https://doi.org/10.1016/j.ejor.2018.07.040

A further study on two-agent parallel-batch scheduling with release dates and deteriorating jobs to minimize the makespan

Yuan Gao[a], **Jinjiang Yuan**[a,*], **C.T. Ng**[b], **T.C.E. Cheng**[b]

[a]School of Mathematics and Statistics, Zhengzhou University,

Zhengzhou, Henan 450001, China

[b]Department of Logistics and Maritime Studies, The Hong Kong Polytechnic University,

Hung Hom, Kowloon, Hong Kong

Abstract We re-visit the two-agent scheduling on a parallel-batch machine to minimize the makespan, where the jobs have release dates and linear deteriorating processing times. The objective is to minimize the makespan of agent A with the makespan of agent B being bounded. In the paper [Tang, L.X., Zhao, X.L., Liu, J.Y., & Leung, J.Y.T. (2017) Competitive two-agent scheduling with deteriorating jobs on a single parallel-batching machine. European Journal of Operational Research, 263, 401-411.], the authors reported comprehensive research for this scheduling model. Especially, they presented polynomial-time algorithms for the following four problems. In the first, the batch capacity is unbounded and the two agents are compatible. In the second, the batch capacity is bounded, the two agents are incompatible, the A-jobs have a fixed number of normal processing times, and the B-jobs have a common release date. In the third and forth, the batch capacity is bounded, the two agents are compatible, and the release dates and normal processing times are either agreeable or reversely agreeable. But their

*Corresponding author: Jinjiang Yuan. Email address: yuanjj@zzu.edu.cn

discussions for the above four problems are logically confusing. In this paper we present a more efficient polynomial-time algorithm for the first problem and show that the other three problems are NP-hard. We also present a pseudo-polynomial-time algorithm for the version where the batch capacity is bounded, the two agents are incompatible, and the A-jobs and the B-jobs have their common release dates, respectively. We finally present a strongly polynomial-time algorithm for the version where the batch capacity is unbounded and the two agents are incompatible.

Keywords: parallel-batch; two agents; release dates; deterioration

1 Introduction

This paper is related to several well-studied scheduling models. For our purpose, we only review the most pertinent studies.

Multi-agent scheduling was first introduced by Baker and Smith (2003), and Agnetis et al. (2004). Especially, in two-agent scheduling, there are two agents A and B. Each agent $X = A, B$ has its own set of jobs $\mathcal{J}^X = \{J_1^X, J_2^X, \ldots, J_{n_X}^X\}$. The jobs of agent X are called the X-jobs. We assume that the A-jobs and the B-jobs are disjoint, i.e., $\mathcal{J}^A \cap \mathcal{J}^B = \emptyset$. In a scheduling environment, agent X has a scheduling cost f^X that only depends on the scheduling of the X-jobs. The two agents compete for a common processing resource to process their jobs. The problem has been widely studied in the literature. The methodology and development of multi-agent scheduling can be found in Leung et al. (2010), Perez-Gonzalez and Framinan (2014), Agnetis et al. (2014), Fan and Cheng (2016), and Yuan (2016).

Parallel-batch scheduling was first introduced by Lee et al. (1992). In this scheduling model, a parallel-batch processor is a production facility that can handle up to a batch of c jobs simultaneously, where c is the batch capacity, and the processing time of a batch is equal to the longest processing time of the jobs contained in the batch. All the jobs contained in the same batch have a common starting time and a common completion time. There are two variants of parallel-batch scheduling, namely the unbounded model, in which $c = \infty$, and the bounded model, in which $c < n$. The unbounded model was first studied in Brucker et al. (1998). In the literature a scheduling problem in the

parallel-batch machine setting is denoted by $1|\text{p-batch}, r_j, c|f$, where "p-batch" refers to the parallel-batch machine, r_j means that the jobs have release dates, c is the batch capacity, and f is the scheduling cost to be minimized. For the two-agent scheduling on a parallel-batch machine, we use

$$1|\text{p-batch}, r_j^X, c, CF/IF|f^A : f^B \leq Q_B$$

to denote a problem, where CF means that the two agents are compatible, i.e., the jobs from different agents can be assigned to a common batch, and IF means that the two agents are incompatible, i.e., the jobs from different agents cannot be assigned to a common batch. The goal of the problem is to find a feasible schedule that minimizes f^A, subject to the restriction that $f^B \leq Q_B$. Research on two-agent parallel-batch scheduling can be found in Li and Yuan (2012), Fan et al. (2013), and Wang et al. (2017).

Scheduling with deteriorating jobs was first studied by Melnikov and Shafransky (1979). Although there are many models in this research setting, we only consider the linear deterioration version in which the processing time of a job J_j in a schedule is given by $p_j = \alpha_j(a+bt)$ with $a \geq 0$, $b > 0$, and $\alpha_j > 0$, where α_j is called the normal processing time of job J_j and t is the starting time of job J_j in the schedule. In this scheduling model, a useful observation is that, if a set of jobs $\mathcal{J}$ is consecutively processed (in an arbitrary order) starting at time t, then the last job is completed at time

$$(t + \frac{a}{b}) \prod_{J_j \in \mathcal{J}} (1 + b\alpha_j) - \frac{a}{b}. \tag{1}$$

Equation (1) has been repeatedly used in the literature and will also used in this paper. Li et al. (2011) studied problem $1|\text{p-batch}, p_j = \alpha_j t, r_j, c|C_{\max}$. Specifically, they presented an $O(n \log n)$-time algorithm to solve problem $1|\text{p-batch}, p_j = \alpha_j t, r_j, c = \infty|C_{\max}$. We show in Section 2 that their algorithm can be easily extended to solve problem $1|\text{p-batch}, p_j = \alpha_j(a + bt), r_j, c = \infty|C_{\max}$ in $O(n \log n)$ time.

In this paper we consider the two-agent scheduling on a single parallel-batch machine, where the jobs have release dates and linear deteriorating processing times. The objective is to find a feasible schedule that minimizes the makespan of one agent with the restriction that the makespan of the other agent cannot exceed a given bound. In this scheduling model, there are two agents A and B, the X-jobs (for $X = A, B$) are given by $\mathcal{J}^X = \{J_1^X, J_2^X, \ldots, J_{n_X}^X\}$ with $\mathcal{J}^A \cap \mathcal{J}^B = \emptyset$, each job J_j^X has a release date $r_j^X > 0$, and the

actual processing time of job J_j^X is given by $p_j^X = \alpha_j^X(a+bt)$, where α_j^X is the normal processing time of job J_j^X for $j = 1, 2, \ldots, n_X$, $a \geq 0$ and $b > 0$ are constants, and t is the starting time of job J_j^X in a real schedule. A schedule is determined by a batch sequence $\pi = (B_1, B_2, \ldots, B_m)$, which indicates that the $n = n_A + n_B$ jobs in $\mathcal{J}^A \cup \mathcal{J}^B$ are partitioned into m batches $B_1, B_2, \ldots, B_m$ and are processed in this order. If $B_i \subseteq \mathcal{J}^X$ for some i, we call B_i an X-batch or an X-pure batch. We use $r_{B_i} = \max\{r_j : J_j \in B_i\}$ to denote the release date of batch B_i and $\alpha_{B_i} = \max\{\alpha_j : J_j \in B_i\}$ to denote the normal processing time of batch B_i. Hence, if batch B_i starts at time t in schedule π, then $t \geq r_{B_i}$ and the actual processing time of batch B_i in π is $\alpha_{B_i}(a+bt)$, so batch B_i is completed at time $t + \alpha_{B_i}(a+bt)$. For a given schedule π, we use $C_j^X(\pi)$ to denote the completion time of job J_j^X in π. Then the makespan of agent X under π is given by $C_{\max}^X(\pi) = \max\{C_j^X(\pi) : j = 1, 2, \ldots, n_X\}$. Following the notation in Tang et al. (2017), we use IF and CF to represent incompatible agents and compatible agents, respectively. Then the scheduling problems studied in this paper can be denoted by

$$1|\text{p-batch}, p_j^X = \alpha_j^X(a+bt), r_j^X, c, CF/IF|C_{\max}^A : C_{\max}^B \leq Q_B, \tag{2}$$

which aims to find a feasible schedule that minimizes $C_{\max}^A$, subject to the restriction that $C_{\max}^B \leq Q_B$. In order to express a problem more efficiently in the following table, we just state the choices of c and CF/IF, together with the notation r_j^X. For example, problem $(c = \infty, CF, r_j^X)$ refers to a problem in (2) in which the batch capacity is unbounded, the two agents are compatible, and the jobs have their respective release dates, while problem $(c < n, IF, r_j^X)$ refers to a problem in (2) in which the batch capacity is bounded, the two agents are incompatible, and the jobs have their respective release dates. For the case where r_j^X is omitted, all the jobs are released at a common time $t_0 \geq 0$.

