Mathematical Biosciences and Engineering

http://www.aimspress.com/journal/mbe

MBE, 19(7): 7337–7348.
DOI: 10.3934/mbe.2022346
Received: 21 March 2022
Revised: 10 May 2022
Accepted: 16 May 2022
Published: 18 May 2022

Research article

Pareto optimal algorithms for minimizing total (weighted) completion time and maximum cost on a single machine

Zhimeng Liu and Shuguang Li*

College of Computer Science and Technology, Shandong Technology and Business University, Yantai 264005, China

* **Correspondence:** Email: sgliytu@hotmail.com; Tel: +8618753509226.

Abstract: This paper studies the Pareto scheduling problem of minimizing total weighted completion time and maximum cost on a single machine. It is known that the problem is strongly NP-hard. Algorithms with running time $O(n^3)$ are presented for the following cases: arbitrary processing times, equal release dates and equal weights; equal processing times, arbitrary release dates and equal weights; equal processing times, equal release dates and arbitrary weights.

Keywords: scheduling; Pareto optimization; single machine; total weighted completion time; maximum cost

1. Introduction

In the last decades, multicriteria scheduling problems have attracted wide attention because they are more meaningful from practical point of view [1]. It is clear that unless we are extremely lucky, there will be no schedule that achieves the optimal values for all criteria simultaneously. Sometimes, a so-called optimum with respect to one criterion could perform extremely bad with respect to other criteria, which implies that we have to give in on the quality of at least one of the criteria. For example, decision makers may need to consider several criteria simultaneously such as on time delivery (related to maximum cost criterion) and work-in-process inventory (related to total weighted completion time criterion). If everything is set on keeping work-in-process inventories low, then some products are likely to be completed far beyond their due dates. On the other hand, if the goal is to keep the customers satisfied by observing due dates, then the work-in-process inventories are likely to be large. In recent years, multicriteria-based techniques and approaches have been increasingly applied in many specialized fields, such as healthcare facility location [2], unpaced mixed-model assembly lines [3], and the flexible manufacturing system using fuzzy logic [4], to name just a few.

The bicriteria scheduling problems we consider in this paper are special cases of the following problem. There are a set of n jobs $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ and a single machine which can process at most one job at a time. Each job J_j has a processing time p_j, a release date r_j and a positive weight w_j, and is associated with a cost function $f_j(\cdot)$. Assume that each $f_j(\cdot)$ is regular, i.e., $f_j(\cdot)$ is non-decreasing in the job completion times. The jobs are to be processed on the machine without preemption. A schedule specifies for each job when it is executed on the machine. Under a feasible schedule σ, the start time and completion time of J_j are denoted by $S_j(\sigma)$ and $C_j(\sigma)$, respectively, where $S_j(\sigma) \geq r_j$ and $C_j(\sigma) = S_j(\sigma) + p_j$. The scheduling cost of J_j is given by $f_j(C_j(\sigma))$. Let $\sum_{j=1}^{n} w_j C_j(\sigma)$ denote the total weighted completion time of the jobs. Let $f_{\max}(\sigma) = \max_j f_j(C_j(\sigma))$ denote the maximum cost of σ. Two important special cases of $f_{\max}$ are the makespan $C_{\max}(\sigma) = \max_j\{C_j(\sigma)\}$, and the maximum lateness $L_{\max}(\sigma) = \max_j\{C_j(\sigma) - d_j\}$, where d_j denotes the due date of J_j which indicates a preferred completion time of J_j. We omit the argument σ if there is no confusion possible as to the schedule we are referring to. The goal is to find Pareto optimal schedules which simultaneously optimize $\sum_{j=1}^{n} w_j C_j$ and $f_{\max}$.

