Operational Research (2023) 23:58
https://doi.org/10.1007/s12351-023-00799-1

ORIGINAL PAPER

Bicriteria fabrication scheduling of two-component jobs on a single machine

Yijie Li[1]

Received: 6 July 2022 / Revised: 13 December 2022 / Accepted: 11 September 2023 /
Published online: 3 October 2023

Abstract
This paper studies the bicriteria problem of non-preemptively scheduling n jobs, each of which is associated with a due date and comprises a standard and a specific component, on a single fabrication machine to minimize makespan and maximum lateness simultaneously. The specific components are processed individually and the standard components are grouped into batches for processing. A setup time is required before each batch of standard components is processed. A standard component is available (i.e., ready for delivery to the next production stage) only when the batch it belongs to is totally completed, whereas a specific component is available on completion of its processing. The completion time of a job is defined as the moment when both its two components have been processed and are available. An $O(n^2 \log n)$ -time algorithm with linear memory requirements is presented which can generate all Pareto optimal points and find a corresponding Pareto optimal schedule for each Pareto optimal point.

Keywords Scheduling · Batch processing · Bicriteria optimization · Two-component jobs · Maximum lateness · Single machine

Mathematics Subject Classification 90B35 · 68Q25

1 Introduction

In this paper, we study a scheduling problem related to the two-component jobs. Let $\mathcal{J} = \{J_1, J_2, \dots, J_n\}$ be a set of n jobs, to be non-preemptively processed on a single machine. Each job J_j has two components: a *standard component* $O_j^{(1)}$ and a *specific component* $O_j^{(2)}$. The two components of J_j have processing times $p_j^{(1)}$ and $p_j^{(2)}$, respec-

✉ Yijie Li
yjliyt@hotmail.com

1 School of Computer Science, University of St Andrews, St Katharine's West, St Andrews, Fife KY16 9SX, Scotland, UK

tively. On the machine, the specific components are processed individually, and the standard components are processed in batches with a *setup time* s incurred for each batch. (Since setups for specific components depend on the particular jobs, the processing times of the specific components are assumed to include the relevant setup times.) We assume *batch availability* of the standard components (Santos and Magazine 1985) in this paper. That is, a standard component is available (i.e., ready for delivery to the next production stage) only when the batch it belongs to is completed. On the other hand, a specific component is available when its processing is completed. A job is completed when both its standard and specific components have been processed and are available.

The problem of scheduling two-component jobs is introduced in Baker (1988) and has a variety of applications (Baker 1988; Yang 2004). Consider a production environment where a fabrication machine manufactures two-component products (jobs). Usually there is an assembly machine which subsequently assembles the manufactured components into end products. Since the components are all processed on the fabrication machine, this machine is a bottleneck facility in the production process. Therefore, as in most studies, we assume that the assembly stage is not a bottleneck and can be ignored.

In the area of production management, timeliness of product delivery is usually crucial to meeting market demands and customer satisfaction. As a result, due date constraints are introduced and extensively studied (Brucker 2007). In this paper, each job J_j is associated with a due date d_j, representing the interest of a decision-maker, e.g., the completion time expected by the consumer or the producer. Given a schedule, let C_j and $L_j = C_j - d_j$ denote the completion time and lateness of J_j in this schedule. Let $C_{\max} = \max_j\{C_j\}$ and $L_{\max} = \max_j\{L_j\}$ denote the makespan and maximum lateness of the schedule.

The problem studied in this paper is to simultaneously minimize makespan $C_{\max}$ and maximum lateness $L_{\max}$ by enumerating all Pareto optimal points and finding a corresponding Pareto optimal schedule for each Pareto optimal point. Following (Brucker 2007; Hoogeveen 2005; T'Kindt and Billaut 2006), it can be denoted by $1|(p_j^{(1)}, p_j^{(2)}), batch|(C_{\max}, L_{\max})$, where "$(p_j^{(1)}, p_j^{(2)})$" means "two-component jobs", and "*batch*" means "batch availability".

A feasible schedule σ is *Pareto optimal* with respect to $C_{\max}$ and $L_{\max}$ if there is no feasible schedule σ' such that $C_{\max}(\sigma') \le C_{\max}(\sigma)$ and $L_{\max}(\sigma') \le L_{\max}(\sigma)$, where at least one of the inequalities is strict. When σ is Pareto optimal, the objective vector $(C_{\max}(\sigma), L_{\max}(\sigma))$ is called a *Pareto optimal point* (Hoogeveen 2005).

The paper is organized as follows. In Sect. 2, we provide a literature review. In Sect. 3, an $O(n^3)$-time algorithm for $1|(p_j^{(1)}, p_j^{(2)}), batch|(C_{\max}, L_{\max})$ is presented. In Sect. 4, the time complexity of the obtained algorithm is improved to $O(n^2 \log n)$. Finally, some concluding remarks are drawn in Sect. 5.

2 Literature review

Multicriteria scheduling has been extensively studied in the last decades. The fundamental notations, principles and methodologies can be found in Hoogeveen (2005), T'Kindt and Billaut (2006), Herzel et al. (2021). Scheduling with batching and/or with setups has also been widely studied in the literature. We refer the reader to the surveys (Potts and Kovalyov 2000; Allahverdi et al. 2008). We next only mention the results on fabrication scheduling of two-component jobs.