The scheduling model in (2) was first studied by Tang et al. (2017), who considered other regular scheduling costs such as $f_{\max}$, $\sum C_j$, and $\sum U_j$. The work in Tang et al. (2017) is innovative as they proposed a scheduling model of a real production problem for the ingot soaking process of a primary rolling plant in the steel industry. We complement their study by analyzing some open cases to complete the research on this scheduling model related to makespan minimization. Table 1 summarizes the comprehensive research in Tang et al. (2017) concerning makespan minimization with release dates.

Table 1: The related results in Tang et al. (2017).

Scheduling Problem	Algorithm	Reference in Tang et al. (2017)
$c = \infty,\ CF,\ r_j^X$	$O(n^3)$	Discussion of Algorithm 1
$c < n,\ IF,\ r_j^X,\ (l, r^B)$	$O(n_A^{2l} c^l)$	Discussion of Algorithm DP6
$c < n,\ CF,\ agr(r_j, \alpha_j)$	$O(n^2 c)$	Discussion of Algorithm 2
$c < n,\ CF,\ revagr(r_j, \alpha_j)$	$O(n^2 c)$	Discussion of Algorithm 2
$c = \infty,\ IF,\ r_j^X$	$O(n_A n_B n \log(Q'_U - Q'_L))$	Theorem 3.2.3

Note that the following notations are used in Table 1:

• Q'_U is an upper bound on $C^A_{\max}$ and Q'_L is a lower bound on $C^A_{\max}$.

• (l, r^B) denotes the restriction that the A-jobs have l distinct normal processing times and the B-jobs have a common release date r^B.

• $agr(r_j, \alpha_j)$ denotes that the release dates and normal processing times of the jobs are agreeable, i.e., $r_i < r_j$ implies that $\alpha_i \leq \alpha_j$.

• $revagr(r_j, \alpha_j)$ denotes that the release dates and normal processing times of the jobs are reversely agreeable, i.e., $r_i < r_j$ implies that $\alpha_i \geq \alpha_j$.

Among the plentiful results obtained by Tang et al. (2017) are polynomial-time solution algorithms for the following four problems.

Problem 1. $1|\text{p-batch}, p_j^X = \alpha_j^X(a + bt), r_j^X, c = \infty, CF|C^A_{\max} : C^B_{\max} \leq Q_B$.

Problem 2. $1|\text{p-batch}, p_j^X = \alpha_j^X(a + bt), r_j^X, c < n, IF|C^A_{\max} : C^B_{\max} \leq Q_B$ with the restriction that the A-jobs have l (a constant) distinct normal processing times and the B-jobs have a common release date.

Problem 3. $1|\text{p-batch}, p_j^X = \alpha_j^X(a + bt), agr(r_j, \alpha_j), c < n, CF|C^A_{\max} : C^B_{\max} \leq Q_B$.

Problem 4. $1|\text{p-batch}, p_j^X = \alpha_j^X(a + bt), revagr(r_j, \alpha_j), c < n, CF|C^A_{\max} : C^B_{\max} \leq Q_B$.

Unfortunately, their discussions for the above four problems are logically confusing. Then we re-visit these problems and make the following contributions.

• For Problem 1, we present an $O(n^2 \log n)$-time algorithm, which is more efficient than their $O(n^3)$-time algorithm.

• For Problems 2-4, we study the following related problem:

Problem 5. 1|p-batch, $p_j^X = \alpha_j^X(a+bt), r^X, c < n|C_{\max}^A : C_{\max}^B \leq Q_B$, where r^X means that the X-jobs have a common release date r^X, $X = A, B$.

We show that Problem 5 is NP-hard even when $c = 1$, $n_A = 1$, and either the condition $agr(r_j, \alpha_j)$ holds or the condition $revagr(r_j, \alpha_j)$ holds. Note that $c = 1$ implies that CF and IF have no difference. As a consequence, Problems 2-4 are NP-hard. We further present a pseudo-polynomial-time algorithm for Problem 5 under the IF assumption.

- Finally we study the following problem.

Problem 6. 1|p-batch, $p_j^X = \alpha_j^X(a+bt), r_j^X, c = \infty, IF|C_{\max}^A : C_{\max}^B \leq Q_B$.

For this problem, Tang et al. (2017) presented an $O(n_A n_B n \log(Q'_U - Q'_L))$-time algorithm, which is weakly polynomial. In this paper, we present an $O(n^3)$-time algorithm, which is strongly polynomial.

We organize the rest of the paper as follows: In Section 2 we present a fundamental algorithm, which is repeatedly used in this paper. In Section 3 we study Problem 1. In Section 4 we study Problem 5, together with discussions of the NP-hardness of Problems 2-4. In Section 5 we study Problem 6. We conclude the paper in the last section.

2 A fundamental algorithm

For problem 1|p-batch, $p_j = \alpha_j t, r_j, c = \infty|C_{\max}$, Li et al. (2011) presented a dynamic programming algorithm to solve the problem in $O(n \log n)$ time. In the following we borrow their method to solve problem

$$1|\text{p-batch}, p_j = \alpha_j(a+bt), r_j, c = \infty|C_{\max}. \tag{3}$$

By the job-shifting argument, the following lemma can be easily verified.

Lemma 2.1. *For the problem in (3), there exists an optimal batch sequence $\pi = (B_1, B_2, \ldots, B_m)$ such that if two jobs J_i and J_j belong to distinct batches with $J_i \in B_x$, $J_j \in B_y$, and $x < y$, then $\alpha_i > \alpha_j$.*

If there exists two jobs J_i and J_j such that $r_i \leq r_j$ and $\alpha_i \leq \alpha_j$, then we can put J_i in the same batch as J_j without increasing the makespan. Thus, we can delete job J_i from the job set. Therefore, by an $O(n \log n)$-time reduction, we may assume that the jobs can

be re-indexed such that $r_1 < r_2 < \cdots < r_n$ and $\alpha_1 > \alpha_2 > \cdots > \alpha_n$. This assumption, together with Lemma 2.1, leads to the following corollary.

Corollary 2.1. *There exists an optimal batch sequence $\pi = (B_1, B_2, \ldots, B_m)$ for the problem in (3) such that each batch B_x is in the form $B_x = \{J_j : l \le j \le u\}$ for some numbers l and u.*

Now we present a dynamic programming algorithm to solve the problem in (3).

Algorithm DP1. Define $F(k)$ as the minimum makespan $C_{\max}$ of a partial schedule of the jobs $J_1, J_2, \ldots, J_k$. By Corollary 2.1, in an optimal schedule of the jobs $J_1, J_2, \ldots, J_k$ assuming $F(k)$, the last batch can be chosen as $\{J_{i+1}, J_{i+2}, \ldots, J_k\}$ for some index i. The initial condition is $F(0) = 0$ and the recursive formula for $F(k)$ is given by

$$F(k) = \min_{0 \le i \le k-1} \{\max\{F(i), r_k\} \cdot (1 + b\alpha_{i+1}) + a\alpha_{i+1}\}. \tag{4}$$

The optimal value of the problem in (3) is $F(n)$. A straightforward implementation of the algorithm requires $O(k)$ time to compute $F(k)$, given $F(i)$, $0 \le i \le k-1$, so algorithm DP1 runs in $O(n^2)$ time.

Similar to the discussion in Li et al. (2011), we can also adopt the technique provided by Poon and Zhang (2004) to reduce the running time of DP1 to $O(n \log n)$ in the following way.