A feasible schedule σ is Pareto optimal with respect to $\sum_{j=1}^{n} w_j C_j$ and $f_{\max}$ if there is no feasible schedule σ' such that $\sum_{j=1}^{n} w_j C_j(\sigma') \leq \sum_{j=1}^{n} w_j C_j(\sigma)$ and $f_{\max}(\sigma') \leq f_{\max}(\sigma)$, where at least one of the inequalities is strict. The objective vector $(\sum_{j=1}^{n} w_j C_j(\sigma), f_{\max}(\sigma))$ is called a Pareto optimal point [1]. Following the notation schemes of [5, 6], the problem under consideration is denoted as $1|r_j|(\sum_{j=1}^{n} w_j C_j, f_{\max})$.

Problem $1|r_j|(\sum_{j=1}^{n} w_j C_j, f_{\max})$ is strongly NP-hard [1]. Its special cases, $1||(\sum_{j=1}^{n} w_j C_j, f_{\max})$ (all jobs have the same release dates) and $1|r_j|(\sum_{j=1}^{n} C_j, f_{\max})$ (all jobs have the same weights) are also strongly NP-hard [7], due to the strong NP-hardness result by Lenstra et al. [8] for problems $1|\bar{d}_j|\sum_{j=1}^{n} w_j C_j$ and $1|r_j, \bar{d}_j|\sum_{j=1}^{n} C_j$, where $\bar{d}_j$ denotes the deadline of job J_j before which J_j must be completed in any feasible schedule.

Wassenhove and Gelders [9] presented an algorithm for $1||(\sum_{j=1}^{n} C_j, L_{\max})$ which finds each Pareto optimal point in $O(n \log n)$ time. They first characterized the set of Pareto optimal points and then gave the algorithm. John [10] extended the idea to obtain an algorithm for $1||(\sum_{j=1}^{n} C_j, f_{\max})$ which finds each Pareto optimal point in $O(n^2)$ time. He also reduced significantly the computational requirements for generating all Pareto optimal schedules by proving that a small perturbation of one Pareto optimal schedule generates its adjacent Pareto optimal schedule. Hoogeveen and van de Velde [11] proved that for $1||(\sum_{j=1}^{n} C_j, f_{\max})$ there are at most $n(n-1)/2 + 1$ Pareto optimal points. Hence, problems $1||(\sum_{j=1}^{n} C_j, L_{\max})$ and $1||(\sum_{j=1}^{n} C_j, f_{\max})$ can be solved in $O(n^3 \log n)$ and $O(n^4)$ time, respectively. Hoogeveen and van de Velde [11] also obtained a $O(n^3 \log \sum_j p_j)$-time algorithm for $1||(\sum_{j=1}^{n} C_j, f_{\max})$. Gao and Yuan [12] showed that the complexity analysis of this algorithm is invalid, and further gave a new $O(n^3 \log \sum_j p_j)$-time algorithm for $1||(\sum_{j=1}^{n} C_j, f_{\max})$. In another paper, Gao and Yuan [13] studied $1|k_j|(\sum_{j=1}^{n} C_j, f_{\max})$ and presented two $O(n^4)$-time algorithm for it, where k_j indicates that the ordinal number of J_j in the schedule is no more than k_j. He et al. [14] presented an $O(n^3 \log n)$-time algorithm for $1|p_j = p|(\sum_{j=1}^{n} w_j C_j, f_{\max})$. Steiner and Stephenson [7] studied $1||(\sum_{j=1}^{n} w_j C_j, L_{\max})$. They showed that the master sequence concept is also applicable to this problem and the master sequence implies the existence of global dominance orders for it. They then incorporated the dominance results into a new branch and bound algorithm for $1||(\sum_{j=1}^{n} w_j C_j, L_{\max})$ which can enumerate all Pareto optimal schedules for over 90% of the 1440 randomly generated problems with up to 50 jobs. Lazarev et al. [15] presented an $O(n^3 \log n)$-time algorithm for $1|r_j, p_j = p|(C_{\max}, L_{\max})$, the core idea of which is to construct a schedule in the current iteration with $L_{\max}$ value lower than in the previous iteration.

