$. If $ON$ places $J_3$ onto the first machine, we have $V_{ON}(σ_1) = 2 + x + 1 = 3 + x$. $ON$ also gains the same profit if $ON$ places $J_3$ onto the second machine. On the other hand, the machine onto which $OPT$ places both $J_1$ and $J_2$ is different from that onto which $J_3$ is placed. Thus, $V_{OPT}(σ_1) = 2 + 2 + x = 4 + x$. By the above argument,

$$\frac{V_{OPT}(σ_1)}{V_{ON}(σ_1)} = \frac{4 + x}{3 + x}. \quad (6)$$

Second, we consider the case where $ON$ places $J_1$ and $J_2$ onto the first machine. The third job $J_1'$ such that $r(J_1') = 1 - y$ and $d(J_1') = 1 + x$ and the fourth job $J_2'$ such that $r(J_2') = 1$ and $d(J_2') = 1 + x + y$ are given, where $y$ is fixed later. No further job is given; we call this input $σ_2$. We first consider the case where $ON$ places $J_1'$ and $J_2'$ on different machines. If $J_1'$ is placed onto the first machine, on which $J_1$ and $J_2$ are placed,

$$V_{ON}(σ_2) = 1 + x + 1 + x + y = 2 + 2x + y. \quad (7)$$

$ON$ gains the same profit if $J_2'$ is placed onto the first machine. Next, we consider the case in which $ON$ places $J_1'$ and $J_2'$ onto the machine. If the machine is the second one, then it is clear that $ON$ gains larger profits than it does in the other case. Hence,

$$V_{ON}(σ_2) = 2 + x + 2y. \quad (8)$$

Now, set $y = x$ and we have $V_{ON}(σ_2) = 2 + 3x$ by Eqs. (7) and (8). On the other hand, $OPT$ places both $J_1$ and $J_2$ onto the first machine and both $J_2$ and $J_1'$ onto the second machine. Thus, $V_{OPT}(σ_2) = 2(1 + x + y) = 2 + 4x$. By the above argument,

$$\frac{V_{OPT}(σ_2)}{V_{ON}(σ_2)} = \frac{2 + 4x}{2 + 3x}. \quad (9)$$

Therefore, by Eqs. (6) and (9),

$$\frac{V_{OPT}(σ)}{V_{ON}(σ)} \geq \min \left\{ \frac{4 + x}{3 + x}, \frac{2 + 4x}{2 + 3x} \right\} = \frac{4 + \sqrt{2}}{3 + \sqrt{2}} = \frac{10 - \sqrt{2}}{7},$$

where we choose $x = \sqrt{2}$. \qed

The following theorem provides lower bounds for $m \geq 3$ by generalizing the input used to prove Theorem 5.1.

Theorem 5.2 The competitive ratio of any deterministic algorithm is at least $1.101$. It is better for fixed $m$ and then refer to Table 2 for details.
Proof. Consider an online algorithm $ON$. First, $\lceil \frac{m}{2} \rceil (= m')$ jobs $J_{1,j}$ ($j = 1, \ldots, m'$) such that $r(J_{1,j}) = 0$ and $d(J_{1,j}) = 1$ are given. Let $S_1$ denote the set of these jobs. Next, $m'$ jobs $J_{2,j}$ ($j = 1, \ldots, m'$) such that $r(J_{2,j}) = 1 + x$ and $d(J_{2,j}) = 2 + x$ arrive. $x$ is set later. Let $S_2$ denote the set of these jobs. Let $A$ be the set of machines onto which $ON$ places at least one job from $S_1$ and $S_2$. Let $B$ be the set of machines onto which $ON$ places at least one job from $S_1$ but $ON$ does not place a job from $S_2$. Let $B'$ be the set of machines onto which $ON$ places at least one job from $S_2$ but $ON$ does not place a job from $S_1$. Let $C$ be the set of machines onto which $ON$ places no jobs from either $S_1$ or $S_2$. Let the number of machines in $A, B, B'$ and $C$ denote $a, b, b'$ and $c$, respectively. Then, we have

$$a + b + b' + c = m,$$  \hspace{1cm} (10)

$$a + b \leq m',$$ \hspace{1cm} (11)

and

$$a + b' \leq m'.$$ \hspace{1cm} (12)

In the following, we provide two inputs and first consider input $\sigma_1$. $\lceil \frac{m}{2} \rceil (= m'')$ jobs $J_{3,j}$ ($j = 1, \ldots, m''$) such that $r(J_{3,j}) = 0$ and $d(J_{3,j}) = 2 + x$ are issued. Let $S_3$ be the set of these jobs.