First, we set $F(0) = 0$. Generally, if $F(0), F(1), \ldots, F(k-1)$ with $1 \le k \le n$ are given, we can easily calculate the value $F(k)$. Note that

$$0 = F(0) \le F(1) \le \cdots \le F(n). \tag{5}$$

Since $F(k) > r_k$, from (5), there is a unique index m_k such that $F(m_k) \le r_k < F(m_k+1)$. This implies that

$$\max\{F(i), r_k\} = \begin{cases} r_k, & \text{if } i \le m_k, \\ F(i), & \text{otherwise.} \end{cases}$$

Thus, the recursive relation in (4) can be re-written as

$$F(k) = \min \begin{cases} r_k(1 + b\alpha_{m_k+1}) + a\alpha_{m_k+1}, \\ F(i)(1 + b\alpha_{i+1}) + a\alpha_{i+1}, \quad m_k + 1 \le i \le k-1. \end{cases} \tag{6}$$

Next, let $A_k = \{F(i)(1 + b\alpha_{i+1}) + a\alpha_{i+1} : m_k + 1 \leq i \leq k - 1\}$, which is the main part for calculating $F(k)$ in (6). To obtain A_{k+1} from A_k, we compute m_{k+1} (by a sequential search for r_{k+1} on the sequence $F(m_k + 1), F(m_k + 2), \ldots, F(n)$), deleting the elements in $A_k \backslash A_{k+1} = \{F(i)(1 + b\alpha_{i+1}) + a\alpha_{i+1} : m_k + 1 \leq i \leq m_{k+1}\}$, and insert the element $F(k)(1 + b\alpha_{k+1}) + a\alpha_{k+1}$. Third, we store A_k as a heap and keep an array P of pointers so that $P[i]$ points to the element $F(i)(1 + b\alpha_{i+1}) + a\alpha_{i+1}$ in the heap when it is inserted. Note that finding the minimum value in A_k takes a constant time and each value of the form $F(k)(1 + b\alpha_{k+1}) + a\alpha_{k+1}$ is inserted into and deleted from the heap at most once. Thus, our algorithm requires $O(n \log n)$ time in total. Then we have the following result.

Theorem 2.1. *The problem in (3) can be solved in $O(n \log n)$ time.*

3 The first problem

In Section 3.2.1.1 of Tang et al. (2017), the authors studied Problem 6, i.e.,

$$1|\text{p-batch}, p_j^X = \alpha_j^X(a + bt), r_j^X, c = \infty, IF|C_{\max}^A : C_{\max}^B \leq Q_B,$$

and provided the following lemma.

Lemma 3.2.1 in Tang et al. (2017). *For Problem 6, there is an optimal batch schedule $\pi = (B_1, B_2, \ldots, B_m)$ such that if two jobs J_i and J_j belong to the same agent and distinct batches with $J_i \in B_x$, $J_j \in B_y$ and $x < y$, then $\alpha_i > \alpha_j$.*

In Section 3.2.1.2 of Tang et al. (2017), the authors studied Problem 1, i.e.,

$$1|\text{p-batch}, p_j^X = \alpha_j^X(a + bt), r_j^X, c = \infty, CF|C_{\max}^A : C_{\max}^B \leq Q_B.$$

It is not hard to see that the result in their Lemma 3.2.1 is still valid for Problem 1. But their following statement is unreasonable. As in the first paragraph of Section 3.2.1.2 in Tang et al. (2017), the author wrote the statement:

In view of Lemma 3.2.1, we may assume that the jobs have been indexed such that

$$r_1 < \cdots < r_n \text{ and } \alpha_1 > \cdots > \alpha_n. \tag{7}$$

After this statement, the authors established the following properties for Problem 1.

Lemma 3.2.4 in Tang et al. (2017). *For Problem 1, there is an optimal schedule in the form* (π_1, π_2, π_3) *that has the following properties:*

(1) the partial schedule π_2 *contains only all the* B*-pure batches,* π_3 *contains part of* A*-pure batches (if any), and* π_1 *contains the remaining batches;*

(2) all the jobs (batches) in the partial schedules π_1 *and* π_2 *are scheduled in decreasing order of their normal processing times, all the jobs (batches) in the partial schedules* π_1 *and* π_3 *are also scheduled in decreasing order of their normal processing times.*

Finally, based on Lemma 3.2.1 (in fact, the above statement) and Lemma 3.2.4, the authors presented an $O(n^3)$ algorithm (their **Algorithm 1**) for Problem 1.

Since two jobs belonging to distinct agents cannot be absorbed by each other, the statement in (7) does not hold in general. From the context of Tang et al. (2017), it seems that they studied Problem 1 under the assumption that the jobs are indexed in such a way that $r_1 < \cdots < r_n$ and $\alpha_1 > \cdots > \alpha_n$. In this case, property (2) in their Lemma 3.2.4 is inaccurate and the implementation of their Algorithm 1 is confusing. So we present in the following a new algorithm for Problem 1.

Consider an instance $\{J_1, J_2, \ldots, J_n\}$ of Problem 1 as follows:

$$1|\text{p-batch}, p_j^X = \alpha_j^X(a+bt), r_j^X, c = \infty, CF|C_{\max}^A : C_{\max}^B \leq Q_B.$$

If there exist two jobs J_i and J_j belonging to the same agent such that $r_i \leq r_j$ and $\alpha_i \leq \alpha_j$, from the fact that $c = \infty$, we can always put J_i in the same batch as J_j without increasing the makespan of each agent. This implies that we can delete job J_i from the job set in our discussion. Therefore, we may assume that, for every two jobs J_i and J_j belonging to the same agent, we have that $r_i \neq r_j$ and $\alpha_i \neq \alpha_j$, and moreover, $r_i < r_j$ implies that $\alpha_i > \alpha_j$. As a result, we can re-index the jobs of each agent X such that $r_1^X < r_2^X < \cdots < r_{n_X}^X$ and $\alpha_1^X > \alpha_2^X > \cdots > \alpha_{n_X}^X$ for $X = A, B$.

Note that the result of Lemma 3.2.1 in Tang et al. (2017) for incompatible agents are also valid for compatible agents and can be re-stated in the following form.

Lemma 3.1. *For Problem 1, there is an optimal batch schedule* $\pi = (B_1, B_2, \ldots, B_m)$

such that if two jobs J_i and J_j belong to the same agent and distinct batches with $J_i \in B_x$, $J_j \in B_y$, and $x < y$, then $\alpha_i > \alpha_j$.

The problem in (3), i.e., $1|\text{p-batch}, p_j = \alpha_j(a+bt), r_j, c = \infty|C_{\max}$, is important for our discussion. Relating to this problem, we consider the following three auxiliary problems.

Problem Aux(k_A, k_B): This is the single-agent scheduling problem in (3) for the job set $\{J_1^A, \ldots, J_{k_A}^A, J_1^B, \ldots, J_{k_B}^B\}$, where $0 \le k_A \le n_A$ and $0 \le k_B \le n_B$. We use $C(k_A, k_B)$ to denote the optimal value of Problem Aux(k_A, k_B) and $\pi(k_A, k_B)$ to denote an arbitrary optimal schedule.

Problem Aux$^A(k_A^+)$: This is the single-agent scheduling problem in (3) for the job set $\{J_{k_A+1}^A, \ldots, J_{n_A}^A\}$, where $0 \le k_A \le n_A - 1$, with the restriction that no jobs start earlier than $C(k_A, n_B)$. We use $C^A(k_A^+)$ to denote the optimal value of Problem Aux$^A(k_A^+)$ and use $\pi^A(k_A^+)$ to denote an arbitrary optimal schedule.

Problem Aux$^B(k_B^+)$: This is the single-agent scheduling problem in (3) for the job set $\{J_{k_B+1}^B, \ldots, J_{n_B}^B\}$, where $0 \le k_B \le n_B - 1$, with the restriction that no jobs start earlier than $C(n_A, k_B)$. We use $C^B(k_B^+)$ to denote the optimal value of Problem Aux$^B(k_B^+)$ and use $\pi^B(k_B^+)$ to denote an arbitrary optimal schedule.

Lemma 3.2. *For Problem 1 with a feasible instance, one of the following $n+1$ schedules is optimal: (i) $\pi(n_A, n_B)$; (ii) $(\pi(k_A, n_B), \pi^A(k_A^+))$, $0 \le k_A \le n_A - 1$; and (iii) $(\pi(n_A, k_B), \pi^B(k_B^+))$, $0 \le k_B \le n_B - 1$.*

Proof. Let $\pi = (B_1, B_2, \ldots, B_m)$ be an optimal schedule for Problem 1 with the property stated in Lemma 3.1. We consider the following three distinct cases.

