On-line scheduling with delivery time on a single batch machine

Ji Tian, Ruyan Fu, Jinjiang Yuan*

Department of Mathematics, Zhengzhou University, Zhengzhou, Henan 450052, People's Republic of China

Received 19 September 2006; received in revised form 20 November 2006; accepted 7 December 2006

Communicated by D.-Z. Du

Abstract

We consider a single batch machine on-line scheduling problem with jobs arriving over time. A batch processing machine can handle up to $B$ jobs simultaneously as a batch, and the processing time for a batch is equal to the longest processing time among the jobs in it. Each job becomes available at its arrival time, which is not known in advance, and its characteristics, such as processing time and delivery time, become known at its arrival. Once the processing of a job is completed we deliver it to the destination. The objective is to minimize the time by which all jobs have been delivered. In this paper, we deal with two variants: the unbounded model where $B$ is sufficiently large and the bounded model where $B$ is finite. We provide on-line algorithms with competitive ratio 2 for the unbounded model and with competitive ratio 3 for the bounded model. For when each job has the same processing time, we provide on-line algorithms with competitive ratios $(\sqrt{5}+1)/2$, and these results are the best possible.

Keywords: Single machine; Parallel batching; On-line algorithm; Competitive ratio; Delivery time

1. Introduction

Recently, on-line scheduling and parallel batch scheduling have been two flourishing scheduling models. There has been much work on these scheduling models. Here, on-line scheduling means that jobs arrive over time, and all job characteristics become known at their arrival times. Parallel batch scheduling means that a machine can process several jobs simultaneously as a batch, and the processing time of a batch is equal to the longest processing time of the jobs assigned to it. Once a batch is started, it cannot be stopped until its completion. The scheduling model of a batch processing machine is motivated by burn-in ovens in the final testing stage of semiconductor manufacturing (Lee et al. [8], Uzsoy et al. [12,13]).

In this paper, we consider an on-line scheduling model: on-line scheduling with delivery time on a single batch machine. Here, we have a batch machine and sufficiently many vehicles. There are $n$ jobs $J_1, J_2, \ldots, J_n$. Each job has a release time, a processing time, and a delivery time. These characteristics are known about a job until it arrives. The objective is to minimize the time by which all jobs have been delivered. In this model, each job needs to be processed on the machine, and once the job is completed we deliver it to the destination by a vehicle.

* Research Supported by NNSFC(10671183).
* Corresponding author. Tel.: +86 371 67767835.
E-mail address: yuanjj@zzu.edu.cn (J. Yuan).
Throughout the paper we use $J_j$ to denote a job indexed by $j$, and $r_j, p_j, q_j$ to denote the release time, the processing time, and the delivery time of $J_j$, respectively. Let $\sigma$ be a schedule of the jobs. We denote by $S_j(\sigma), C_j(\sigma)$ and $L_j(\sigma)$, respectively, the starting time of $J_j$, the completion time of $J_j$ and the time by which $J_j$ is delivered in schedule $\sigma$. Using the 3-field notation of Graham et al. [4], the problem is denoted as $1|on-line, r_j, q_j, B|L_{\text{max}}$, where $L_{\text{max}} = \max\{L_j : L_j = C_j + q_j, \ 1 \leq j \leq n\}$.

The quality of an on-line algorithm is measured by the competitive ratio. Let $L_{\text{max}}(\sigma)$ and $L_{\text{max}}(\pi)$ denote the times by which all jobs have been delivered given by an on-line algorithm and an optimal algorithm, respectively, for an input job list $L$. The competitive ratio $R$ of the on-line algorithm is defined as

$$R = \sup_{\forall L} \{L_{\text{max}}(\sigma)/L_{\text{max}}(\pi)\}.$$ 

There have been many results concerning the scheduling problem in which the objective is to minimize the time by which all jobs have been delivered. In 1993, Lawer et al. [7] proved that the off-line problem $1|r_j, q_j|L_{\text{max}}$ is strongly NP-hard, but the preemptive version can be solved in polynomial time. If all jobs are released at the same time, there is a well known algorithm LDT (every time, choose from among available jobs the one with the largest delivery time) for this problem. For the case with unequal release times, Kise et al. [8] proved that the LDT rule has a competitive ratio of 2 in 1979. In 2000, Hoogeveen et al. [5] considered the on-line problem $1|on-line, r_j, q_j|L_{\text{max}}$. They provided an on-line algorithm with competitive ratio $(\sqrt{5} + 1)/2$, and showed that it is the best possible. The above results are all for the non-batch machine model, and we will discuss the batch machine model in this paper.