Baker (1988) initiated problem $1|(p_j^{(1)}, p_j^{(2)}), batch|\sum C_j$ (minimizing total completion time of the jobs). Under the agreeability assumption (i.e., the jobs can be ordered such that the standard and specific components both appear in non-decreasing order of their processing times), he designed an $O(n^2)$-time dynamic programming algorithm. Coffman et al. (1990) further improved the complexity to $O(n \log n)$. Aneja and Singh (1990) later extended Baker's work to a more general case, where each job has a specific component and M standard components (each requiring a separate setup). Gerodimos et al. (2000) presented an $O(n^2)$-time dynamic programming algorithm for $1|(p_j^{(1)}, p_j^{(2)}), batch|L_{\max}$. They also proved that $1|(p_j^{(1)}, p_j^{(2)}), batch|\sum U_j$ (minimizing total number of late jobs) is NP-hard and gave a pseudo-polynomial time dynamic programming algorithm for it, where $U_j = 1$ if $C_j > d_j$ and $U_j = 0$ otherwise. Moreover, for $1|(p_j^{(1)} = p, p_j^{(2)}), batch|\sum U_j$ (all standard components have equal processing times), they obtained an $O(n^4 \log n)$-time dynamic programming algorithm. Wagelmans and Gerodimos (2000) gave an improved algorithm for $1|(p_j^{(1)}, p_j^{(2)}), batch|L_{\max}$ which runs in $O(n \log n)$ time.

Vickson et al. (1993) studied problem $1|(p_j^{(1)}, p_j^{(2)}), item|\sum C_j$(*item availability* of the standard components: a standard component is available immediately after the completion of its processing). Under the assumption that all standard components have equal processing times, they presented an $O(n^3)$-time algorithm. Rana and Singh (1994) presented a scheme for $1|(p_j^{(1)}, p_j^{(2)}), item|Lex(\sum C_j, C_{\max}, HC)$ which can obtain all optimal scheduling alternatives efficiently with minimization of total completion time, makespan, and total holding cost (HC denotes the total holding cost of all jobs) being primary, secondary and tertiary criteria respectively. Lin (2002) presented an $O(n^5)$-time dynamic programming algorithm for $1|(p_j^{(1)}, p_j^{(2)}), item|T_{\max}$, where $T_{\max} = \max_j\{\max\{C_j - d_j, 0\}\}$ denotes the maximum tardiness of all jobs. He also showed NP-hardness proof and designed a pseudo-polynomial time dynamic programming algorithm for $1|(p_j^{(1)}, p_j^{(2)}), item|\sum U_j$. Gerodimos et al. (2001) studied problems $1|(p_j^{(1)}, p_j^{(2)}), item|\sum C_j$, $1|(p_j^{(1)}, p_j^{(2)}), item|L_{\max}$ and $1|(p_j^{(1)}, p_j^{(2)}), item|\sum U_j$. They gave an $O(n \log n)$-time algorithm for $1|(p_j^{(1)}, p_j^{(2)}), item|\sum C_j$ under the agreeability assumption, and an $O(n^2)$-time algorithm for $1|(p_j^{(1)}, p_j^{(2)}), item|L_{\max}$. They also proved that $1|(p_j^{(1)}, p_j^{(2)}), item|\sum U_j$ is NP-hard and gave a pseudo-polynomial time dynamic programming algorithm for it. Moreover, for $1|(p_j^{(1)} = p, p_j^{(2)}), item|\sum U_j$, they obtained an $O(n^5 \log n \log P)$-time dynamic programming algorithm, where P is the sum of processing times of all specific components. Wagelmans and Gerodimos (2000) gave an improved algorithm for $1|(p_j^{(1)}, p_j^{(2)}), item|L_{\max}$ which runs in $O(n \log n)$ time.

3 The algorithm

In this section we will present an $O(n^3)$-time algorithm for $1|(p_j^{(1)}, p_j^{(2)}), batch|(C_{\max}, L_{\max})$.

From now on, we assume that all jobs have been reindexed according to EDD (earliest due date) rule such that $d_1 \le d_2 \le \cdots \le d_n$.

Since $C_{\max}$ and $L_{\max}$ are regular functions (i.e., non-decreasing functions of the job completion times), following (Gerodimos et al. 2000), we get:

Lemma 1 *For each Pareto optimal point of* $1|(p_j^{(1)}, p_j^{(2)}), batch|(C_{\max}, L_{\max})$, *there exists a corresponding Pareto optimal schedule in which the standard component of each job appears in the batch immediately preceding the specific component.*

Proof Consider a Pareto optimal schedule. If there is a job J_j such that its specific component $O_j^{(2)}$ precedes the standard component $O_j^{(1)}$, then we move $O_j^{(2)}$ to be the first specific component after the batch containing $O_j^{(1)}$. Clearly, the two objective function values $C_{\max}$ and $L_{\max}$ keep unchanged. Therefore, the modified schedule is also Pareto optimal. Repetition of this argument shows that for each Pareto optimal point, there is a corresponding Pareto optimal schedule in which the standard component of each job precedes its specific component.