Case 1: $C_{\max}^A(\pi) = C_{\max}^B(\pi)$. Then both $J_{n_A}^A$ and $J_{n_B}^B$ are included in the last batch B_m of π. Let $\pi^* = \pi(n_A, n_B)$. Then $\max\{C_{\max}^A(\pi^*), C_{\max}^B(\pi^*)\} = C_{\max}(\pi^*) \le C_{\max}(\pi) = C_{\max}^A(\pi) = C_{\max}^B(\pi)$. This means that $\pi^* = \pi(n_A, n_B)$ is also optimal for Problem 1.

Case 2: $C_{\max}^A(\pi) > C_{\max}^B(\pi)$. Then the last batch of π consists of some jobs of agent A. From Lemma 3.1, there is a job index $k_A < n_A$ and a batch index $y < m$ such that $J_{n_B}^B \in B_y$ and $B_{y+1} \cup B_{y+2} \cup \cdots \cup B_m = \{J_{k_A+1}^A, J_{k_A+2}^A, \ldots, J_{n_A}^A\}$. Let π' be the sub-schedule of $B_1, B_2, \ldots, B_y$ in π and let π'' be the sub-schedule of $B_{y+1}, B_{y+2}, \ldots, B_m$ in π. Then $\pi = (\pi', \pi'')$, $C_{\max}^B(\pi) = C_{\max}(\pi')$, and $C_{\max}^A(\pi) = C_{\max}(\pi) = C_{\max}(\pi'')$. Now let $\pi^* = (\pi(k_A, n_B), \pi^A(k_A^+))$ be the schedule for Problem 1 that is obtained from

$\pi = (\pi', \pi'')$ by replacing π' and π'' with $\pi(k_A, n_B)$ and $\pi^A(k_A^+)$, respectively. For Problem $\text{Aux}(k_A, n_B)$, $\pi(k_A, n_B)$ is an optimal schedule and π' is a feasible schedule. Then

$$C^B_{\max}(\pi^*) \leq C(k_A, n_B) \leq C_{\max}(\pi') = C^B_{\max}(\pi). \tag{8}$$

The inequality in (8) also implies that π'' (which starts no earlier than $C_{\max}(\pi')$) is a feasible schedule for Problem $\text{Aux}^A(k_A^+)$. From optimality, we further have

$$C^A_{\max}(\pi^*) = C^A(k_A^+) \leq C_{\max}(\pi'') = C^A_{\max}(\pi). \tag{9}$$

From (8) and (9), we have that $C^A_{\max}(\pi^*) \leq C^A_{\max}(\pi)$ and $C^B_{\max}(\pi^*) \leq C^B_{\max}(\pi)$. Consequently, $\pi^* = (\pi(k_A, n_B), \pi^A(k_A^+))$ is also optimal for Problem 1.

Case 3: $C^B_{\max}(\pi) > C^A_{\max}(\pi)$. This case is essentially the same as Case 2. By replacing the roles of A and B, we can show that $(\pi(n_A, k_B), \pi^B(k_B^+))$ is optimal for Problem 1 for some job index k_B with $0 \leq k_B \leq n_B - 1$. The lemma follows. □

By Theorem 2.1, the problem in (3) for a set of n jobs is solvable in $O(n \log n)$ time. Then each of the $n+1$ schedules in Lemma 3.2 can be obtained in $O(n \log n)$ time. Thus, we can first generate the $n+1$ schedules in $O(n^2 \log n)$ time and then pick the best feasible schedule for Problem 1. This leads to the following result.

Theorem 3.1. *Problem 1 is solvable in $O(n^2 \log n)$ time.*

4 Study on Problem 5 and extensions

4.1 NP-hardness proof

The NP-hardness of the single-agent problem $1|\text{p-batch}, p_j = \alpha_j t, r_j, c < n|C_{\max}$ was established in Li et al. (2011). This implies that the two-agent problem

$$1|\text{p-batch}, p_j^X = \alpha_j^X(a + bt), r_j^X, c < n|C^A_{\max} : C^B_{\max} \leq Q_B \tag{10}$$

is also NP-hard for both incompatible agents and compatible agents. Tang et al. (2017) presented polynomial-time algorithms for Problems 2-4, all of which are sub-problems of the problem in (10). However, their treatments of Problems 2-4 are logically confusing. We now show that Problem 5, i.e.,

$$1|\text{p-batch}, p_j^X = \alpha_j^X(a + bt), r^X, c < n|C^A_{\max} : C^B_{\max} \leq Q_B,$$

is NP-hard, which implies that Problems 2-4 are NP-hard, too.

Theorem 4.1. *Problem 5 is NP-hard even when $c = 1$ and $n_A = 1$. Moreover, the problem is also NP-hard under each one of the following restrictions:*

(i) The release dates and normal processing times of the jobs are agreeable, i.e., $r_i < r_j$ implies that $\alpha_i \leq \alpha_j$.

(ii) The release dates and normal processing times of the jobs are reversely agreeable, i.e., $r_i < r_j$ implies that $\alpha_i \geq \alpha_j$.

Proof. We use the Subset-Product Problem for the reduction. For this problem, Garey and Johnson (1979) erroneously reported its strong NP-completeness. The binary NP-completeness of the problem was established in Ng et al. (2010).

Subset-Product Problem: Given $t+1$ positive integers $\{a_1, a_2, \ldots, a_t, H\}$ such that $\prod_{i=1}^{t} a_i = H^2$ and $a_i \geq 2$ for $i = 1, 2, \ldots, t$, does there exist a subset N_1 of the index set $N = \{1, 2, \ldots, t\}$ such that $\prod_{i \in N_1} a_i = H$?

For a given instance $(a_1, a_2, \ldots, a_t; H)$ of the Subset-Product Problem, we construct a job instance of the decision version of Problem 5 as follows:

• $a = 0$, $b = 1$, $c = 1$, $n_A = 1$, and $n_B = t$.

• Job J_1^A has a release date $r_1^A = H$ and a normal processing time $\alpha_1^A = h - 1$, where $2 \leq h \leq H$.

• Jobs $J_1^B, J_2^B, \ldots, J_t^B$ have release dates $r_i^B = 1$ and normal processing times $\alpha_i^B = a_i - 1$ for $i = 1, 2, \ldots, t$.

• The decision asks whether there is a feasible schedule π such that $C_{\max}^A(\pi) \leq hH$ and $C_{\max}^B(\pi) \leq hH^2$.

The above construction can be performed in polynomial time. In the following we show that the Subset-Product Problem instance has a solution if and only if the above constructed job instance has a feasible schedule π such that $C_{\max}^A(\pi) \leq hH$ and $C_{\max}^B(\pi) \leq hH^2$.

Necessity Proof: Suppose that the Subset-Product Problem instance has a solution. Then there exists a subset N_1 of the index set $N = \{1, 2, \ldots, t\}$ such that $\prod_{i \in N_1} a_i = H$. Let $N_2 = N \backslash N_1$. Set $\mathcal{J}_{N_i}^B = \{J_j^B : j \in N_i\}$ for $i = 1, 2$. Then we construct a schedule $\pi = (\mathcal{J}_{N_1}^B \prec J_1^A \prec \mathcal{J}_{N_2}^B)$ in the following way.

– The B-jobs in $\mathcal{J}_{N_1}^B$ are processed consecutively from time $r^B = 1$. Since $\prod_{i \in N_1} a_i = H$, the completion time of the last job in $\mathcal{J}_{N_1}^B$ is given by $C(\mathcal{J}_{N_1}^B) = 1 \cdot \prod_{i \in N_1} (\alpha_i^B + 1) = 1 \cdot \prod_{i \in N_1} a_i = H$. Thus, the B-jobs in $\mathcal{J}_{N_1}^B$ are scheduled in the time interval $[1, H]$ in π.

– The A-job J_1^A is processed from time $r_1^A = H$. Then the completion time of J_1^A is given by $C_1^A(\pi) = S_1^A(\pi) \cdot (\alpha_1^A + 1) = hH$. Thus, $C_{\max}^A(\pi) = hH$.