If we assume that all the jobs have the same delivery time, then the scheduling problem considered will degenerate to the scheduling problem $1|on-line, r_j, B|C_{\text{max}}$. For this problem, there has been a great deal of study. In the off-line scheduling model, if all jobs have the same release time, the optimal schedule can be found by the FBLPT rule (every time, give priority to the job with the longest processing time to form a batch) of Bartholdi [1] (Brucker et al. [2], Lee and Uzsoy [9]). The FBLPT rule takes an important role in the batch scheduling problem. If all jobs have different release times, for when the capacity of the batch machine is sufficiently large ($B = \infty$), Lee and Uzsoy [9] presented a dynamic programming algorithm with running time $O(n^2)$; for when the capacity of the batch machine is finite ($B < n$), Brucker et al. [2] and Liu et al. [10] showed that the problem is NP-hard. For the on-line scheduling model, Deng et al. [3] and Zhang et al. [15] have studied the parallel batch scheduling problem. They proved that there is no on-line algorithm with competitive ratio smaller than $(\sqrt{5} + 1)/2$, and for the case $B = \infty$, they independently gave the same on-line algorithm with competitive ratio matching the lower bound. For the case $B < n$, the first on-line algorithm is a greedy heuristic, GRLPT, of Lee and Uzsoy [9] which was shown to be 2-competitive by Liu and Yu [10]. Zhang et al. [15] presented two on-line algorithms with competitive ratio not greater than 2. The above three on-line algorithms are all based on the ideas of the FBLPT rule. Later, Poon and Yu [11] presented a class of algorithms called the FBLPT-based algorithms that contains all the above three algorithms as special cases, and showed that any FBLPT-based algorithm has competitive ratio at most 2. In particular, for the case $B = 2$, they also gave an on-line algorithm with competitive ratio 7/4.

Adding delivery time to jobs, we get the model $1|on-line, r_j, q_j, B|L_{\text{max}}$ studied in this paper. We provide an on-line algorithm with competitive ratio 2 for the unbounded model and an on-line algorithm with competitive ratio 3 for the bounded model. For when each job has the same processing time, we provide on-line algorithms with competitive ratios $(\sqrt{5} + 1)/2$, and these results match the lower bound provided by the Zhang et al. [15].

2. Preliminaries

First of all we consider a special case where all jobs are available simultaneously, which is denoted by $1|q_j, B = \infty|L_{\text{max}}$. We may assume that the job list $\tau = \{J_1, \ldots, J_n\}$ satisfies

$$p_1 < p_2 < \cdots < p_n, \quad \text{and} \quad q_1 > q_2 > \cdots > q_n.$$ 

In fact, if there exist two jobs $J_i$ and $J_j$ such that $p_i \leq p_j$ and $q_i \leq q_j$, this can be regarded as $J_i$ being absorbed by $J_j$, i.e., the job $J_i$ could be ignored. Since the batch capacity is infinite, obviously, the optimal value of the objective function will not change. Hence, we can recursively do the above operation such that, for each pair of jobs $J_i$ and $J_j$, if $p_i < p_j$, then $q_i > q_j$. 

In the above assumption, we know that all jobs are indexed according to the SPT rule, i.e., \( p_1 < p_2 < \cdots < p_n \). An SPT-batch schedule is one in which adjacent jobs in the sequence \((J_1, \ldots, J_n)\) may be grouped to form batches. For example, a possible batch schedule for a 10-job problem is the sequence of four batches specified by \((J_1, J_2, J_3), (J_4), (J_5, J_6, J_7, J_8), (J_9, J_{10})\). For problem 1\(|q_j, B = \infty|L_{\text{max}}, since the jobs are also indexed according to the LDT rule in our assumption, i.e., \( q_1 > q_2 > \cdots > q_n \), like in the work by Brucker et al. [2], there exists an optimal solution that is an SPT-batch schedule.

In the following we give a forward dynamic programming algorithm implied in [2]:

**Algorithm A.** Let \( R(k) \) be the minimum time by which all jobs have been delivered among all SPT-batch schedules containing jobs \( J_1, \ldots, J_k \). Let \( \pi(k) \) denote the set of optimal SPT-batch schedules containing the first \( k \) jobs, and let \( C(k) \) be the minimum makespan among all schedules in \( \pi(k) \), i.e.,

\[
C(k) = \min_{\pi \in \pi(k)} \{ C_{\text{max}}(\pi) : L_{\text{max}}(\pi) = R_k \}.
\]

Let

\[
\tau(k) = \{ j : R(k) = \max\{R(j), C(j) + p_k + q_{j+1}\} \};
\]

then we have

\[
C(k) = \min_{j \in \tau(k)} \{C(j) + p_k\}.
\]

The initializations are

\[
\tau(0) = \emptyset, \quad C(0) = 0, \quad R(0) = 0,
\]

and the recursion for \( k = 1, \ldots, n \) is

\[
R(k) = \min_{0 \leq j \leq k-1} \max\{R(j), C(j) + p_k + q_{j+1}\}.
\]

The optimal solution value is equal to \( R(n) \). The algorithm requires \( O(n^3) \) time.