Hoogeveen [16] presented an $O(n^4)$-time algorithm for the problem of minimizing two maximum cost criteria, $1||(f_{\max}, g_{\max})$, and an $O(n^8)$-time algorithm for the problem of minimizing three maximum cost criteria, $1||(f_{\max}, g_{\max}, h_{\max})$. They also showed that the algorithms can be used if precedence constraints exist between the jobs or if all penalty functions are non-increasing in the job completion times.

In many manufacturing systems, the products may have similar designs or characteristics (corresponding to similar job data, e.g., processing times, release dates, or weights). In standardized systems in practice, the products consistently have exactly the same processing times (or equal release dates or weights), e.g., transmission packets in data communication networks, Full Truck Loads (FTLs) in transportation by truck, and Twenty-foot Equivalent Units (TEUs) in container shipments [17]. Meanwhile, individual products may be subject to different constraints. For example, they may have different weights (importance) or may be required to meet unequal due dates or release dates requested by customers.

Motivated from this, in this paper we study the following three special cases of $1|r_j|(\sum_{j=1}^{n} w_jC_j, f_{\max})$: arbitrary processing times, equal release dates and equal weights, denoted as $1||(\sum_{j=1}^{n} C_j, f_{\max})$; equal processing times, arbitrary release dates and equal weights, denoted as $1|r_j, p_j = p|(\sum_{j=1}^{n} C_j, f_{\max})$; equal processing times, equal release dates and arbitrary weights, denoted as $1|p_j = p|(\sum_{j=1}^{n} w_jC_j, f_{\max})$. We present $O(n^3)$-time algorithms for these three cases. The results improve the previous algorithms in [9–15]. In particular, the $O(n^3)$-time algorithm presented for $1|r_j, p_j = p|(\sum_{j=1}^{n} C_j, f_{\max})$ is the first polynomial time algorithm for it.

To prove the correctness of the obtained algorithms, we need the following approach:

Lemma 1.1. *([1]) 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 $\alpha||(f, g)$.*

The paper is organized as follows. Sections 2–4 are devoted to problems $1||(\sum_{j=1}^{n} C_j, f_{\max})$, $1|r_j, p_j = p|(\sum_{j=1}^{n} C_j, f_{\max})$ and $1|p_j = p|(\sum_{j=1}^{n} w_jC_j, f_{\max})$, respectively. Finally, we conclude the paper in Section 5 with some discussion and directions for future work.

2. An $O(n^3)$-time algorithm for $1||(\sum_{j=1}^{n} C_j, f_{\max})$

In this section we will present an $O(n^3)$-time algorithm for $1||(\sum_{j=1}^{n} C_j, f_{\max})$.

Let $\sigma = (J_{\sigma(1)}, J_{\sigma(2)}, \cdots, J_{\sigma(n)})$ denote a schedule in which $J_{\sigma(i)}$ is the job scheduled at the i-th position in σ. Since $\sum_{j=1}^{n} C_j$ and $f_{\max}$ are both regular, we can consider only the schedules without idle times. Therefore, we have:

Lemma 2.1. *In a feasible schedule $\sigma = (J_{\sigma(1)}, J_{\sigma(2)}, \cdots, J_{\sigma(n)})$, $S_{\sigma(1)} = 0$, $S_{\sigma(i)} = S_{\sigma(i-1)} + p_{\sigma(i-1)}$, $i = 2, \ldots, n$.*

The algorithm maintains implicitly a position index k_j^m for each job J_j in the m-th iteration, $m = 0, 1, \ldots$. The ordinal number of J_j in the current tentative Pareto optimal schedule cannot be larger than k_j^m. Initially, the position indices of all the jobs are equal to n, i.e., $k_j^0 = n$, $j = 1, 2, \ldots, n$. During the m-th iteration, all the position indices of the jobs can be maintained via a Candidate Set Family $F^m = \{\mathcal{J}_1^m, \mathcal{J}_2^m, \ldots, \mathcal{J}_n^m\}$, where $\mathcal{J}_i^m$ is the set of jobs with position index i, $i = 1, 2, \ldots, n$.