– Finally, the B-jobs in $\mathcal{J}_{N_2}^B$ are processed consecutively from time hH. Since $\prod_{i \in N_2} a_i = H$, the completion time of the last job in $\mathcal{J}_{N_2}$ is given by $C(\mathcal{J}_{N_2}^B) = hH \cdot \prod_{i \in N_2} (\alpha_i^B + 1) = hH \cdot \prod_{i \in N_2} a_i = hH^2$. It follows that $C_{\max}^B(\pi) = hH^2$.

Figure 3 displays the structure of schedule π. The above discussion implies that π is our required feasible schedule.

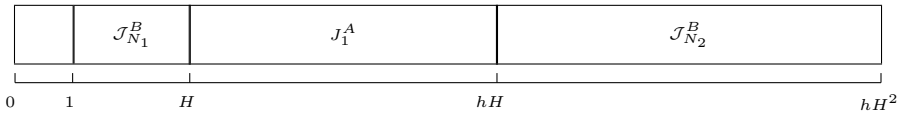

Figure 3. The schedule π

Sufficiency Proof: Suppose that π is a feasible schedule of the above job instance such that $C_{\max}^A(\pi) \leq hH$ and $C_{\max}^B(\pi) \leq hH^2$. We prove that the Subset-Product Problem instance has a solution in the following. Note that $c = 1$.

Let τ be the starting time of job J_1^A in π. Then $\tau \geq r_1^A = H$ and $h\tau = \tau \cdot (\alpha_1^A + 1) = C_{\max}^A(\pi) \leq hH$. This just implies that $\tau = H$. Consequently, job J_1^A is scheduled in the time interval $[H, hH]$ in π.

From the fact that the B-jobs have a common release date $r^B = 1$ and $C_{\max}^B(\pi) \leq hH^2$, we deduce that the B-jobs are processed between 1 and hH^2. We use $\mathcal{J}_{N_1}^B$ to denote the set of the B-jobs processed in the time interval $[1, H]$ and $\mathcal{J}_{N_2}^B$ to denote the set of the B-jobs processed in the time interval $[hH, hH^2]$, where N_1 and N_2 form a partition of the index set $\{1, 2, \ldots, t\}$. Then

$$C(\mathcal{J}_{N_1}^B) = \prod_{i \in N_1} a_i \leq H$$

and

$$C(\mathcal{J}_{N_2}^B) = hH \cdot \prod_{i \in N_2} a_i \leq hH^2.$$

It follows that $\prod_{i \in N_1} a_i \leq H$ and $\prod_{i \in N_2} a_i \leq H$. Since $\prod_{i \in N_1} a_i \cdot \prod_{i \in N_2} a_i = H^2$, we have that $\prod_{i \in N_1} a_i = \prod_{i \in N_2} a_i = H$. This implies that the Subset-Product Problem instance has a solution.

Since $c = 1$ in our job instance, the above discussion implies that Problem 5 is NP-hard for both compatible agents and incompatible agents.

Note that our job instance satisfies the conditions that

$$r_1^B = r_2^B = \cdots = r_t^B = 1 < r_1^A = H$$

and

$$\alpha_i^B = a_i - 1 \leq H - 1 \text{ for } i = 1, 2, \ldots, t.$$

When $h = H$, we have that $\alpha_i^B = a_i - 1 \leq \alpha_1^A = H - 1$ for $i = 1, 2, \ldots, t$. In this case, all the job release dates and normal processing times are agreeable. When $h = 2$, we have that $\alpha_i^B = a_i - 1 \geq \alpha_1^A = h - 1 = 1$ for $i = 1, 2, \ldots, t$. In this case, all the job release dates and normal processing times are reversely agreeable. It follows that Problem 5 is also NP-hard under each one of the restrictions (i) and (ii). The result follows. □

As a consequence of Theorem 4.1, we have

Corollary 4.1. *Problems 2-4 are NP-hard.*

4.2 Problem 5 under the IF assumption

We now consider Problem 5 under the IF assumption, i.e.,

$$1|\text{p-batch}, p_j^X = \alpha_j^X(a+bt), r^X, c < n, IF|C_{\max}^A : C_{\max}^B \leq Q_B. \tag{11}$$

To present a pseudo-polynomial-time algorithm, we assume that all the parameters a, b, α_j^X, and r^X are non-negative integers. As in Tang et al. (2017), we use the following Full Batch Longest Normal Processing Time (FBLNPT) rule to form the batches in our discussion.

FBLNPT: Sort the jobs in the non-increasing order of their normal processing times and form full batches by always assigning the first c unassigned jobs into a full batch. Finally, the remaining fewer than c jobs, if any, are assigned to a non-full batch.

It is easy to verify that, for the single-agent problem with a common release date r_0, i.e.,

$$1|\text{p-batch}, p_j = \alpha_j(a+bt), r_0, c < n|C_{\max}, \tag{12}$$

the schedule in which the batches are generated by the FBLNPT rule, starting at time r_0 and processed consecutively, is an optimal schedule.

Lemma 4.1. *For a feasible instance of the problem in (11), there exists an optimal schedule satisfying the following conditions:*

(i) the batches of each agent $X = A, B$ are generated by the FBLNPT rule;

(ii) if $r^A \leq r^B$, the batches of agent B are processed consecutively; and

(iii) if $r^B \leq r^A$, the batches of agent A are processed consecutively.

Proof. Since the jobs of agent $X = A, B$ have a common release date r^X, by using the job-swapping argument and job-shifting argument, we can show that the batches of agent X can be generated by the FBLNPT rule in an optimal schedule for the problem in (11). This proves result (i).

Now let $\pi = (B_1, B_2, \ldots, B_m)$ be an optimal schedule with satisfying result (i). Suppose first that $r^A \leq r^B$. Let B_i be the first batch in π that starts no earlier than r^B and let τ be the starting time of B_i. For $X = A, B$, let $\mathcal{B}^X$ be the set of X-batches in $\{B_i, B_{i+1}, \ldots, B_m\}$. Note that $\mathcal{B}^B$ consists of all the B-batches since the jobs of agent B are released at time r^B, so they cannot be scheduled earlier than r^B. We define π' as a new schedule that is obtained from π by re-scheduling the batches in $\{B_i, B_{i+1}, \ldots, B_m\}$ in the following way.

– If B_m is a B-batch, then starting at time τ, first schedule the batches in $\mathcal{B}^A$ consecutively, followed by scheduling the batches in $\mathcal{B}^B$ consecutively.

– If B_m is an A-batch, then starting at time τ, first schedule the batches in $\mathcal{B}^B$ consecutively, followed by scheduling the batches in $\mathcal{B}^A$ consecutively.

It is not hard to verify that $C^A_{\max}(\pi') \leq C^A_{\max}(\pi)$ and $C^B_{\max}(\pi') \leq C^B_{\max}(\pi)$. Then π' is also an optimal schedule satisfying result (i). Since the batches in $\mathcal{B}^B$ (i.e., all the B-batches) are processed consecutively in π', it follows that π' also satisfies result (ii).

When $r^A \geq r^B$, by exchanging the roles of A and B, a same argument leads to an optimal schedule π'' that satisfies results (i) and (iii). The lemma follows. □

From Lemma 4.1(i), we may suppose that all the batches of each agent have been obtained by using the FBLNPT rule, which takes $O(n \log n)$ time. By regarding each batch as a single job, we reduce the problem in (11) to its sub-version with $c = 1$, which is the traditional single-machine two-agent scheduling problem, denoted as

$$1|p_j^X = \alpha_j^X(a + bt), r^X|C^A_{\max} : C^B_{\max} \leq Q_B. \tag{13}$$

When only the B-jobs are considered, by processing all the B-jobs consecutively starting at time r^B, the makespan is given by $(r^B + \frac{a}{b}) \prod_{k=1}^{n_B}(1 + b\alpha_k^B) - \frac{a}{b}$. Hence, to guarantee feasibility of the problem in (13), by using the expression in (1), we assume in the following that

$$Q_B \geq (r^B + \frac{a}{b}) \prod_{k=1}^{n_B}(1 + b\alpha_k^B) - \frac{a}{b}. \tag{14}$$

We consider the following three distinct cases.