In Section 4, for convenience of application of the above algorithm, the notation \( R(k) \) and \( C(k) \) will be rewritten as \( R(\{J_1, \ldots, J_k\}) \) and \( C(\{J_1, \ldots, J_k\}) \), respectively.

If \( B < n \) and all jobs have the same processing times, we offer a simple algorithm, like the well known algorithm FBLPT, denoted by FBLDT (Full Batch Largest Delivery Time).

**Algorithm (FBLDT).** Step 1: Index the jobs such that \( q_1 \geq q_2 \geq \cdots \geq q_n \).

Step 2: Form batches by placing jobs \( iB + 1 \) through \((i+1)B\) together in a batch for \( i = 0, \ldots, \lfloor n/B \rfloor \), where \( \lfloor x \rfloor \) represents the largest integer smaller than \( x \).

Step 3: Schedule the batches accordingly.

**Theorem 1.** For problem 1\(|q_j, p_j = p, B < n|L_{\text{max}}, the FBLDT algorithm is optimal.

**Proof.** Let \( \tau = \{J_1, \ldots, J_n\} \) be the job set such that \( q_1 \geq q_2 \geq \cdots \geq q_n \). Let \( \pi \) be an optimal schedule for it. Suppose that \( \pi \) consists of \( k \) batches, denoted by \( B_1, \ldots, B_k \). If there exist two jobs \( J_i \in B_x \) and \( J_j \in B_y \) such that \( i > j \) and \( 1 \leq x < y \leq k \). Then we exchange \( J_i \) and \( J_j \). The resulting new schedule is denoted by \( \pi' \). Assume that the batch \( B_x \) starts at time \( t \) in schedule \( \pi \). Then it follows that \( \pi' \) is also an optimal schedule. We have

\[
L_i(\pi) = t + p + q_i; \quad L_j(\pi) = t + (y - x + 1)p + q_j.
\]

On the other hand, in schedule \( \pi' \), we have

\[
L_j(\pi') = t + p + q_j; \quad L_i(\pi') = t + (y - x + 1)p + q_i.
\]

Since \( q_i \leq q_j \), we further have

\[
L_i(\pi') \leq L_j(\pi') \leq L_{\text{max}}(\pi), \quad L_j(\pi) \leq L_j(\pi) \leq L_{\text{max}}(\pi).
\]
A Step 0: For any instance \( \mathcal{I} \), \( \sigma \) be the schedule produced by Algorithm A, the greedy algorithm, and \( \pi \) be the schedule produced by Algorithm H. \( \pi \) consists of full batches except the last batch, then it is just the schedule obtained by FBLPT.

If there exists a non-full batch that is not the last batch in \( \pi \), denote the first non-full batch by \( B_1 \); then we remove the jobs from the bottom of batch \( B_{i+1} \) and put them into batch \( B_i \) until it becomes a full batch. We do this repeatedly until there no longer exists any non-full batch except the last batch in the final schedule. After doing this we can observe that the objective value of this problem does not increase and the final schedule is just the schedule obtained by FBLPT. \( \square \)

3. A lower bound

To find a lower bound of all on-line algorithms for the problem in this paper, we consider the scheduling model \( 1|\text{on-line}, r_j, B|C_{\text{max}} \). Note that this model is a special case of the scheduling problem studied in this paper. It is clear that a lower bound of the former is also a lower bound of the latter. For the former problem, Zhang [15] presented a lower bound of all on-line algorithms. Here, we quote it in the following.

**Lemma 2** (Zhang [15]). There does not exist any on-line algorithm with competitive ratio less than \( 1 + \alpha \) for the scheduling problem: \( 1|\text{on-line}, r_j, B|C_{\text{max}} \), where \( \alpha = (\sqrt{5} - 1)/2 \).

According to the proof of the above Lemma 2 in [15], the lower bound is applicable to both the unbounded case and the bounded case (even though all jobs have the same processing time in both cases).