A family F of sets $\mathcal{J}_1, \mathcal{J}_2, \ldots, \mathcal{J}_n$ is called a Candidate Set Family (CSF for short) if these (possibly empty) sets are disjoint, their union is $\mathcal{J}$, and any jobs in $\mathcal{J}_i$ ($i = 1, 2, \ldots, n$) cannot be assigned to any position whose ordinal number is greater than i.

A CSF can be regarded as a generalized partition of $\mathcal{J}$, since empty sets are not allowed in an ordinary partition of $\mathcal{J}$.

The idea of CSF is motivated from [15]. The concept of position index is similar to that used in [13]. But in [13], the position indices of the jobs are a part of input. In the algorithms throughout this paper, the position indices of the jobs are updated during the iterations.

We say that a feasible schedule satisfies a CSF $F = \{\mathcal{J}_1, \mathcal{J}_2, \ldots, \mathcal{J}_n\}$ if for $i = 1, 2, \ldots, n$, the i-th job in the schedule come from $\bigcup_{h=i}^{n} \mathcal{J}_h$. Certainly, we require that $\sum_{h=1}^{i} |\mathcal{J}_h| \le i$, $i = 1, 2, \ldots, n$.

Let $\Pi(\mathcal{J})$ denote the set of all feasible schedules for $\mathcal{J}$. Let $\Pi(\mathcal{J}, F, y) \subseteq \Pi(\mathcal{J})$ denote the set of the schedules which satisfy CSF F and have maximum cost less than y. We have $\Pi\left(\mathcal{J}, F^0, +\infty\right) = \Pi(\mathcal{J})$, where $F^0 = \{\emptyset, \emptyset, \ldots, \mathcal{J}\}$.

Before solving $1||(\sum_{j=1}^{n} C_j, f_{\max})$, let us formulate the following auxiliary problem.

Auxiliary Problem: Find a schedule σ in $\Pi(\mathcal{J}, F, y)$ with minimum total completion time, i.e., $\sigma \in \Pi(\mathcal{J}, F, y)$ which solves $1|f_{\max} < y| \sum_{j=1}^{n} C_j$ (with F). (The restriction F is useful in the algorithm, but Lemmas 2.3 and 2.6 below ensure that in the analysis it can be safely ignored.)

The auxiliary problem can be solved in $O(n^2)$ time by suitably modifying Emmons's algorithm [18], or can be transformed into a problem with deadlines (and CSF F) in $O(\log \sum_j p_j)$ time and then be solved in $O(n \log n)$ time by modifying Smith's rule [19]. However, combining the fact that there are at most $n(n-1)/2+1$ Pareto optimal points, we can only obtain an $O(n^3 \min\{n, \log \sum_j p_j\})$-time algorithm for $1||(\sum_{j=1}^{n} C_j, f_{\max})$.

In fact, CSF is introduced mainly for improving the overall running time of the obtained algorithms. Below, we first give an $O(n^3 \log n)$-time algorithm for the auxiliary problem, and then improve its time complexity to $O(n^3)$. The improved algorithm will be used as a central subroutine in the $O(n^3)$-time algorithm for $1||(\sum_{j=1}^{n} C_j, f_{\max})$.

Algorithm AUX1:

Step 1. Initially, set $m = 0$. Let $F^m = \{\mathcal{J}_1^m, \mathcal{J}_2^m, \ldots, \mathcal{J}_n^m\}$, where $\mathcal{J}_i^m = \mathcal{J}_i$ and the jobs in $\mathcal{J}_i^m$ are stored in a Max-heap ordered by processing times, $i = 1, 2, \ldots, n$.

Step 2. Construct schedule σ^m: For $i = n, n-1, \ldots, 1$ (this backward ordering is used crucially), assign the i-th position to the job with largest processing time in $\bigcup_{h=i}^{n} \mathcal{J}_h^m \backslash \{J_{\sigma^m(n)}, J_{\sigma^m(n-1)}, \ldots, J_{\sigma^m(i+1)}\}$ (ties broken arbitrarily).