Consider such a Pareto optimal schedule. If there are other batches of standard components processed between the batch containing $O_j^{(1)}$ and $O_j^{(2)}$, then we move $O_j^{(1)}$ into the last batch of standard components preceding $O_j^{(2)}$. Clearly, the modified schedule is also Pareto optimal. Repetition of this argument shows that for each Pareto optimal point, there is a corresponding Pareto optimal schedule in which the standard component of each job appears in the batch immediately preceding the specific component. □

Lemma 2 *For each Pareto optimal point of* $1|(p_j^{(1)}, p_j^{(2)}), batch|(C_{\max}, L_{\max})$, *there exists a corresponding Pareto optimal schedule in which the jobs are sequenced in EDD order.*

Proof Consider a Pareto optimal schedule described in Lemma 1. If there are two consecutive jobs J_j and J_k such that $d_j < d_k$ but the components of J_k precedes those of J_j in the schedule, then we move the components of J_k immediately after the relevant components of J_j (the standard components of J_j and J_k are thus in the same batch). Only J_k is completed later as a result of the move. Since $d_j < d_k$, the maximum lateness of J_j and J_k is no-worse in the modified schedule. Clearly, the modified schedule is also Pareto optimal. Repetition of this argument shows that for each Pareto optimal point, there is a corresponding Pareto optimal schedule in which the jobs are sequenced in EDD order. □

The above two lemmas reveal the structure of Pareto optimal schedules we are searching for. By the two lemmas, a feasible schedule can be simply represented as a sequence of *job-blocks* $F_1, F_2, \dots, F_n$, where job-block $F_i = \{J_{j_i}, J_{j_i+1}, \dots, J_{j_{i+1}-1}\}$ consists of the jobs in the i-th batch of the schedule, $i = 1, 2, \dots, n$, $1 = j_1 < j_2 < \cdots < j_{n+1} = n+1$. The last l job-blocks $F_{n-l+1}, F_{n-l+2}, \dots, F_n$ are nonempty, and the first $n-l$ job-blocks $F_1, F_2, \dots, F_{n-l}$ are empty.

The last job $J_{j_{i+1}-1}$ of F_i is called the *decision job* of this job-block. Clearly, a schedule is specified by the decision jobs of its job-blocks. Further, the sub-schedule for the jobs in F_i can be represented as $\alpha_i\beta_i$, where α_i is a *batch* consisting of the standard components of the jobs in F_i, and β_i is a *specific-block* consisting of the specific components of the jobs in F_i, $i = 1, 2, \dots, n$.

In the schedule, the standard components in each batch and the specific components in each specific-block are processed in EDD order respectively. Let $p(\alpha_i) = \sum_{j \in F_i} p_j^{(1)}$ and $p(\beta_i) = \sum_{j \in F_i} p_j^{(2)}$ denote the processing times of the batch and the specific-block of F_i, respectively. Let $p(F_i) = p(\alpha_i) + p(\beta_i)$ denote the processing time of F_i. Moreover, the processing time of any empty job-block is zero.

Let $s(\alpha_i)$ denote the setup time of batch α_i, which is s if α_i is nonempty, and 0 otherwise. Let $S(F_i)$ and $C(F_i)$ denote the start time and completion time of F_i, respectively. We have: $C(F_i) = S(F_i) + p(F_i)$.

Since both criteria $C_{\max}$ and $L_{\max}$ are regular, we can consider only the schedules without idle times. Therefore, we have:

Lemma 3 *In a feasible schedule* $\sigma = (F_1, F_2, \dots, F_n)$, $S(F_1) = s(\alpha_1)$, $S(F_i) = C(F_{i-1}) + s(\alpha_i)$, $i = 2, \dots, n$.

In Algorithm EDD-MoveLeft, to search for the Pareto optimal schedules for constructing Pareto set $\Omega(\mathcal{J})$, we will apply the following well-known generic approach for multicriteria scheduling (Hoogeveen 2005).

Lemma 4 (Hoogeveen 2005) *Let* y *be the optimal value of problem* $\alpha|f \le \hat{x}|g$, *and let* x *be the optimal value of problem* $\alpha|g \le y|f$. *Then* (x, y) *is a Pareto optimal point for problem* $\alpha||(f, g)$.

Let $\Pi(\mathcal{J})$ denote the set of all feasible schedules for $\mathcal{J}$. We concentrate our attention to the schedules in $\Pi(\mathcal{J})$ which have the properties described in Lemmas 1, 2 and 3. Let $\Pi(\mathcal{J}, y)$ denote the set of the schedules in $\Pi(\mathcal{J})$ with $L_{\max}$ less than y. We have $\Pi(\mathcal{J}, +\infty) = \Pi(\mathcal{J})$.

In the forthcoming algorithm, the cyclic queue data structure will be used. Please refer to (Cormen et al. 2009) for cyclic queue and its elementary operations DEQUEUE and ENQUEUE.