**Corollary 3.** There does not exist any on-line algorithm with competitive ratio less than \( 1 + \alpha \) for the scheduling problem: \( 1|\text{on-line}, r_j, q_j, B|L_{\text{max}} \), even though the jobs have the same processing times in the two cases.

4. The unbounded case

In this section, we deal with the unbounded case where the capacity \( B \) is sufficiently large, i.e., all jobs can be processed simultaneously in a single batch. Let \( U(t) \) denote the set of unscheduled jobs available at time \( t \).

**Algorithm** \( (H_1^\infty) \). **Step 0:** Set \( t = 0 \).

**Step 1:** If the machine is idle at time \( t \) and \( U(t) = \emptyset \), go to step 4; otherwise, apply Algorithm A to \( U(t) \) to obtain a schedule \( \pi(t) \) for \( U(t) \) and compute \( R(U(t)) \) and \( C(U(t)) \). Set \( s = \max\{t, R(U(t))\} \).

**Step 2:** At time interval \([t, s)\), whenever a new job \( J_h \) comes in, say, at time \( t' \), let \( U(t') = U(t) \cup J_h \). Reset \( t := t' \), back to step 1.

**Step 3:** At time \( s \), schedule all the batches produced by the dynamic algorithm on \( U(s) \). Let \( t = s + C(U(s)) \). Go to step 1.

**Step 4:** If there are still some jobs arriving, set \( t \) as the arrival time of the next job and back to step 1; otherwise, stop and complete the schedule at time \( t \).

**Remark.** In algorithm \( H_1^\infty \), at time \( s \), once we decide to schedule the batches produced by Algorithm A, we need to consecutively schedule all of them. If we name these batches as a block at each time \( s \), then the schedule produced by \( H_1^\infty \) consists of several blocks. Furthermore, when we apply Algorithm A to \( U(t) \), the release times of all jobs in \( U(t) \) are reset to being 0.

**Theorem 4.** For the problem \( 1|\text{on-line}, r_j, q_j, B = \infty|L_{\text{max}} \), the competitive ratio of algorithm \( H_1^\infty \) is exactly 2.

**Proof.** For any instance \( \mathcal{I} \), let \( \sigma \) and \( \pi \) be the schedule produced by algorithm \( H_1^\infty \) and an optimal schedule for \( \mathcal{I} \), respectively. Let \( J_I \) denote the first job in \( \sigma \) that assumes the value \( L_{\text{max}}(\sigma) \). Assume that the block containing job \( J_I \) in \( \sigma \) is \( D_I \). Denote the block completed before block \( D_I \) by \( D^* \) in \( \sigma \) (if there exists a such block). We assume that the starting times of the blocks \( D_I \) and \( D^* \) are \( t \) and \( t^* \) (if \( D^* \) does not exist, set \( t^* = 0 \), respectively.

If \( D^* \) does not exist, by \( H_1^\infty \), we have \( t = R(U(t)) \), and

\[
L_{\text{max}}(\sigma) = t + R(U(t)) = 2R(U(t)).
\]
On the other hand, we have
\[ L_{\text{max}}(\pi) \geq R(U(t)). \]

Hence, we have
\[ \frac{L_{\text{max}}(\sigma)}{L_{\text{max}}(\pi)} \leq 2. \]

We suppose in the following that \( D^* \) exists. We consider two cases accordingly.

**Case 1:** The machine has an idle time between \( D^* \) and \( D_l \). We denote by \( r \) the minimum release time of jobs scheduled in \( D_l \). By algorithm \( H_{1\infty}^\infty \), we have \( t^* < r \leq t \). According to the value of \( r \), we have two cases as follows.

**Case 1.1:** \( t^* < r < t \). By algorithm \( H_{1\infty}^\infty \), we have \( t^* = R(U(t)) \). By a proof like that of case 1, the theorem holds.

**Case 1.2:** \( r = t \). By algorithm \( H_{1\infty}^\infty \), we have \( t^* \geq R(U(t)) \). If \( t = R(U(t)) \), the case is reduced to case 1.1. Suppose \( t > R(U(t)) \); then we have
\[ L_{\text{max}}(\sigma) = t^* + R(U(t)) \]
and
\[ L_{\text{max}}(\pi) \geq r + R(U(t)) = t + R(U(t)). \]

Hence,
\[ L_{\text{max}}(\sigma) = L_{\text{max}}(\pi). \]

**Case 2:** The machine has no idle time between \( D^* \) and \( D_l \). According to the algorithm \( H_{1\infty}^\infty \), we can observe that \( t^* \geq R(U(t^*)) \geq C(U(t^*)), \)

and
\[ L_{\text{max}}(\sigma) = t^* + C(U(t^*)) + R(U(t)). \]