Since $a \leq m'$ by definition, $b + b' + c = m - a \geq m - m' = m''$ by Eq. (10). Thus, $ON$ can place each job in $S_3$ onto each machine in $B \cup B' \cup C$. In this way, $ON$ gains more profits than placing the jobs onto machines in $A$. Then, $ON$ places each of $\min\{c, m''\}$ jobs from $S_3$ onto each machine in $C$, and places the remaining $\max\{m'' - c, 0\}$ jobs onto machines from either $B$ or $B'$. Thus,

$$V_{ON}(\sigma_1) \leq 2a + b + b' + (2 + x)\min\{c, m''\} + (1 + x)\max\{m'' - c, 0\}$$

$$= a + m - c + (2 + x)\min\{c, m''\} + (1 + x)\max\{m'' - c, 0\} \quad \text{(by Eq. (10)).}$$

When $c \geq m''$,

$$V_{ON}(\sigma_1) \leq a + m - c + (2 + x)m'' + (1 + x)0 \leq a + m - m'' + (2 + x)m'' = a + m + (1 + x)m'',$$ \hspace{1cm} (13)

where the second inequality follows from $c \geq m''$. When $c < m''$,

$$V_{ON}(\sigma_1) \leq a + m - c + (2 + x)c + (1 + x)(m'' - c) = a + m + (1 + x)m''.$$ \hspace{1cm} (14)

On the other hand, for each $j = 1, \ldots, m'$, $OPT$ places both $J_{1,j}$ and $J_{2,j}$ onto the $j$th machine, and places each $J_{3,j}$ onto each of the remaining machines. Hence,

$$V_{OPT}(\sigma_1) = 2m' + (2 + x)m''.$$ \hspace{1cm} (15)

Second, consider the input $\sigma_2$. After $J_{2,j}$ ($j = 1, \ldots, m'$) are given, $m''$ jobs $J'_{1,j}$ ($j = 1, \ldots, m''$) arrive such that $r(J'_{1,j}) = 1 - x$ and $d(J'_{1,j}) = 1 + x$. Let $S'_1$ denote the set of these jobs. Then, $m''$ jobs $J'_{2,j}$ ($j = 1, \ldots, m''$) arrive such that $r(J'_{2,j}) = 1$ and $d(J'_{2,j}) = 1 + 2x$. Let $S'_2$ denote the set of these jobs. $\bar{b}$ denotes the number of machines from $B'$ onto which $ON$ places at least one job from $S'_1$ plus the number of machines from $B$ onto which $ON$ places at least one job from $S'_2$. If $ON$ places at least one job from $S'_1$ ($S'_2$) onto a machine from $B'$ ($B$), then $ON$ gains the profit of $2x$ per machine in addition to the profits of jobs in $S_1$ and $S_2$. Let $\tilde{c}_1$ denote the number of machines from $C$ each of which $ON$ places one job from $S'_1 \cup S'_2$ onto. Then, $ON$ gains the profit of $2x$ per machine. Let $\tilde{c}_2$ denote the number of machines from $C$ each of which $ON$ places at least two jobs from $S'_1 \cup S'_2$ onto. Then, $ON$ gains the profit of at most $3x$ per machine. Let $\bar{a}$ denote the number of machines
of machines from $A$ each of which $ON$ places at least one job from $S'_1 \cup S'_2$ onto. Then, $ON$ gains the profit of $2 + x$ per machine. By the above definitions and Eq. (10),

$$\tilde{b} \leq b + b',$$  \hspace{1cm} (16)

$$\tilde{c}_1 + \tilde{c}_2 \leq c = m - a - b - b',$$  \hspace{1cm} (17)

and

$$\tilde{a} \leq 2m'' - \tilde{b} - \tilde{c}_1 - 2\tilde{c}_2.$$  \hspace{1cm} (18)