Algorithm 1 (Algorithm EDD-MoveLeft)

```
Input: The set of jobs $\mathcal{J}$ and the fabrication machine.
Output: The Pareto set $\Omega(\mathcal{J})$ for $C_{\max}$ and $L_{\max}$ which consists of all Pareto optimal points together with the corresponding schedules.
Step 1. Initially, $\Omega(\mathcal{J}) \leftarrow \varnothing$, $k \leftarrow 0$, $h \leftarrow 0$, $y^{(h)} \leftarrow +\infty$. Let $\sigma^{(h)} = (F_1^{(h)}, F_2^{(h)}, \ldots, F_n^{(h)}) \leftarrow (\varnothing, \varnothing, \ldots, \mathcal{J})$, where the jobs in $F_i^{(h)}$ are stored in a cyclic queue in EDD order, $i = 1, 2, \ldots, n$. Compute for each job $J_j$ its lateness $L_j^{(h)}$ in $\sigma^{(h)}$ (by Lemma 3).
Step 2. The $(h+1)$-th iteration:
    $y^{(h+1)} \leftarrow L_{\max}(\sigma^{(h)})$. Adjust $\sigma^{(h)}$ to construct $\sigma^{(h+1)} = (F_1^{(h+1)}, F_2^{(h+1)}, \ldots, F_n^{(h+1)})$ as follows:
    Step 2.1. For $i = n, n-1, \ldots, 1$, check the inequality $L_j^{(h)} < y^{(h+1)}$ for each job $J_j$ in $F_i^{(h)}$. The jobs in $F_i^{(h)}$ are checked backwardly, starting at the rear of the queue. If the inequality does not hold for $J_k$ (i.e., $d_k = \max\{d_j | \ J_j \in F_i^{(h)} \wedge L_j^{(h)} \geq y^{(h+1)}\}$; $J_k$ is moved truly due to the inequality violation), if $i = 1$ or $J_k$ has the largest due date in $F_i^{(h)}$, then $\sigma^{(h+1)} \leftarrow \varnothing$ and go to Step 4. Otherwise (DEQUEUE) remove all the jobs with due dates no more than $d_k$ in $F_i^{(h)}$ and (ENQUEUE) insert these jobs into $F_{i-1}^{(h)}$ (by Lemma 2). Update for each job its lateness in the schedule.
    Step 2.2. Repeat Step 2.1 until there is no inequality violation in modified $\sigma^{(h)}$. Let $\sigma^{(h+1)}$ be the modified $\sigma^{(h)}$. If $C_{\max}(\sigma^{(h+1)}) > C_{\max}(\sigma^{(h)})$, then $k \leftarrow k+1$, $\pi_k \leftarrow \sigma^{(h)}$, $\Omega(\mathcal{J}) \leftarrow \Omega(\mathcal{J}) \cup \{(C_{\max}(\pi_k), L_{\max}(\pi_k), \pi_k)\}$.
Step 3. $h \leftarrow h+1$. Go to Step 2.
Step 4. $\pi^* \leftarrow \sigma^{(h)}$, $\Omega(\mathcal{J}) \leftarrow \Omega(\mathcal{J}) \cup \{(C_{\max}(\pi^*), L_{\max}(\pi^*), \pi^*)\}$. Return $\Omega(\mathcal{J})$.
```

Remark 1 About Step 1: In schedule $\sigma^{(h)}$, for $i = 1, 2, \ldots, n$, the standard components of the jobs in $F_i^{(h)}$ are first processed as a batch in EDD order, and the specific components of the jobs in $F_i^{(h)}$ are then processed individually in EDD order (by Lemma 1).

Remark 2 About Step 2.1: When the rightmost, i.e., the last, empty job-block becomes non-empty, the lateness values of all jobs need to be updated. Otherwise, only the lateness values of the jobs in $F_i^{(h)}$ and $F_{i-1}^{(h)}$ need to be updated.

Step 1 of Algorithm EDD-MoveLeft can be implemented in $O(n \log n)$ time. Step 2 can be done in $O(n)$ time for each inequality violation adjustment. Steps 3 and 4 can be done in $O(1)$ time in each iteration.

In each inequality violation adjustment, there is at least one job which has to be moved to the left. Actually, it is worth stressing that in Algorithm EDD-MoveLeft we never move a job to the right. Later, we will prove that any job can only be moved to the left; no job can be moved to the right (Lemma 6). Hence, the total number of inequality violation adjustments is $O(n^2)$. The running time of Algorithm EDD-MoveLeft is $O(n^3)$.

Given a schedule $\sigma = (F_1, F_2, \dots, F_n)$, for any job $J_j \in \mathcal{J}$ let $O(j, \sigma) = i$ denote the ordinal number of job J_j if $J_j \in F_i$ in σ. We have:

Lemma 5 *Let $\sigma^{(h)} = (F_1^{(h)}, F_2^{(h)}, \dots, F_n^{(h)})$ be the schedule obtained at iteration h $(h = 0, 1, \dots)$ of Algorithm EDD-MoveLeft, where the last l_h job-blocks $F_{n-l_h+1}^{(h)}, F_{n-l_h+2}^{(h)}, \dots, F_n^{(h)}$ are nonempty. Let $\sigma = (F_1, F_2, \dots, F_n)$ be any feasible schedule in $\Pi\big(\mathcal{J}, y^{(h)}\big)$, where the last l job-blocks $F_{n-l+1}, F_{n-l+2}, \dots, F_n$ are nonempty. Then the following properties hold*:

(1) $l_h \le l$;
(2) $S(F_i^{(h)}) \le S(F_i), i = 1, 2, \dots, n$;
(3) $C(F_i^{(h)}) \le C(F_i), i = 1, 2, \dots, n$;
(4) $\forall J_j \in \mathcal{J}, O(j, \sigma^{(h)}) \ge O(j, \sigma)$;
(5) $\forall J_j \in \mathcal{J}$, *if* $O(j, \sigma^{(h)}) = O(j, \sigma)$, *then* $C_j(\sigma^{(h)}) \le C_j(\sigma)$.

Proof We prove the lemma by induction on h.