On the other hand, we have
\[ L_{\text{max}}(\pi) \geq t^* + R(U(t)). \]

Thus we have
\[ \frac{L_{\text{max}}(\sigma)}{L_{\text{max}}(\pi)} \leq \frac{t^* + C(U(t^*)) + R(U(t))}{t^* + R(U(t))} = 1 + \frac{C(U(t^*))}{t^* + R(U(t))} \leq 2. \]

When all jobs arrive at time 0, according to algorithm \( H_{1\infty}^\infty \), we have \( L_{\text{max}}(\sigma) = 2R(U(0)) \). On the other hand, we have \( L_{\text{max}}(\sigma) = R(U(0)) \). Hence, the competitive ratio of algorithm \( H_{1\infty}^\infty \) is exactly 2. This completes the proof. \( \square \)

From above discussions, we get an on-line algorithm \( H_{1\infty}^\infty \) with competitive ratio 2 for the general version. If all jobs have the same processing times, we can find a best possible on-line algorithm, i.e., which has a competitive ratio \( 1 + \alpha \). Here, we quote the algorithm provided by Yuan et al. [14] for \( 1|\text{on-line}, r_j, p_j = p, \text{prec}, q_j, B = \infty|C_{\text{max}} \).

**Algorithm \( (H_{2\infty})^\infty \). Step 0:** Set \( k = 0 \).

**Step 1:** At time \( \alpha p + kp \), form the \((k+1)\)-batch \( B_{k+1} \) by including all the unprocessed available jobs (here, we allow \( B_{k+1} = \emptyset \)).

**Step 2:** Set \( k := k + 1 \) and go to step 2.

**Theorem 5.** For the problem \( 1|\text{on-line}, r_j, p_j = p, q_j, B = \infty|L_{\text{max}} \), the competitive ratio of algorithm \( H_{2\infty}^\infty \) is not greater than \( 1 + \alpha \).
Consider an instance $I$, let $\sigma$ be the schedule produced by algorithm $H_2^\infty$ for $I$, and let $\pi$ be an off-line optimal schedule for $I$. Assume that $\sigma$ consists of $n$ batches $B_1, \ldots, B_n$ and $\pi$ consists of $m$ batches $B_1^*, \ldots, B_m^*$. Let the starting time of batch $B_i$ in $\sigma$ be $S_i$, $1 \leq i \leq n$, and let the starting time of batch $B_j^*$ in $\pi$ be $S_j^*$, $1 \leq j \leq m$. Then we have $S_i = (\alpha + x - 1)p$, for $1 \leq i \leq n$. For any job $J_j$, if $J_j \in B_i^*$, we have $L_j(\pi) = S_i^* + p + q_j$; and if $J_j \in B_1^*$, we have $L_j(\pi) = S_j + p + q_j = (\alpha + y)p + q_j$. If we define $S_0 = (\alpha - 1)p$, then $S_i = S_{i-1} + p = (\alpha + i - 1)p$, for $1 \leq i \leq n$.

First of all we have two claims in the following.

**Claim 1.** For any job $J_j$, we have $L_j(\pi) < L_j(\pi) + p$.

Suppose, to the contrary, that there exists a job $J_j$ such that $L_j(\pi) \geq L_j(\pi) + p$. Suppose further that $J_j \in B_1^*$ and $J_j \in B_2^*$, where $1 \leq x \leq m$ and $1 \leq y \leq n$. We choose a job $J_j$ such that index $y$ is as small as possible.

If $x \geq y$, then $L_j(\pi) \geq xp + q_j \geq yp + q_j$, and $L_j(\sigma) = (\alpha + y)p + q_j$. Thus we have $L_j(\sigma) < L_j(\pi) + p$. This is contrary to the choice of the job $J_j$. Hence, we have $x < y$.

According to the algorithm $H_2^\infty$, $r_j \geq S_{y-1} = (\alpha + y - 2)p$. Since $L_j(\sigma) = (\alpha + y)p + q_j \geq L_j(\pi) + p = S_1^* + 2p + q_j$, we have $S_1^* \leq (\alpha + y - 2)p < r_j$. This is contrary to the fact that job $J_j$ is scheduled in batch $B_1^*$ in $\pi$. Claim 1 follows.