Case 1. $r^A = r^B$. In this case, we define two schedules $\pi = (\pi_A, \pi_B)$ and $\sigma = (\sigma_B, \sigma_A)$, where

– π_A is the partial schedule of π that processes all the A-jobs consecutively starting at time $r^A = r^B$,

– π_B is the partial schedule of π that processes all the B-jobs consecutively starting at time $C^A_{\max}(\pi_A)$,

– σ_B is the partial schedule of σ that processes all the B-jobs consecutively starting at time $r^A = r^B$,

– σ_A is the partial schedule of σ that processes all the A-jobs consecutively starting at time $C^B_{\max}(\sigma_B)$.

From Lemma 4.1, one of π and σ is an optimal schedule for the problem in (13). Since π and σ can be generated in $O(n)$ time, the problem in (13) is solvable in $O(n)$ time in this case.

Case 2. $r^A < r^B$. In this case, we first construct a schedule π in which all the A-jobs are scheduled consecutively from time r^A and all the B-jobs are scheduled consecutively from time $\max\{C^A_{\max}, r^B\}$. If $C^B_{\max}(\pi) \leq Q_B$, then π is clearly an optimal schedule. Hence, we assume in the following that $C^B_{\max}(\pi) > Q_B$. This means that the last job in an optimal schedule must be an A-job.

According to Lemma 4.1, together with its proof, there is an optimal schedule in which the jobs are scheduled in the order

$$(\mathcal{J}^A_1 \prec \mathcal{J}^B \prec \mathcal{J}^A_2), \tag{15}$$

where $\mathcal{J}^A_1$ and $\mathcal{J}^A_2$ form a partition of $\mathcal{J}^A$ such that $\mathcal{J}^A_1$ is the set of A-jobs starting at the time interval $[r^A, r^B)$, and $\mathcal{J}^A_2$ is the set of the A-jobs starting after r^B. It is easy to see that in an optimal schedule of the form in (15), we can further require that, for $i = 1, 2$, the jobs in $\mathcal{J}^A_i$ are processed in non-increasing order of their normal processing times. Note that in an optimal schedule of the form in (15), the starting time of the first B-job, denoted as s, is less than $r^B + \alpha^A_{\max}(a + b \cdot r^B)$ since at most one job in $\mathcal{J}^A_1$ is completed after r^B, where $\alpha^A_{\max} = \max\{\alpha^A_i : i = 1, 2, \ldots, n_A\}$. This means that $r^B \leq s < r^B + \alpha^A_{\max}(a + b \cdot r^B)$. We enumerate all the possible choices of s. For each given s, we find an optimal schedule of the form in (15) and pick the best one as the solution for the problem in (13).

We now re-index all the A-jobs such that $\alpha^A_1 \geq \alpha^A_2 \geq \cdots \geq \alpha^A_{n_A}$. Fix a positive integer s with $r^B \leq s < r^B + \alpha^A_{\max}(a + b \cdot r^B)$. For each pair (i, t) of integers with $0 \leq i \leq n_A$ and

$r^A \le t \le s$, let $f_s(i,t)$ be the minimum makespan of a partial schedule that contains the jobs $J_1^A, J_2^A, \ldots, J_i^A$, $J_1^B, J_2^B, \ldots, J_{n_B}^B$ and has the processing structure $(\mathcal{J}_1^A \prec \mathcal{J}^B \prec \mathcal{J}_2^A)$ in (15), where $\mathcal{J}_1^A$ and $\mathcal{J}_2^A$ form a partition of $\{J_1^A, J_2^A, \ldots, J_i^A\}$, the last job in $\mathcal{J}_1^A$ starts before r^B and is completed at time t, and the processing of the jobs in $\mathcal{J}^B$ starts at time s. This implies that $s \ge \max\{t, r^B\}$.

In an optimal schedule assuming $f_s(i,t)$, J_i is either the last job of $\mathcal{J}_1^A$ or the last job of $\mathcal{J}_2^A$. Then we have the following dynamic programming algorithm for calculating $f_s(i,t)$ with fixed s.

- The initialization is

$$f_s(0,t) = \begin{cases} (s + \frac{a}{b}) \prod_{k=1}^{n_B} (1 + b\alpha_k^B) - \frac{a}{b}, & \text{if } s \ge r^B \text{ and } t = r^A, \\ \infty, & \text{otherwise.} \end{cases}$$

- The recursive relation is

$$f_s(i,t) = \min\{f_s(i-1, t^*),\ (1 + b\alpha_i^A) f_s(i-1,t) + a\alpha_i^A\},$$

where $t^* = \frac{t - a\alpha_i^A}{(1 + b\alpha_i^A)}$, which means that if J_i^A starts at time t^*, then it is completed at time $t = (1 + b\alpha_i^A)t^* + a\alpha_i^A$.

- The optimal value of the problem in (13) is given by

$$\min\{f_s(n_A, t) : r^B \le s < M,\ t \le s,\ f_s(0, r^A) \le Q_B\},$$

where $M = r^B + \alpha_{\max}^A (a + b \cdot r^B)$. The corresponding optimal schedule can be found by backtracking.

Note that there are a total of $O(n_A M^2)$ states and each recursion runs in a constant time. So the time complexity of the above dynamic programming algorithm is $O(n_A M^2)$.

Case 3. $r^A > r^B$. According to Lemma 4.1, there is an optimal schedule in which the jobs are scheduled in the order

$$(\mathcal{J}_1^B \prec \mathcal{J}^A \prec \mathcal{J}_2^B), \tag{16}$$

where $\mathcal{J}_1^B$ and $\mathcal{J}_2^B$ form a partition of $\mathcal{J}^B$. We further distinguish the following two distinct cases.

(1) $\mathcal{J}_2^B = \emptyset$. We construct a schedule π in which all the B-jobs are scheduled consecutively from time r^B and all the A-jobs are scheduled consecutively from time $\max\{C_{\max}^B, r^A\}$. Obviously, π is an optimal schedule for the problem in (13).

(2) $\mathcal{J}_2^B \neq \emptyset$. In this case, by exchanging the roles of A and B, the argument for Case 2 is still valid. We only highlight the major differences here.

Fix a positive integer s with $r^A \leq s < r^A + \alpha_{\max}^B(a + b \cdot r^A)$. For each pair (i, t) of integers with $0 \leq i \leq n_B$ and $r^B \leq t \leq s$, let $f_s(i, t)$ be the minimum makespan of a partial schedule that contains the jobs $J_1^B, J_2^B, \ldots, J_i^B$, $J_1^A, J_2^A, \ldots, J_{n_A}^A$ and has the processing structure $(\mathcal{J}_1^B \prec \mathcal{J}^A \prec \mathcal{J}_2^B)$ in (16), where $\mathcal{J}_1^B$ and $\mathcal{J}_2^B$ form a partition of $\{J_1^B, J_2^B, \ldots, J_i^B\}$, the last job in $\mathcal{J}_1^B$ starts before r^A and is completed at time t, and the processing of the jobs in $\mathcal{J}^A$ starts at time s. This implies that $s \geq \max\{t, r^A\}$. Then we have the following dynamic programming algorithm for calculating $f_s(i, t)$ with fixed s.

- The initialization is

$$f_s(0,t) = \begin{cases} (s + \frac{a}{b}) \prod_{k=1}^{n_A}(1 + b\alpha_k^A) - \frac{a}{b}, & \text{if } s \geq r^A \text{ and } t = r^B, \\ \infty, & \text{otherwise.} \end{cases}$$

- The recursive relation is

$$f_s(i,t) = \min\{f_s(i-1, t^*), (1 + b\alpha_i^B) f_s(i-1, t) + a\alpha_i^B\},$$

where $t^* = \frac{t - a\alpha_i^B}{(1 + b\alpha_i^B)}$.

- The optimal value of the problem in (13) is given by

$$\min\{f_s(0, r^B) : r^A \leq s < M, \text{ there exists } t \leq s \text{ such that } f_s(n_B, t) \leq Q_B\},$$

where $M = r^A + \alpha_{\max}^B(a + b \cdot r^A)$. Note that the time complexity of the above dynamic programming algorithm is $O(n_B M^2)$.

From the above analysis, we have the following result.

Theorem 4.2. *Problem* $1|p\text{-}batch, p_j^X = \alpha_j^X(a + bt), r^X, c < n, IF|C_{\max}^A : C_{\max}^B \leq Q_B$ *is solvable in* $O(nM^2)$ *time, where* $M = \max\{r^B + \alpha_{\max}^A(a + b \cdot r^B),\ r^A + \alpha_{\max}^B(a + b \cdot r^A)\}$.