Thus, we have

$$V_{ON}(\sigma_2) \leq 2(a - \tilde{a}) + b + b' + 2x\tilde{b} + 2x\tilde{c}_1 + 3x\tilde{c}_2 + (2 + x)\tilde{a}$$

$$\leq 2m' + x(\tilde{b} + \tilde{c}_1 + \tilde{c}_2) + 2xm'' \quad \text{(by Eqs. (11), (12), and (18))}$$

$$\leq 2m' + x(m - a) + 2xm'' \quad \text{(by Eqs. (16) and (17))} \hspace{1cm} (19)$$

On the other hand, for every $j = 1, \ldots, m''$, $OPT$ places both $J_{1,j}$ and $J_{2,j}$ onto one machine. Additionally, $OPT$ places both $J'_{1,j}$ and $J'_{2,j}$ onto the remaining machines. Hence,

$$V_{OPT}(\sigma_2) = m + 4xm''.$$  \hspace{1cm} (20)

If $m$ is even, then $m' = m'' = m/2$. By Eqs. (13), (14), and (15),

$$\frac{V_{OPT}(\sigma_1)}{V_{ON}(\sigma_1)} \geq \frac{2m' + (2 + x)m''}{a + m + (1 + x)m''} = \frac{2m + xm/2}{a + 3m/2 + xm/2}.$$  \hspace{1cm} (21)

By Eqs. (19) and (20),

$$\frac{V_{OPT}(\sigma_2)}{V_{ON}(\sigma_2)} \geq \frac{m + 4xm''}{2m' + x(m - a) + 2xm''} = \frac{m + 2xm}{m + (2m - a)x}.$$  \hspace{1cm} (22)

If $m$ is odd, $m' = (m + 1)/2$ and $m'' = (m - 1)/2$. By Eqs. (13), (14), and (15),

$$\frac{V_{OPT}(\sigma_1)}{V_{ON}(\sigma_1)} \geq \frac{2m + x(m - 1)/2}{a + (3m - 1)/2 + x(m - 1)/2}.$$  \hspace{1cm} (23)

and by Eqs. (19) and (20),

$$\frac{V_{OPT}(\sigma_2)}{V_{ON}(\sigma_2)} \geq \frac{m + 2x(m - 1)}{m + 1 + (2m - a - 1)x}.$$  \hspace{1cm} (24)

Therefore,

$$\frac{V_{OPT}(\sigma)}{V_{ON}(\sigma)} \geq \max_a \min_x \left\{ \frac{V_{OPT}(\sigma_1)}{V_{ON}(\sigma_1)} \frac{V_{OPT}(\sigma_2)}{V_{ON}(\sigma_2)} \right\}$$  \hspace{1cm} (25)

and we can have the lower bounds in Table 2 using Eqs. (21),(22),(23),(24) and (25) for each $m$. □
Table 2: Lower bounds for each $m(\geq 3)$.

| $m$ | Lower Bound | $\alpha$ | $x$ |
|-----|-------------|---------|-----|
| 3   | $7/6 \geq 1.166$ | 0       | 3   |
| 4   | $\frac{22-2\sqrt{2}}{17} \geq 1.127$ | 0       | $2 + 2\sqrt{2}$ |
| 5   | $\frac{420-15\sqrt{7}}{333} \geq 1.142$ | 1       | $-1+\sqrt{7}$ |
| 6   | $\frac{51-6\sqrt{7}}{41} \geq 1.140$ | 2       | $\sqrt{2}$ |
| 7   | $\frac{280-70\sqrt{11}}{227} \geq 1.158$ | 1       | $1+\sqrt{11}$ |

$\infty \frac{48-2\sqrt{2}}{41} \geq 1.101 \quad m/4 \quad \sqrt{2}$

6 Conclusions

In this paper, we have proposed a novel variant of the interval scheduling problem focusing on best-effort services. For this variant, we have proved that the competitive ratios of an online greedy algorithm are at most $4/3$ and $3$ for $m = 2$ and $m \geq 3$, respectively. Also, we have shown a lower bound on the competitive ratio of any deterministic algorithm for each $m$. We finish the paper by providing some open questions: (i) In the setting studied in the paper, preemption is not allowed. Then, if preemption is allowed, can we design a competitive algorithm for the general profit case? (ii) Will randomization help to improve our results? (iii) An obvious open problem is to close the gaps between our lower and upper bounds for the uniform profit case. In addition, we should discuss offline algorithms for our variant.