The base case trivially holds, which can be checked by comparing $\sigma^{(0)} = (\emptyset, \emptyset, \dots, \mathcal{J})$ with any schedule in $\Pi\big(\mathcal{J}, y^{(0)}\big) = \Pi(\mathcal{J})$.

Now assume that for $\sigma^{(h)}$ and any schedule in $\Pi\big(\mathcal{J}, y^{(h)}\big)$, the lemma holds. We consider $\sigma^{(h+1)}$ and any schedule $\sigma \in \Pi\big(\mathcal{J}, y^{(h+1)}\big)$. Since $y^{(h+1)} < y^{(h)}$, $\sigma \in \Pi\big(\mathcal{J}, y^{(h)}\big)$. By the inductive assumption, the lemma holds for $\sigma^{(h)}$ and σ.

Assume that there is a job $J_j \in \bigcup_{a=i+1}^{n} F_a^{(h)}$ but $J_j \in F_i^{(h+1)}$. Then we can find a truly moved job $J_k \in F_{i+1}^{(h)}$ in (modified) $\sigma^{(h)}$ which violates its inequality $L_k^{(h)} < y^{(h+1)}$. We know that J_j and J_k are in the same job-block $F_{i+1}^{(h)}$ in (modified) $\sigma^{(h)}$ and J_j precedes J_k. If $O(k, \sigma) = i + 1$, since $L_k^{(h)} \ge y^{(h+1)}$ and $C_k(\sigma^{(h)}) \le C_k(\sigma)$ (by the inductive assumption, the fifth Property), we have: $L_k(\sigma) \ge L_k^{(h)} \ge y^{(h+1)}$. If $O(k, \sigma) > i + 1$, certainly we also have: $L_k(\sigma) \ge L_k^{(h)} \ge y^{(h+1)}$. Hence, it must be true that $J_k \notin \bigcup_{a=i+1}^{n} F_a$ in σ, i.e., $O(k, \sigma) \le i$. We get: $O(j, \sigma) \le i = O(j, \sigma^{(h+1)})$. By the principle of induction, we have proved the fourth property of the lemma.

In fact, we have shown that for $i = n, n-1, \dots, 1$, $\bigcup_{a=i}^{n} F_a \subseteq \bigcup_{a=i}^{n} F_a^{(h)}$ holds, $h = 0, 1, \dots$. Equivalently we get: for $i = 1, 2, \dots, n$, $\bigcup_{a=1}^{i} F_a^{(h)} \subseteq \bigcup_{a=1}^{i} F_a$. Hence the first four properties of the lemma are proved easily.

Let us continue to prove the fifth property of the lemma. Assume that there is a job J_j such that $O(j, \sigma^{(h+1)}) = O(j, \sigma) = i$. Clearly, the decision job of $F_i^{(h+1)}$ has an index λ_i^{h+1} no more than the index λ_i of the decision job of F_i. The number μ_i^{h+1} of nonempty job-blocks in $\{F_1^{(h+1)}, F_2^{(h+1)}, \dots, F_i^{(h+1)}\}$ is no more than the number μ_i of nonempty job-blocks in $\{F_1, F_2, \dots, F_i\}$. Since $C_j(\sigma^{(h+1)}) = \mu_i^{h+1} \cdot s + \sum_{q=1}^{\lambda_i^{h+1}} p_q^{(1)} + \sum_{q=1}^{j} p_q^{(2)}$ and $C_j(\sigma) = \mu_i \cdot s + \sum_{q=1}^{\lambda_i} p_q^{(1)} + \sum_{q=1}^{j} p_q^{(2)}$, we get: $C_j(\sigma^{(h+1)}) \le C_j(\sigma)$. By the principle of induction, we complete the proof of the fifth property of the lemma. □

Algorithm EDD-MoveLeft generates nonempty job-blocks every now and then. Consider the nonempty job-blocks in $\sigma^{(h)} = (F_1^{(h)}, F_2^{(h)}, \dots, F_n^{(h)})$ obtained at iteration h. In $\sigma^{(h)}$, for $i = n - l_h + 2, n - l_h + 3, \dots, n$, the decision job in $F_{i-1}^{(h)}$ cannot be included in $F_i^{(h)}$ (and all the subsequent job-blocks $F_{i+1}^{(h)}, F_{i+2}^{(h)}, \dots, F_n^{(h)}$) in any feasible schedule in $\Pi\big(\mathcal{J}, y^{(h)}\big)$, because this job is moved from the i-th job-block into $F_{i-1}^{(h)}$ truly due to the inequality violation in one of the preceding iterations. When this job is moved from the i-th job-block into the left adjacent job-block, the earlier jobs in the i-th job-block also need to be moved together with it; Otherwise one of these jobs has to be completed no earlier than the former completion time of the decision job, incurring an inequality violation since its due date is no more than that of the decision job. This means that Algorithm EDD-MoveLeft does not affect the feasibility of any schedule in $\Pi\big(\mathcal{J}, y^{(h)}\big)$. Each such job needs a separate job-block in any feasible schedule in $\Pi\big(\mathcal{J}, y^{(h)}\big)$. Algorithm EDD-MoveLeft actually assigns each such job to a separate job-block and these job-blocks are all nonempty job-blocks in $\sigma^{(h)}$. By this observation, we know that after a new nonempty job-block is generated, it will never become empty, nor will its due date (which is defined to be the largest due date of the jobs in it, i.e., the due date of its decision job) decrease, since otherwise an inequality violation will occur in its left adjacent job-block. Hence, in Step 2.1 of Algorithm EDD-MoveLeft, if J_k violates the inequality and it has the largest due date in $F_i^{(h)}$, which implies $\Pi\big(\mathcal{J}, y^{(h+1)}\big) = \emptyset$, then we simply set $\sigma^{(h+1)} = \emptyset$.