**Claim 2.** For any job $J_j$, we have $L_j(\sigma) \leq (1 + \alpha)L_j(\pi)$.

If $L_j(\pi) \geq (1 + \alpha)p$, according to Claim 1, we have

$$\frac{L_j(\sigma)}{L_j(\pi)} < \frac{L_j(\pi) + p}{L_j(\pi)} = 1 + \frac{p}{L_j(\pi)} \leq 1 + \alpha.$$

Suppose $L_j(\pi) < (1 + \alpha)p$. Then $J_j \in B_1^*$ and $r_j < \alpha p$. By algorithm $H_2^\infty$, this implies that $J_j$ is scheduled in batch $B_1$. Hence, we have $L_j(\sigma) = (1 + \alpha)p + q_j$. On the other hand, we have $L_j(\pi) \geq p + q_j$. Therefore, we get

$$\frac{L_j(\sigma)}{L_j(\pi)} < \frac{(1 + \alpha)p + q_j}{p + q_j} \leq 1 + \alpha.$$

Let $J_l$ denote the first job in $\sigma$ that assumes the value $L_{\max}(\sigma)$. Then $L_{\max}(\sigma) = L_l \leq (1 + \alpha)L_l(\pi)$. Since $L_{\max}(\pi) \geq L_l(\pi)$, we have

$$\frac{L_{\max}(\sigma)}{L_{\max}(\pi)} \leq \frac{L_l(\sigma)}{L_l(\pi)} \leq 1 + \alpha.$$

This completes the proof. □

**Corollary 3** and **Theorem 5** imply that $H_2^\infty$ is a best possible on-line algorithm for the problem $1|\text{on-line}, r_j, p_j = p, q_j, B = \infty|L_{\max}$.

**5. The bounded case**

In this section, we deal with the bounded case where the capacity $B$ is finite.

**Lemma 6** (Poon and Yu [11]). For the scheduling model $1|\text{on-line}, r_j, p_j = p, q_j, B < n|C_{\max}$, any FBLPT-based algorithm has competitive ratio not greater than 2.

According to Lemma 6, we can obtain the following **Theorem 7**.

**Theorem 7.** For the scheduling model $1|\text{on-line}, r_j, p_j, q_j, B < n|L_{\max}$, any FBLPT-based algorithm has competitive ratio not greater than 3.

**Proof.** Given an instance $I$, let $\sigma$ be the scheduling produced by an FBLPT-based algorithm for $I$. Let $\pi_1$ be an optimal schedule for $I$. If we ignore the delivery time of each job, we can obtain another optimal schedule, which is denoted by $\pi_2$. Let $J_1$ denote the first job in $\sigma$ that assumes the value $L_{\max}$. According to Lemma 6, we have

$$C_{\max}(\sigma) \leq 2C_{\max}(\pi_2).$$
Since
\[ C_{\text{max}}(\pi_2) \leq C_{\text{max}}(\pi_1) \leq L_{\text{max}}(\pi_1), \]
\[ q_l < L_{\text{max}}(\pi_1). \]

Hence, we get
\[ L_{\text{max}}(\sigma) \leq C_{\text{max}}(\sigma) + q_l \]
\[ \leq 2C_{\text{max}}(\pi_2) + q_l \]
\[ \leq 3L_{\text{max}}(\pi_1). \]

This completes the proof. \( \square \)

If all jobs have the same processing times, we can give a best possible on-line algorithm. Without loss of generality, we assume that the first arrival time of jobs is 0. Let \( U(t) \) denote the set of unscheduled jobs available at time \( t \). Before offering the algorithm, we give two definitions:

Given a set of jobs, among all the jobs with the largest delivery time, we arbitrarily pick one and call it the largest job.

Given a set of batches, among all the batches with a largest job in it, we arbitrarily pick one and call it the largest batch.

Algorithm \( (H_B) \).

**Step 0:** Set \( t = 0, U(0) = \{ J_j : r_j = 0 \} \).

**Step 1:** Apply FBLDT to the job set \( U(t) \) resulting in \( M(t) \) batches. If \( |U(t)| < B \), go to step 3.

**Step 2:** Choose a largest batch from \( U(t) \), and schedule it at time \( t \). Reset \( t := t + p \). Then back to step 1.

**Step 3:** If \( |U(t)| = 0 \), go to step 5. Set \( q_k = \max\{q_j : J_j \in U(t)\} \). Let \( \alpha_k = (1 + \alpha)r_k + \alpha p + \alpha q_k \). If \( t \geq \alpha_k \), start the batch at time \( t \). Reset \( t := t + p \); back to step 1.

**Step 4:** Wait for the next job for time \( (1 + \alpha)r_k + \alpha p + \alpha q_k \). If some job arrives in this period, reset \( t \) as the new arrival time; otherwise, reset \( t \) as \( (1 + \alpha)r_k + \alpha p + \alpha q_k \). Back to step 1.