References

[1] E. M. Arkin and E. B. Silverberg, “Scheduling jobs with fixed start and end times,” Discrete Applied Mathematics, Vol. 18, No. 1, pp. 1–8, 1987.

[2] S. K. Baruah and J. R. Haritsa, “Scheduling for overload in real-time systems,” IEEE Transactions on Computers, Vol. 46, No. 9, pp. 1034–1039, 1997.

[3] A. Borodin and R. El-Yaniv, “Online Computation and Competitive Analysis,” Cambridge University Press, Cambridge, 1998.

[4] K. I. Bouzina and H. Emmons, “Interval scheduling on identical machines,” Journal of Global Optimization, Vol. 9, No. 3–4, pp. 379–393, 1996.

[5] R. Canetti and S. Irani, “Bounding the power of preemption in randomized scheduling,” SIAM Journal on Computing, Vol. 27, No. 4, pp. 993–1015, 1998.

[6] M. C. Carlisle and E. L. Lloyd, “On the k-coloring of intervals,” Discrete Applied Mathematics, Vol. 59, No. 3, pp. 225–235, 1995.

[7] B. DasGupta and M. A. Palis, “Online real-time preemptive scheduling of jobs with deadlines on multiple machines,” Journal of Scheduling, Vol. 4, No. 6, pp. 297–312, 2001.

[8] L. Epstein and A. Levin “Improved randomized results for the interval selection problem,” Theoretical Computer Science, Vol. 411, No. 34–36, pp. 3129–3135, 2010.
[9] U. Faigle and W. M. Nawijn, “Note on scheduling intervals on-line,” *Discrete Applied Mathematics*, Vol. 58, No. 1, pp. 13–17, 1995.

[10] S. P. Y. Fung, C. K. Poon and F. Zheng, “Online interval scheduling: Randomized and multiprocessor cases,” *Journal of Combinatorial Optimization*, Vol. 16, No. 3, pp. 248–262, 2008.

[11] S. P. Y. Fung, C. K. Poon and D. K. W. Yung, “On-line scheduling of equal-length intervals on parallel machines,” *Information Processing Letters*, Vol. 112, No. 10, pp. 376–379, 2012.

[12] S. P. Y. Fung, C. K. Poon and F. Zheng, “Improved randomized online scheduling of intervals and jobs,” *Theory of Computing Systems*, Vol. 55, No. 1, pp. 202–228, 2014.

[13] A. W. J. Kolen, J. K. Lenstra, C. H. Papadimitriou and F. C. R. Spieksma, “Interval scheduling: A survey,” *Naval Research Logistics*, Vol. 54, No. 5, pp. 530–543, 2007.

[14] M. Y. Kovalyov, C. T. Ng and T. C. E. Cheng, “Fixed interval scheduling: Models, applications, computational complexity and algorithms,” *European Journal of Operational Research*, Vol. 178, No. 2, pp. 331–342, 2007.

[15] R. J. Lipton and A. Tomkins, “Online interval scheduling,” *In Proc. of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 302–311, 1994.

[16] H. Miyazawa and T. Erlebach, “An improved randomized on-line algorithm for a weighted interval selection problem,” *Journal of Scheduling*, Vol. 7, No. 4, pp. 293–311, 2004.

[17] P. Sankowski and C. Zaroliagis, “The power of migration for online slack scheduling,” *In Proc. of the 24th Annual European Symposium on Algorithms*, pp. 75:1–17, 2016.

[18] S. S. Seiden, “Randomized online interval scheduling,” *Operations Research Letters*, Vol. 22, No. 4–5, pp. 171–177, 1998.

[19] D. D. Sleator, and R. E. Tarjan, “Amortized efficiency of list update and paging rules,” *Communications of the ACM*, Vol. 28, No. 2, pp. 202–208, 1985.