Step 3. Set the start times of the jobs in σ^m: Let $S_{\sigma^m(1)} = 0$, $S_{\sigma^m(i)} = S_{\sigma^m(i-1)} + p_{\sigma^m(i-1)}$, $i = 2, \ldots, n$.

Step 4. Adjust F^m: For $i = n, n-1, \ldots, 1$, check the inequality $f_{\sigma^m(i)}(C_{\sigma^m(i)}) < y$. If the inequality does not hold, then find the largest index e such that $f_{\sigma^m(i)}(C_{\sigma^m(e-1)} + p_{\sigma^m(i)}) < y$. Delete $J_{\sigma^m(i)}$ from its original set in F^m and insert it in $\mathcal{J}_e^m$. If e cannot be found, then return $\emptyset$.

Step 5. If no adjustment has been done for F^m after Step 4 (i.e., for all i the inequality is correct), then return σ^m. Otherwise, check the condition $\sum_{h=1}^{i} \left|\mathcal{J}_h^m\right| \le i$ for $i = 1, 2, \ldots, n$. If for any i the condition does not hold, then return $\emptyset$. Otherwise, let $F^{m+1} = F^m$ (F^m has been adjusted already) and then set $m = m + 1$. Go to the next iteration (Step 2).

In Step 2, Algorithm AUX1 uses the RLPTL (Restricted Largest Processing Time Last) rule: Always select the remaining eligible job with largest processing time and assign it to the currently last position.

Max-heap is used in Step 1 of Algorithm AUX1. Similarly, Min-heap is used in Algorithm AUX3 (see Section 4). Recall that a heap data structure is an array object which can be viewed as a nearly complete binary tree [20]. Max-heap (Min-heap) with τ values has the following properties: 1) the maximum (minimum) of all values stored in the heap can be retrieved in constant time; 2) inserting a value into the heap takes $O(\log \tau)$ time; 3) deleting the maximum (minimum) value from the heap takes $O(\log \tau)$ time.

Step 1 can be implemented in $O(n)$ time. Step 2 can be implemented in $O(n \log n)$ time in each iteration. Steps 3 and 5 require $O(n)$ time in each iteration.

In Step 4, each job deletion or insertion requires $O(\log n)$ time. Since there are n jobs and each job goes through at most $n-1$ sets (from $\mathcal{J}_n$ to $\mathcal{J}_1$), the total number of job deletion and insertion is $O(\sum_i |\mathcal{J}_i^m|) = O(n^2)$, and the total number of iterations is $O(n^2)$. Checking the inequalities for all jobs in one iteration requires $O(n)$ time. Hence, the complexity contribution of Step 4 for all iterations is $O(n^3)$.

The overall running time of Algorithm AUX1 is $O(n^3 \log n)$, which is determined by Step 2 for all iterations.

Lemma 2.2. *Let $\sigma^m = (J_{\sigma^m(1)}, J_{\sigma^m(2)}, \cdots, J_{\sigma^m(n)})$ be the schedule obtained at iteration m ($m = 0, 1, \ldots$) of Algorithm AUX1 subject to F^m. Let $\sigma = (J_{\sigma(1)}, J_{\sigma(2)}, \cdots, J_{\sigma(n)})$ be any feasible schedule subject to F^m. Then the following properties hold:*
1) $S_{\sigma^m(i)} \leq S_{\sigma(i)}$, $i = 1, 2, \ldots, n$;
2) $C_{\sigma^m(i)} \leq C_{\sigma(i)}$, $i = 1, 2, \ldots, n$.

Proof. It suffices to show a transformation of σ into σ^m. The transformation may decrease the job start times at some positions, but will not increase the job start or completion time at any position.