Let $\sigma^{(h)} = (F_1^{(h)}, F_2^{(h)}, \dots, F_n^{(h)})(\sigma^{(h+1)} = (F_1^{(h+1)}, F_2^{(h+1)}, \dots, F_n^{(h+1)}))$ be the schedule obtained at iteration h ($h + 1$) of Algorithm EDD-MoveLeft, where the last l_h (l_{h+1}) job-blocks are nonempty. Since $y^{(h+1)} < y^{(h)}$, $\sigma^{(h+1)} \in \Pi\big(\mathcal{J}, y^{(h)}\big)$. By Lemma 5, for $i = 1, 2, \dots, n$, the decision job of $F_i^{(h)}$ has an index λ_i^h no more than the index λ_i^{h+1} of the decision job of $F_i^{(h+1)}$. The number μ_i^h of nonempty job-blocks in $\{F_1^{(h)}, F_2^{(h)}, \dots, F_i^{(h)}\}$ is no more than the number μ_i^{h+1} of nonempty job-blocks in $\{F_1^{(h+1)}, F_2^{(h+1)}, \dots, F_i^{(h+1)}\}$. By this analysis, we can prove the following lemma.

Lemma 6 *In Algorithm EDD-MoveLeft, any job cannot be moved to the right.*

Proof We prove the lemma by contradiction.

In initial schedule $\sigma^{(0)}$, all jobs are in the rightmost job-block $F_n^{(0)} = \mathcal{J}$. Certainly, none of them can be moved to the right.

Suppose that at iteration h ($h > 0$), job $J_k \in F_i^{(h-1)}$ has to be moved to the left because $L_k^{(h-1)} \geq y^{(h)}$, but after iteration $h'(h' \geq h)$, $J_k \in F_i^{(h')}$. Since $y^{(h')} < y^{(h)}$, $\sigma^{(h')} \in \Pi(\mathcal{J}, y^{(h)})$. From the above analysis, similarly to the proof of the fifth property of Lemma 5, we get: $C_k(\sigma^{(h-1)}) \leq C_k(\sigma^{(h')})$. It follows that $L_k^{(h')} \geq y^{(h)} > y^{(h')}$. Therefore, $J_k \notin F_i^{(h')}$, a contradiction. □

By Lemma 5, we get:

Lemma 7 *Let $\sigma^{(h)}$ be the schedule obtained at iteration h ($h = 0, 1, \dots$) of Algorithm EDD-MoveLeft. If $\sigma^{(h)} = \emptyset$, then $\Pi(\mathcal{J}, y^{(h)}) = \emptyset$; Otherwise $\sigma^{(h)}$ is a schedule which has minimum makespan among all schedules in $\Pi(\mathcal{J}, y^{(h)})$.*

Proof Suppose that in Step 2.1 of Algorithm EDD-MoveLeft, we find a job J_k such that $d_k = \max\{d_j | J_j \in F_i^{(h-1)} \wedge L_j^{(h-1)} \geq y^{(h)}\}$. By Lemma 2, all the jobs with due dates no more than d_k in $F_i^{(h-1)}$ need to be moved into $F_{i-1}^{(h-1)}$. If $i = 1$, then job J_k cannot be scheduled with its lateness less than $y^{(h)}$ in any schedule in $\Pi(\mathcal{J}, y^{(h)})$, implying that $\Pi(\mathcal{J}, y^{(h)}) = \emptyset$. If J_k has the largest due date in $F_i^{(h-1)}$, then we also know that $\Pi(\mathcal{J}, y^{(h)}) = \emptyset$. The reason has been explained above (after the proof of Lemma 5). Thus, in these two cases, we simply set $\sigma^{(h)} = \emptyset$ and return.

On the other hand, if $\sigma^{(h)} \neq \emptyset$, then by property (3) of Lemma 5, $\sigma^{(h)}$ has minimum $C_{\max}$ among all schedules in $\Pi(\mathcal{J}, y^{(h)})$. □

Combining Lemmas 4 and 7, we get:

Theorem 8 *Algorithm EDD-MoveLeft solves $1|(p_j^{(1)}, p_j^{(2)}), batch|(C_{\max}, L_{\max})$ in $O(n^3)$ time. It returns Pareto set $\Omega(\mathcal{J})$ which consists of all Pareto optimal points together with the corresponding Pareto optimal schedules. The last generated schedule π^* has minimum makespan among all optimal schedules for single criterion $L_{\max}$.*

4 An $O(n^2 \log n)$-time implementation

In this section we will illustrate how to improve the time complexity of the obtained algorithm to $O(n^2 \log n)$. The basic idea is to use AVL trees and max-heaps to store the lateness values of the jobs so that it takes only $O(\log n)$ time to find a job violating its inequality. Please refer to Cormen et al. (2009) for AVL trees and max-heaps as well as their elementary operations.