[20] G. J. Woeginger, “On-line scheduling of jobs with fixed start and end times,” *Theoretical Computer Science*, Vol. 130, No. 1, pp. 5–16, 1994.
A Assignment Examples

In all the figures of this section, \(c\)-intervals (\(ge\)-intervals, \(oe\)-intervals and \(n\)-interval, respectively) are shown in blue (green, red and yellow, respectively) squares. Also, only \(GR\)'s jobs and machines are shown. Our assignments are realized as matchings between \(oe\)-intervals of \(OPT\)'s jobs and \(p\)-intervals of \(GR\)'s jobs. However, for ease of presentation, the assignments are presented as matchings between \(oe\)-intervals of “\(GR\)'s jobs” and \(p\)-intervals of \(GR\)'s jobs. For example, in the situation of Fig. 4, the routine assigns the interval \([1, 2]\) of \(GR\)'s \(J_1\) to the interval \([0, 1]\) of \(OPT\)'s \(J_4\). However, the assignment is described as the one from the interval \([1, 2]\) of \(GR\)'s \(J_1\) to the interval \([0, 1]\) of \(GR\)'s \(J_4\) in the figure. In addition, we use situations which cannot happen to explain assignments. For example, in Fig. 4, \(GR\) cannot place \(J_2\) onto the second machine according to its definition but places it onto the first machine.

A.1 General \(m\)

In Fig. 4, at first, the routine assigns the \(c\)-interval \([1, 2]\) of \(GR\)'s \(J_1\), which is placed on the first machine, to the \(oe\)-interval \([0, 1]\) of \(OPT\)'s \(J_4\) in the left figure. However, after \(GR\) places \(J_5\) onto the second machine, the routine reassigns the \(c\)-interval \([1, 2]\) of \(GR\)'s \(J_5\) to the \(oe\)-interval \([0, 1]\) of \(OPT\)'s \(J_4\), and assigns the \(c\)-interval \([1, 2]\) of \(GR\)'s \(J_1\) to the \(oe\)-interval \([0, 1]\) of \(OPT\)'s \(J_5\) in the right figure. Of course, it is possible that the routine does not change the assignment of \(J_4\) and assigns the \(c\)-interval \([1, 2]\) of \(J_5\) to the \(oe\)-interval \([0, 1]\) of \(J_5\).

![Figure 4: Assignment example 1 for general \(m\).](image_url)

In Fig. 5, the routine assigns the \(c\)-interval \([0, 1]\) of \(J_1\) to the \(oe\)-interval \([0, 1]\) of \(J_4\). Since the \(n\)-interval \([2, 3]\) of \(J_4\) exists, the routine assigns the \(c\)-interval \([3, 4]\) of \(J_2\) to the \(oe\)-interval \([1, 2]\) of \(J_3\).

In Fig. 6, the routine executes Case 2.1 assigning the \(c\)-interval \([1, 2]\) of \(J_1\) to the \(oe\)-interval \([0, 1]\) of \(J_3\), and assigning the \(c\)-interval \([1, 2]\) of \(J_3\) to the \(oe\)-interval \([1, 2]\) of \(J_4\). Then, the routine executes Case 2.2 and assigns the \(c\)-interval \([2, 3]\) of \(J_6\) to the \(oe\)-interval \([1, 2]\) of \(J_5\).
A.2 $m = 2$

In Fig. 7, the c-interval $[1, 2]$ of $J_2$ contains the p-fraction $f'$ called in the routine. Also, the c-interval $[2, 3]$ of $J_3$ and the ge-interval $[2, 3]$ of $J_4$ contain the p-fraction $f_1$ and the p-fraction $f_2$, respectively, called in the routine. The routine executes Case 2.2 and assigns the c-interval $[1, 2]$ of $J_2$, the c-interval $[2, 3]$ of $J_3$ and the ge-interval $[2, 3]$ of $J_4$ to the oe-interval $[1, 2]$ of $J_4$. Then, the routine executes Case 2.1 and assigns the ge-interval $[2, 3]$ of $J_4$ to the oe-interval $[2, 3]$ of $J_5$. 

Figure 5: Assignment example 2 for general $m$. 
Figure 6: Assignment example 3 for general $m$.

Figure 7: Assignment example for $m = 2$. 