Let us compare σ^m and σ backwardly (right-to-left) looking for a difference between the jobs. Suppose that the first difference occurs at the k-th position, which is occupied by jobs J_i and J_j in σ^m and σ, respectively. By the RLPTL rule, we know that $p_i \geq p_j$. Moreover, both J_i and J_j come from $\bigcup_{h=k}^{n} \mathcal{J}_h^m \backslash \{J_{\sigma^m(n)}, J_{\sigma^m(n-1)}, \ldots, J_{\sigma^m(k+1)}\}$. We can safely interchange J_i and J_j in σ, obeying F^m, without increasing the job start or completion time at any position.

Repetition of this argument shows that σ can be safely transformed into σ^m. It follows that properties (1) and (2) of the lemma hold.

Lemma 2.3 below is due to [15]. This lemma is crucial in the analysis of the presented algorithms. We include its proof here for completeness.

Lemma 2.3. *For $m = 1, 2, \ldots$, Algorithm AUX1 ensures that $\Pi(\mathcal{J}, F^m, y) = \Pi\left(\mathcal{J}, F^{m-1}, y\right)$.*

Proof. Obviously, we have $\Pi(\mathcal{J}, F^m, y) \subseteq \Pi\left(\mathcal{J}, F^{m-1}, y\right)$. Thus, it is sufficient to prove that $\Pi\left(\mathcal{J}, F^{m-1}, y\right) \subseteq \Pi(\mathcal{J}, F^m, y)$.

Let $\sigma \in \Pi\left(\mathcal{J}, F^{m-1}, y\right)$. Let $\mathcal{J}_i^{m-1} \in F^{m-1}$ and $\mathcal{J}_i^m \in F^m$ with $J_j \in \bigcup_{h=i+1}^{n} \mathcal{J}_h^{m-1}$ but $J_j \in \mathcal{J}_i^m$. Then we have: $f_j(C_{\sigma^{m-1}(i+1)}) \geq y$. By Lemma 2.2, we have: $f_j(C_{\sigma(i+1)}) \geq f_j(C_{\sigma^{m-1}(i+1)}) \geq y$, which means that in σ job J_j cannot be assigned to a position whose ordinal number is larger than i. Hence we get $\sigma \in \Pi(\mathcal{J}, F^m, y)$.

Combining Lemmas 2.2 and 2.3, we get the following theorem, which shows that Algorithm AUX1 solves the Auxiliary Problem.

Theorem 2.4. *Let σ^{last} be the schedule obtained at the last iteration of Algorithm AUX1. If $\sigma^{last} = \emptyset$, then $\Pi(\mathcal{J}, F, y) = \emptyset$; Otherwise σ^{last} is a schedule which has minimum total completion time among all schedules in $\Pi(\mathcal{J}, F, y)$.*

Next, we improve AUX1 to ensure that each iteration can be accomplished in $O(n)$ time. Thus, the overall running time is $O(n^3)$. The idea is motivated from [21].

Algorithm IMPROAUX1:

Step 1. Initially, set $m = 0$. Let $F^m = \{\mathcal{J}_1^m, \mathcal{J}_2^m, \ldots, \mathcal{J}_n^m\}$, where $\mathcal{J}_i^m = \mathcal{J}_i$, $i = 1, 2, \ldots, n$.

Step 2. Construct schedule σ^m: For $i = n, n-1, \ldots, 1$, assign the i-th position to the job with largest processing time in $\bigcup_{h=i}^{n} \mathcal{J}_h^m \backslash \{J_{\sigma^m(n)}, J_{\sigma^m(n-1)}, \ldots, J_{\sigma^m(i+1)}\}$ (ties broken in favor of the job with smallest cost when completed at time $C_{\sigma^m(i)} = \sum_j p_j - \sum_{l=i+1}^{l=n} p_{\sigma^m(l)}$).

Step 3. Set the start times of the jobs in σ^m: Let $S_{\sigma^m(1)} = 0$, $S_{\sigma^m(i)} = S_{\sigma^m(i-1)} + p_{\sigma^m(i-1)}$, $i = 2, \ldots, n$.