During the implementation of Algorithm EDD-MoveLeft, we will maintain an array, *Aglobal*, to store the completion times of all the jobs in the current schedule. The j-th position of *Aglobal* stores the completion time of job J_j (i.e., the completion time of its specific component), $j = 1, 2, \dots, n$. Each job-block has an indicator which shows the position of its decision job in *Aglobal*. By these indicators, we immediately know the content of each job-block.

For each job-block, there is an AVL tree which stores the lateness values of its jobs. Let the lateness of the job-block be defined as the largest lateness value of its jobs. We use a max-heap, *Hglobal*, to store the lateness values of all nonempty job-blocks in the current schedule. Hence, we can extract the $L_{\max}$ value of the schedule (stored in the root of *Hglobal*) in $O(\log n)$ time, ensuring to find a job violating its inequality in $O(\log n)$ time.

Suppose that we are adjusting $\sigma^{(h)} = (F_1^{(h)}, F_2^{(h)}, \dots, F_n^{(h)})$ in Step 2.1 of Algorithm EDD-MoveLeft, and we find job $J_k \in F_i^{(h)}$ violating its inequality. We illustrate how to maintain the array, the AVL trees and the max-heap efficiently. Let $F_{i,1}^{(h)}$ denote the set of the jobs to be moved from $F_i^{(h)}$ to $F_{i-1}^{(h)}$ (i.e., $F_{i,1}^{(h)}$ is the set of all the jobs with due dates no more than d_k in $F_i^{(h)}$). Accordingly, let $\alpha_{i,1}^{(h)}$ and $\beta_{i,1}^{(h)}$ denote the standard and specific components of the jobs in $F_{i,1}^{(h)}$, respectively. Let $p(\alpha_{i,1}^{(h)}) = \sum_{O_j^{(1)} \in \alpha_{i,1}^{(h)}} p_j^{(1)}$.

Consider the following two different cases.

Case 1. $F_{i-1}^{(h)} \neq \emptyset$.

After the jobs in $F_{i,1}^{(h)}$ are moved into $F_{i-1}^{(h)}$, the completion times of the jobs in $F_{i-1}^{(h)}$ (before its update), and thus their lateness values, will increase by $p(\alpha_{i,1}^{(h)})$. The completion times and lateness values of the jobs in $F_{i,1}^{(h)}$ will also change. All the other jobs keep their completion times and lateness values unchanged.

It is easy to update the lateness values of the jobs in $F_{i-1}^{(h)}$. We just add a new field called *increment*, denoted by $inc(F_{i-1}^{(h)})$, in the AVL tree $T(F_{i-1}^{(h)})$ which stores the lateness values of the jobs in $F_{i-1}^{(h)}$. The initial value of the increment is zero. At this moment, we set $inc(F_{i-1}^{(h)}) = inc(F_{i-1}^{(h)}) + p(\alpha_{i,1}^{(h)})$. With the help of $inc(F_{i-1}^{(h)})$, we can lazily update the lateness values of the jobs in $F_{i-1}^{(h)}$ which are stored in $T(F_{i-1}^{(h)})$; we do not change these values at all. When a value is extracted from $T(F_{i-1}^{(h)})$ in $O(\log n)$ time, we can get the true lateness by simply adding the amount $inc(F_{i-1}^{(h)})$ to the extracted value.

It is also easy to update the lateness values of the jobs in $F_{i,1}^{(h)}$, since we know the completion time of $F_{i-1}^{(h)}$, $C(F_{i-1}^{(h)})$, which is equal to the completion time of the decision job of $F_{i-1}^{(h)}$ stored in *Aglobal*. At this moment, we update $C(F_{i-1}^{(h)})$ to be $C(F_{i-1}^{(h)}) + inc(F_{i-1}^{(h)})$. Suppose that the jobs in $F_{i,1}^{(h)}$ are $J_{i_1}, J_{i_1+1}, \dots, J_{i_1+x}$. Then, their lateness values are $C(F_{i-1}^{(h)}) + p_{i_1}^{(2)} - d_{i_1}, C(F_{i-1}^{(h)}) + p_{i_1}^{(2)} + p_{i_1+1}^{(2)} - d_{i_1+1}, \dots, C(F_{i-1}^{(h)}) + p_{i_1}^{(2)} + p_{i_1+1}^{(2)} + \cdots + p_{i_1+x}^{(2)} - d_{i_1+x}$. We remove from $T(F_i^{(h)})$ the old lateness values of the jobs in $F_{i,1}^{(h)}$, and insert their new lateness values minus $inc(F_{i-1}^{(h)})$ into $T(F_{i-1}^{(h)})$. Update $F_i^{(h)}$ to be $F_i^{(h)} \backslash F_{i,1}^{(h)}$. Update $F_{i-1}^{(h)}$ to be $F_{i-1}^{(h)} \cup F_{i,1}^{(h)}$.

Update *Hglobal* accordingly. Only the lateness values of $F_i^{(h)}$ and $F_{i-1}^{(h)}$ in *Hglobal* need to be updated. The new lateness value of $F_i^{(h)}$ ($F_{i-1}^{(h)}$, resp.) is equal to the maximum value stored in $T(F_i^{(h)})$ ($T(F_{i-1}^{(h)})$, resp.) plus the corresponding increment.

Since moving a job to the left once can be accomplished in $O(\log n)$ time and there are $O(n^2)$ movements, it takes $O(n^2 \log n)$ time to deal with this case.

Case 2. $F_{i-1}^{(h)} = \emptyset$.