**Step 5:** If there are still some jobs arriving, set \( t \) as the arrival time of the next job and go back to step 1; otherwise, stop and complete the schedule at time \( t \).

**Remark.** In schedule \( \sigma \) is produced by algorithm \( H_B \), a full batch starts as soon as possible; and a non-full batch starts at time \( (1 + \alpha)r_k + \alpha p + \alpha q_k \), where \( r_k, q_k \) denote, respectively, the release time and the delivery time of the largest job \( J_k \) in \( U(t) \).

**Theorem 8.** For the problem \( 1|\text{on-line}, r_j, p_j = p, q_j, B < \infty|L_{\text{max}}, \) the competitive ratio of algorithm \( H_B \) is not greater than \( 1 + \alpha \).

**Proof.** Let \( \sigma \) and \( \pi \) be the schedules generated by algorithm \( H_B \) and an optimal algorithm for a given job list \( L \), respectively. Denote the starting time of the \( i \)th batch in schedule \( \sigma \) by \( s_{(i)} \). Let \( J_l \) denote the first job in \( \sigma \) that assumes the value \( L_{\text{max}}(\sigma) \), and \( B_l \) be the batch which contains \( J_l \). Let \( t \) be the minimum time such that there are no idle times between \( t \) and \( s_{(l)} \) in \( \sigma \). If the machine is busy all the time, then set \( t = 0 \). We assume that there exist \( m \) batches between \( t \) and \( s_{(j)} \) in \( \sigma \). We index them in non-decreasing order of their completion times; clearly, \( s_{(j)} = t + (j - 1)p \), where \( 1 \leq j \leq m \). It can be observed that \( L_{\text{max}}(\sigma) = t + (m + 1)p + q_l \).

First suppose that \( m = 0 \). If \( r_l = s_{(l)} \), then \( B_l \) is a full batch, Hence, we have \( L_{\text{max}}(\sigma) = (s_{(l)} + p + q_l) = (r_l + p + q_l) \leq (L_{\text{max}}(\pi)) \). If \( r_l < s_{(l)} \), then \( s_{(l)} \leq (1 + \alpha)r_l + \alpha p + \alpha q_l \). So, we have \( L_{\text{max}}(\sigma) \leq (1 + \alpha)(r_l + p + q_l) \leq (1 + \alpha)(L_{\text{max}}(\pi)) \). Hence, the result holds when \( m = 0 \).

In the following, we assume that \( m \geq 1 \). If \( s_{(l)} \leq (1 + \alpha)r_l + \alpha p + \alpha q_l \), then clearly, \( L_{\text{max}}(\sigma) \leq (1 + \alpha)(L_{\text{max}}(\pi)) \) holds. So we just need to consider the case in which \( s_{(l)} > (1 + \alpha)r_l + \alpha p + \alpha q_l \). First of all we have the following claim.

**Claim.** \( L_{\text{max}}(\pi) \geq t + mp + q_l \).

If \( r_l \geq t \), we assume that \( B_j \) is being processed when \( J_l \) arrives, i.e., \( r_l \in [t + (j - 1)p, t + jp] \), where \( 1 \leq j \leq m \).
Case 1: $B_j$ is a non-full batch. According to the algorithm $H^B$, the jobs scheduled in batches $B_{j+1}, \ldots, B_m$ arrive not before $s_{(j)}$, and each of them has a delivery time not smaller than $q_l$. Hence, we have

$$L_{\max}(\pi) \geq s_{(j)} + (m - j)p + p + q_l = t + mp + q_l.$$  

Case 2: $B_j$ is a full batch. Let $B_k$ be the last batch such that there exists a job $J_k \in B_k$ with $q_k < q_l$, where $1 \leq k \leq m$.

If $B_k$ exists, we consider the following two subcases.

Case 2.1: There exist some non-full batches between $B_k$ and $B_j$. We denote the last such non-full batch by $B_n$, where $k < n < j$. By the algorithm $H^B$, the jobs scheduled in batches $B_{n+1}, \ldots, B_m$ arrive not before $s_{(n)}$, and each such job has a delivery time not smaller than $q_l$. So we have

$$L_{\max}(\pi) \geq s_{(n)} + (m - n)p + p + q_l = t + mp + q_l.$$  

Case 2.2: The batches between $B_k$ and $B_j$ are all full batches. According to the algorithm $H^B$, the jobs scheduled in batches $B_{k+1}, \ldots, B_m$ arrive not before $s_{(k)}$, and each such job has a delivery time of at least $q_l$. So we have

$$L_{\max}(\pi) \geq s_{(k)} + (m - k)p + p + q_l = t + mp + q_l.$$  

If $B_k$ does not exist, then each job in $B_1, \ldots, B_m$ has a delivery time not smaller than $q_l$. We consider the following two cases.