5 A new algorithm for Problem 6

We present in this section a strongly polynomial-time algorithm to solve Problem 6, i.e.,

$$1|\text{p-batch}, p_j^X = \alpha_j^X(a+bt), r_j^X, c=\infty, IF|C_{\max}^A : C_{\max}^B \leq Q_B.$$

Related to our research is the following two-agent scheduling problem

$$1|\text{p-batch}, p_j^X = \alpha_j^X(a+bt), r_j^X, c=\infty, IF|C_{\max}, \tag{17}$$

in which the jobs from distinct agents cannot be assigned to a common batch and the goal is to find a feasible schedule that minimizes the makespan of all the jobs, i.e., $C_{\max} = \max\{C_{\max}^A, C_{\max}^B\}$.

Consider an instance $\{J_1, J_2, \ldots, J_n\}$ of Problem 6 and the problem in (17). If there exist two jobs J_i and J_j belonging to the same agent such that $r_i \leq r_j$ and $\alpha_i \leq \alpha_j$, from the fact that $c = \infty$, we can always put J_i in the same batch as J_j without increasing the makespan of each agent. This implies that we can delete job J_i from the job set in our discussion. Therefore, we may assume that, for every two jobs J_i and J_j belonging to the same agent, we have that $r_i \neq r_j$ and $\alpha_i \neq \alpha_j$, and moreover, $r_i < r_j$ implies that $\alpha_i > \alpha_j$. As a result, we can re-index the jobs of each agent X such that $r_1^X < r_2^X < \cdots < r_{n_X}^X$ and $\alpha_1^X > \alpha_2^X > \cdots > \alpha_{n_X}^X$ for $X = A, B$.

Noting that the result of Lemma 3.2.1 in Tang et al. (2017) for Problem 6 is also valid for the problem in (17), we re-state it in the following lemma.

Lemma 5.1. *For Problem 6 and the problem in (17), there is an optimal batch schedule $\pi = (B_1, B_2, \ldots, B_m)$ in which if two jobs J_i and J_j belong to the same agent and distinct batches with $J_i \in B_x$, $J_j \in B_y$, and $x < y$, then $\alpha_i > \alpha_j$.*

As in Section 3.2, we consider the following auxiliary problem.

Problem Aux$^*(k_A, k_B)$: This is the two-agent scheduling problem in (17) for the job set $\{J_1^A, \ldots, J_{k_A}^A, J_1^B, \ldots, J_{k_B}^B\}$, where $0 \leq k_A \leq n_A$ and $0 \leq k_B \leq n_B$. We use $C^*(k_A, k_B)$ to denote the optimal value of Problem Aux$^*(k_A, k_B)$ and $\pi^*(k_A, k_B)$ to denote an arbitrary optimal schedule possessing the property stated in Lemma 5.1.

From Lemma 5.1, given (k_A, k_B) with $1 \le k_A \le n_A$ and $1 \le k_B \le n_B$, the last batch of $\pi^*(k_A, k_B)$ is either $\mathcal{J}^A(l_A, k_A)$ for some $l_A \in \{0, 1, \ldots, k_A - 1\}$ or $\mathcal{J}^B(l_B, k_B)$ for some $l_B \in \{0, 1, \ldots, k_B - 1\}$, where

$$\mathcal{J}^X(l_X, k_X) = \{J^X_{l_X+1}, J^X_{l_X+2}, \ldots, J^X_{k_X}\}, \quad X = A, B.$$

If the last batch is $\mathcal{J}^A(l_A, k_A)$, then it starts at time $\max\{C^*(l_A, k_B), r^A_{k_A}\}$ and is completed at time $\max\{C^*(l_A, k_B), r^A_{k_A}\}(1+b\cdot\alpha^A_{l_A+1})+a\cdot\alpha^A_{l_A+1}$. If the last batch is $\mathcal{J}^B(l_B, k_B)$, then it starts at time $\max\{C^*(k_A, l_B), r^B_{k_B}\}$ and is completed at time $\max\{C^*(k_A, l_B), r^B_{k_B}\}(1+b\cdot\alpha^B_{l_B+1})+a\cdot\alpha^B_{l_B+1}$.

The above discussion implies that all the values $C^*(k_A, k_B)$ with $0 \le k_A \le n_A$ and $0 \le k_B \le n_B$ can be calculated by the following dynamic programming algorithm.

- The initialization is $C^*(0, 0) = 0$.
- The recursive relation is

$$C^*(k_A, k_B) = \min \begin{cases} \min\{\tau^A_{l_A}(k_A, k_B) : 0 \le l_A \le k_A - 1\}, & \text{if } k_A \ge 1, \\ \min\{\tau^B_{l_B}(k_A, k_B) : 0 \le l_B \le k_B - 1\}, & \text{if } k_B \ge 1, \end{cases}$$

where

$$\tau^A_{l_A}(k_A, k_B) = \max\{C^*(l_A, k_B), r^A_{k_A}\}(1 + b\cdot\alpha^A_{l_A+1}) + a\cdot\alpha^A_{l_A+1}$$

and

$$\tau^B_{l_B}(k_A, k_B) = \max\{C^*(k_A, l_B), r^B_{k_B}\}(1 + b\cdot\alpha^B_{l_B+1}) + a\cdot\alpha^B_{l_B+1}.$$

Finally, the optimal schedule $\pi^*(k_A, k_B)$ can be obtained by backtracking.

Note that there are a total of $O(n_A n_B)$ states and each recursion runs in $O(n)$ time. Then the time complexity of the above dynamic programming algorithm is $O(n_A n_B n)$. We also notice that the optimal value of the problem in (17) is $C^*(n_A, n_B)$. Then we have the following result.

Lemma 5.2. *Problem Aux*(k_A, k_B) with $0 \le k_A \le n_A$ and $0 \le k_B \le n_B$, including the problem in (17), is solvable in $O(n_A n_B n)$ time.*

Let us return to Problem 6 now. As for the analysis of Problem 1, we also need the following two auxiliary problems.

Problem Aux$^{*A}(k_A^+)$: This is the single-agent scheduling problem in (3) for the job set $\{J_{k_A+1}^A, \ldots, J_{n_A}^A\}$, where $0 \leq k_A \leq n_A - 1$, with the restriction that no jobs start earlier than $C^*(k_A, n_B)$. We use $C^{*A}(k_A^+)$ to denote the optimal value of Problem Aux$^{*A}(k_A^+)$ and $\pi^{*A}(k_A^+)$ to denote an arbitrary optimal schedule.

Problem Aux$^{*B}(k_B^+)$: This is the single-agent scheduling problem in (3) for the job set $\{J_{k_B+1}^B, \ldots, J_{n_B}^B\}$, where $0 \leq k_B \leq n_B - 1$, with the restriction that no jobs start earlier than $C^*(n_A, k_B)$. We use $C^{*B}(k_B^+)$ to denote the optimal value of Problem Aux$^{*B}(k_B^+)$ and $\pi^{*B}(k_B^+)$ to denote an arbitrary optimal schedule.

Lemma 5.3. *For Problem 6 with a feasible instance, one of the following n schedules is optimal: (i)* $(\pi^*(k_A, n_B), \pi^{*A}(k_A^+))$, $0 \leq k_A \leq n_A - 1$ *and (ii)* $(\pi^*(n_A, k_B), \pi^{*B}(k_B^+))$, $0 \leq k_B \leq n_B - 1$.

Proof. Let $\pi = (B_1, B_2, \ldots, B_m)$ be an optimal schedule for Problem 6 possessing the property stated in Lemma 5.1. Then the last batch B_m is either an A-batch or a B-batch.