After the jobs in $F_{i,1}^{(h)}$ are moved into $F_{i-1}^{(h)}$, the completion times of all the jobs will change, since there is a necessary setup time s in $F_{i-1}^{(h)}$. Thus, we recalculate the completion times of all the jobs and store them in $Aglobal$. We build an AVL tree for storing the lateness values of the jobs in $F_{i-1}^{(h)}$ in $O(n \log n)$ time. Update $inc(F_q^{(h)})$ to be its old value plus s, $q = i, i+1, \ldots, n$. Update $Hglobal$ accordingly. The lateness values of all nonempty job-blocks need to be updated. Since this case occurs at most $n-1$ times, it takes $O(n^2 \log n)$ time to deal with it.

The array $Aglobal$ and max-heap $Hglobal$ have sizes of n. The AVL trees have n nodes in total. Hence, we get:

Theorem 9 *A careful implementation of Algorithm EDD-MoveLeft leads to an $O(n^2 \log n)$-time algorithm for $1|(p_j^{(1)}, p_j^{(2)}), batch|(C_{\max}, L_{\max})$ with $O(n)$ memory requirements.*

5 Conclusions

In this paper we studied the bicriteria problem of scheduling jobs with due dates and two components on a single fabrication machine to minimize makespan and maximum lateness simultaneously, under the assumption of batch availability of the standard components. We presented an $O(n^2 \log n)$-time algorithm with linear memory requirements which can generate all Pareto optimal points and find a corresponding Pareto optimal schedule for each Pareto optimal point. Note that there are several recent papers on Pareto optimization batch scheduling which dealt with makespan and maximum cost (Geng et al. 2018; He et al. 2020, 2022; Gao 2022; Gao et al. 2022). These research papers discussed the problems (such as serial batch or parallel batch) which are different from that studied in this paper. Since maximum lateness is a special case of maximum cost, for future research, a vexing problem is to consider Pareto optimization scheduling of two-component jobs for maximum cost instead of makespan, in combination with maximum cost or a general min-sum objective function.

Declarations

Conflict of interest The authors certify that they have no affiliations with or involvement in any organization or entity with any financial interest or non-financial interest in the subject matter or materials discussed in this manuscript.

References

Allahverdi A, Ng CT, Cheng TE, Kovalyov MY (2008) A survey of scheduling problems with setup times or costs. Eur J Oper Res 187(3):985–1032

Aneja Y, Singh N (1990) Scheduling production of common components at a single facility. IIE Trans 22(3):234–237

Baker KR (1988) Scheduling the production of components at a common facility. IIE Trans 20(1):32–35

Brucker P (2007) Scheduling algorithms, 5th edn. Springer, Berlin

Coffman E, Yannakakis M, Magazine M, Santos C (1990) Batch sizing and job sequencing on a single machine. Ann Oper Res 26(1):135–147

Cormen TH, Leiserson CE, Rivest RL, Stein C (2009) Introduction to algorithms, 3rd edn. MIT press, Cambridge

Gao Y (2022) Min-max scheduling of batch or drop-line jobs under agreeable release and processing times. Asia-Pacific J Oper Res 39(02):2150023

Gao Y, Yuan J, Ng C, Cheng T (2022) Pareto-scheduling with family jobs or ND-agent on a parallel-batch machine to minimize the makespan and maximum cost. 4OR 20(2):273–287

Geng Z, Yuan J, Yuan J (2018) Scheduling with or without precedence relations on a serial-batch machine to minimize makespan and maximum cost. Appl Math Comput 332:1–18

Gerodimos AE, Glass CA, Potts CN (2000) Scheduling the production of two-component jobs on a single machine. Eur J Oper Res 120(2):250–259

Gerodimos A, Glass C, Potts C (2001) Scheduling of customized jobs on a single machine under item availability. IIE Trans 33(11):975–984

He C, Xu C, Lin H (2020) Serial-batching scheduling with two agents to minimize makespan and maximum cost. J Sched 23(5):609–617

He C, Wu J, Lin H (2022) Two-agent bounded parallel-batching scheduling for minimizing maximum cost and makespan. Discret Optim 45:100698

Herzel A, Ruzika S, Thielen C (2021) Approximation methods for multiobjective optimization problems: A survey. INFORMS J Comput 33(4):1284–1299

Hoogeveen H (2005) Multicriteria scheduling. Eur J Oper Res 167(3):592–623

Lin BM (2002) Fabrication scheduling on a single machine with due date constraints. Eur J Oper Res 136(1):95–105

Potts CN, Kovalyov MY (2000) Scheduling with batching: a review. Eur J Oper Res 120(2):228–249

Rana S, Singh N (1994) Group scheduling jobs on a single machine: A multi-objective approach with preemptive priority structure. Eur J Oper Res 79(1):38–50

Santos C, Magazine M (1985) Batching in single operation manufacturing systems. Oper Res Lett 4(3):99–103

T'Kindt V, Billaut JC (2006) Multicriteria scheduling: theory, models and algorithms, 2nd edn. Springer, Berlin

Vickson R, Magazine M, Santos C (1993) Batching and sequencing of components at a single facility. IIE Trans 25(2):65–70

Wagelmans APM, Gerodimos AE (2000) Improved dynamic programs for some batching problems involving the maximum lateness criterion. Oper Res Lett 27(3):109–118

Yang W-H (2004) Scheduling two-component products on parallel machines. Omega 32(5):353–359