Case 2.3: $B_1, \ldots, B_m$ are all full batches. Then there are at most $B - 1$ jobs that arrive before time $t$. So we have

$$L_{\max}(\pi) \geq t + (m - 1)p + p + q_l = t + mp + q_l.$$  

Case 2.4: There exist non-full batches between $t$ and $B_j$. Denote the last such non-full batch by $B_x$, where $1 \leq x \leq j$. Clearly, each job in batches $B_{x+1}, \ldots, B_m$ cannot arrive before $s_{(x)}$. So we have

$$L_{\max}(\pi) \geq s_{(x)} + (m - x)p + p + q_l = t + mp + q_l.$$  

Consequently, the claim holds when $r_l \geq t$.

If $r_l < t$, according to the algorithm $H^B$, there does not exist any non-full batch among these $m$ batches, each job in these $m$ batches has a delivery time at least $q_l$, and there are at most $B - 1$ such jobs arriving before time $t$. So we have

$$L_{\max}(\pi) \geq t + (m - 1)p + p + q_l.$$  

The claim holds.

Now we continue the proof of Theorem 8:

If $m \geq 2$, by the above claim we have

$$\frac{L_{\max}(\sigma)}{L_{\max}(\pi)} \leq \frac{t + (m + 1)p + q_l}{t + mp + q_l} \leq \frac{m + 1}{m} \leq \frac{3}{2}.$$  

If $m = 1$ and $r_l \geq t$, then $r_l \in [t, t + p)$. We consider the following two cases:

Case a: $B_1$ is a full batch. By the above claim, we have

$$L_{\max}(\pi) \geq \max\{t + p + q_l, 2p\}.$$  

If $t + p_l \geq \alpha p$, by the above claim, we have

$$\frac{L_{\max}(\sigma)}{L_{\max}(\pi)} \leq \frac{t + 2p + q_l}{t + p + q_l} \leq 1 + \frac{p}{t + p + q_l} \leq 1 + \alpha.$$  

Otherwise, $t + p_l < \alpha p$. Then we have

$$\frac{L_{\max}(\sigma)}{L_{\max}(\pi)} \leq \frac{t + 2p + q_l}{2p} \leq \frac{2 + \alpha}{2} < 1 + \alpha.$$  

Case b: $B_1$ is a non-full batch. By the algorithm $H^B$, $t = s_{(1)} = (1 + \alpha)r_{(1)} + \alpha p + \alpha q_{(1)}$, where $r_{(1)}$ and $q_{(1)}$ denote the arrival time and the delivery time of a job with largest delivery time in $B_1$, respectively. Note that $r_l \geq t$; we have $r_l \geq \alpha p$. According to the assumption $s_l > (1 + \alpha)r_l + \alpha p + \alpha q_l$, we have

$$r_l + p \geq s_l > (1 + \alpha)r_l + \alpha p + \alpha q_l,$$

i.e., $r_l + p_l < \alpha p$, a contradiction. So this case could not occur.
If \( m = 1 \) and \( r_1 < t \), according to the algorithm \( H^B \), \( B_1 \) is a full batch where each job in \( B_1 \) has a delivery time not smaller than \( J_1 \), and there must exist one such job arriving not before \( t \). By the above claim, we have \( L_{\text{max}}(\pi) \geq \max\{t + p + q_1, 2p\} \). Like for case a, we have \( L_{\text{max}}(\sigma) \leq (1 + \alpha)L_{\text{max}}(\pi) \). This completes the proof of Theorem 8. □

Corollary 3 and Theorem 8 imply that \( H^B \) is a best possible on-line algorithm for the problem \( 1|\text{on-line}, r_j, p_j = p, q_j, B < \infty|L_{\text{max}} \).

6. Conclusions

In this paper we consider an on-line scheduling model with delivery time on a single batch machine. If all jobs have the same processing time we find a best possible on-line algorithm for a problem in which the capacity is unbounded or bounded. In the general case, the problem becomes difficult. We provide an on-line algorithm with competitive ratio 2 when the capacity is unbounded and an on-line algorithm with competitive ratio 3 when the capacity is bounded. However, the lower bound for both cases is \( 1 + \alpha \approx 1.618 \). For further research, it is still expected that better on-line algorithms will be looked for, for this problem.

References