If B_m is an A-batch, from Lemma 5.1, there is a job index $k_A < n_A$ and a batch index $y < m$ such that $J_{n_B}^B \in B_y$ and $B_{y+1} \cup B_{y+2} \cup \cdots \cup B_m = \{J_{k_A+1}^A, J_{k_A+2}^A, \ldots, J_{n_A}^A\}$. Let π' be a sub-schedule of $B_1, B_2, \ldots, B_y$ in π and π'' be a sub-schedule of $B_{y+1}, B_{y+2}, \ldots, B_m$ in π. Then $\pi = (\pi', \pi'')$, $C_{\max}^B(\pi) = C_{\max}(\pi')$, and $C_{\max}^A(\pi) = C_{\max}(\pi) = C_{\max}(\pi'')$. Now let $\pi^* = (\pi^*(k_A, n_B), \pi^{*A}(k_A^+))$ be the schedule for Problem 6 that is obtained from $\pi = (\pi', \pi'')$ by replacing π' and π'' with $\pi^*(k_A, n_B)$ and $\pi^{*A}(k_A^+)$, respectively. For Problem Aux$^*(k_A, n_B)$, $\pi^*(k_A, n_B)$ is an optimal schedule and π' is a feasible schedule. For Problem Aux$^{*A}(k_A^+)$, $\pi^{*A}(k_A^+)$ is an optimal schedule and π'' is a feasible schedule. Hence, $C_{\max}^A(\pi^*) \leq C_{\max}^A(\pi)$ and $C_{\max}^B(\pi^*) \leq C_{\max}^B(\pi)$. Consequently, $\pi^* = (\pi^*(k_A, n_B), \pi^{*A}(k_A^+))$ is also optimal for Problem 6.

Similarly, if B_m is a B-batch, we can show that $(\pi^*(n_A, k_B), \pi^{*B}(k_B^+))$ is optimal for Problem 6 for some job index k_B with $0 \leq k_B \leq n_B - 1$. The lemma follows. □

To solve Problem 6, from Lemma 5.2, we can first generate the n schedules $\pi^*(k_A, n_B)$ with $0 \leq k_A \leq n_A - 1$ and $\pi^*(n_A, k_B)$ with $0 \leq k_B \leq n_B - 1$ in $O(n_A n_B n)$ time. Next, from Theorem 2.1, the problem in (3) for a set of n jobs is solvable in $O(n \log n)$ time. Then the n schedules $\pi^{*A}(k_A^+)$ with $0 \leq k_A \leq n_A - 1$ and $\pi^{*B}(k_B^+)$ with $0 \leq k_B \leq n_B - 1$ can be obtained in $O(n^2 \log n)$ time. Consequently, the n schedules in Lemma 5.3 can be obtained in $O(n^3)$ time, from which we pick the best feasible one for solving Problem 6. This leads to the following result.

Theorem 5.1. *Problem 6 is solvable in $O(n^3)$ time.*

6 Conclusion

We re-visit the problem of two-agent scheduling on a parallel-batch machine to minimize the makespan, where the jobs have release dates and time-dependent proportional-linear deteriorating processing times. We correct four invalid results in the literature and present some new results. We summarize in Table 2 the known algorithms and complexity of the two-agent scheduling problem in (2), i.e.,

$$1|\text{p-batch}, p_j^X = \alpha_j^X(a+bt), r_j^X, c, CF/IF|C_{\max}^A : C_{\max}^B \leq Q_B.$$

Table 2: The latest results for the problem in (2).

Scheduling Problem	Algorithm/Complexity	Reference
$c = \infty,\ CF,\ r_j^X$	$O(n^2 \log n)$	Theorem 3.1
$c < n,\ IF/CF,\ r^X, n_A = 1$	NP-hard	Theorem 4.1
$c < n,\ IF/CF,\ agr(r_j, \alpha_j)$	NP-hard	Theorem 4.1
$c < n,\ IF/CF,\ revagr(r_j, \alpha_j)$	NP-hard	Theorem 4.1
$c < n,\ IF,\ r^X$	$O(nM^2)$	Theorem 4.2
$c = \infty,\ IF,\ r_j^X$	$O(n^3)$	Theorem 5.1

For further research, it is worth studying the Pareto-optimization versions of Problems 1 and 6, i.e., $1|\text{p-batch}, p_j^X = \alpha_j^X(a+bt), r_j^X, c = \infty, CF/IF|^{\#}(C_{\max}^A, C_{\max}^B)$. As for the NP-hard problems $1|\text{p-batch}, p_j^X = \alpha_j^X(a+bt), r_j^X, c < n, IF/CF|C_{\max}^A : C_{\max}^B \leq Q_B$, heuristics with good performances are expected, and the algorithms in Tang et al. (2017) can act as possible choices. In addition, we believe that a more refined research can lead to a pseudo-polynomial-time algorithm for the $(c < n,\ CF,\ r^X)$ case of the problem , i.e., $1|\text{p-batch}, p_j^X = \alpha_j^X(a+bt), r^X, c < n, CF|C_{\max}^A : C_{\max}^B \leq Q_B$.

Acknowledgments

The authors would like to thank the associate editor and three anonymous referees for their constructive comments and helpful suggestions. This research was supported by NSFC

(11671368), NSFC (11771406), and NSFC (11571323). It was also supported in part by the Research Grants Council of Hong Kong under grant number PolyU 152207/17E.

References

Agnetis, A., Billaut, J.C., Gawiejnowicz, S., Pacciarelli, D., & Soukhal, A. (2014) *Multiagent Scheduling: Models and Algorithms*. Berlin Heidelberg: Springer.

Agnetis, A., Mirchandani, P.B., Pacciarelli, D., & Pacifici, A. (2004) Scheduling problems with two competing agents. *Operations Research*, 52, 229-242.

Baker, K.R. & Smith, J.C. (2003) A multiple-criterion model for machine scheduling. *Journal of Scheduling*, 6, 7-16.

Brucker, P., Gladky, A., Hoogeveen, J.A., Kovalyov, M.Y., Potts, C.N., Tautenhahn, T., & van de Velde, S.L. (1998) Scheduling a batching machine. *Journal of Scheduling*, 1, 31-54.

Fan, B.Q. & Cheng, T.C.E. (2016) Two-agent scheduling in a flowshop. *European Journal of Operational Research*, 252, 376-384.

Fan, B.Q., Cheng, T.C.E., Li, S.S., & Feng, Q. (2013) Bounded parallel-batching scheduling with two competing agents. *Journal of Scheduling*, 16, 261-271.

Garey, M.R. & Johnson, D.S. (1979) *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, New York.

Leung, J.Y.T., Pinedo, M.L., & Wan, G. (2010) Competitive two agent scheduling and its applications. *Operations Research*, 58, 458-469.

Lee, C.Y., Uzsoy, R., & Martin-Vega, L.A. (1992) Efficient algorithms for scheduling semiconductor burn-in operations. *Operations Research*, 40, 764-775.

Li, S.S., Ng, C.T., Cheng, T.C.E., & Yuan, J.J. (2011) Parallel-batch scheduling of deteriorating jobs with release dates to minimize the makespan. *European Journal of Operational Research*, 210, 482-488.

Li, S.S. & Yuan, J.J. (2012) Unbounded parallel-batching scheduling with two competitive agents. *Journal of Scheduling*, 15, 629-640.

Melnikov, O.I. & Shafransky, Y.M. (1979) Parametric problem of scheduling theory. *Kibernetika* (in Russian), 3, 55-57.

Ng, C.T., Barketau, M.S., Cheng, T.C.E., & Kovalyov, M.Y. (2010) "Product Partition" and related problems of scheduling and systems reliability: Computational complexity and approximation. *European Journal of Operational Research*, 207, 601-604.

Perez-Gonzalez, P. & Framinan, J.M. (2014) A common framework and taxonomy for multicriteria scheduling problem with interfering and competing jobs: Multi-agent scheduling problems. *European Journal of Operational Research*, 235, 1-16.

Poon, C.K. & Zhang, P.X. (2004) Minimizing makespan in batch machine scheduling. *Algorithmica*, 39, 155-174.

Tang, L.X., Zhao, X.L., Liu, J.Y., & Leung, J.Y.T. (2017) Competitive two-agent scheduling with deteriorating jobs on a single parallel-batching machine. *European Journal of Operational Research*, 263, 401-411.

Wang, J.Q., Fan, G.Q., Zhang, Y.Q., Zhang, C.W., & Leung, J.Y.T. (2017) Two-agent scheduling on a single parallel-batching machine with equal processing time and non-identical job sizes. *European Journal of Operational Research*, 258, 478-490.

Yuan, J.J. (2016) Complexities of some problems on multi-agent scheduling on a single machine. *Journal of the Operations Research Society of China*, 4, 379-384.