Step 4. Adjust F^m and σ^m: For $i = n, n-1, \ldots, 1$, check the inequality $f_{\sigma^m(i)}(C_{\sigma^m(i)}) < y$. If all inequalities hold, then return σ^m and the algorithm terminates. Otherwise, pick a job $J_{\sigma^m(i)}$ such that $f_{\sigma^m(i)}(C_{\sigma^m(i)}) \geq y$. Delete $J_{\sigma^m(i)}$ from its original set in F^m and insert it in $\mathcal{J}_{i-1}^m$. Then, adjust σ^m as follows. Let $E(i) = \{l | 1 \leq l \leq i-1 \wedge f_{\sigma^m(l)}(C_{\sigma^m(i)}) < y\}$. If $E(i) = \emptyset$, then return $\emptyset$. Otherwise, find the job with largest processing time in $E(i)$, say $J_{\sigma^m(e)}$ (ties broken in favor of the job with smallest cost when completed at time $C_{\sigma^m(i)}$). Let $J_{\sigma^m(e)}$ be scheduled at the i-th position instead of $J_{\sigma^m(i)}$. Move backward over consecutive positions, starting from $i-1$ and ending by e, to find suitable positions for jobs $J_{\sigma^m(i)}, J_{\sigma^m(i-1)}, \ldots, J_{\sigma^m(e+1)}$. Let c denote the current position. Let J_x denote the current job, initially $J_{\sigma^m(i)}$. When $c > e$, if $p_{\sigma^m(c)} > p_x$, or $p_{\sigma^m(c)} = p_x$ and $f_{\sigma^m(c)}(C_{\sigma^m(c)}) \leq f_x(C_{\sigma^m(c)})$, then $J_{\sigma^m(c)}$ and J_x keep unchanged and we continue with position $c-1$ and job J_x. Otherwise, let J_x be scheduled at the c-position, and continue with position $c-1$ and job $J_{\sigma^m(c)}$ (i.e., update the current job J_x to be $J_{\sigma^m(c)}$). When $c = e$, simply let J_x be scheduled at the c-position. Finally, let $F^{m+1} = F^m$, $\sigma^{m+1} = \sigma^m$ (F^m and σ^m have been adjusted already) and then set $m = m + 1$. Go to Step 3.

In Steps 2 and 4 of Algorithm IMPROAUX1, the jobs are chosen by the RLPTL-SC (RLPTL-Smallest Cost) rule: Always select the remaining eligible job with largest processing time, ties broken in favor of the job with smallest cost.

In IMPROAUX1, Step 2 is executed only once (for constructing σ^0 in $O(n^2)$ time). For $m > 0$, σ^m is obtained by adjusting σ^{m-1}. It is easy to check that IMPROAUX1 obeys the RLPTL rule and the schedule obtained at its last iteration is the same as that obtained in AUX1. Step 4 of IMPROAUX1 requires $O(n)$ time in one iteration for adjusting an inequality violation. Since the number of inequality violations is $O(n^2)$, the overall running time of Algorithm IMPROAUX1 is $O(n^3)$.

There are two main differences between Algorithms AUX1 and IMPROAUX1: (1) In AUX1, in each iteration, we obtain a CSF and then use it to construct a schedule, i.e., adjusting a CSF and constructing a schedule are done separately. It takes $O(n \log n)$ time to construct a schedule (in Step 2). Since there are $O(n^2)$ iterations, AUX1 is implemented in $O(n^3 \log n)$ time. In IMPROAUX1, we obtain a schedule by adjusting the current CSF and the current schedule in Step 4. Adjusting the CSF is easy. We just move an inequality-violated job from its current set in the CSF into the preceding set. Adjusting the current schedule can be done in $O(n)$ time. Therefore, IMPROAUX1 is implemented in $O(n^3)$ time. (2) If IMPROAUX1 returns a feasible schedule, then this schedule must be Pareto optimal (by Lemma 2.5). However, if AUX1 returns a feasible schedule, we cannot ensure this schedule is Pareto optimal. To ensure it is Pareto optimal, we have to take the job costs into account in Step 2 of AUX1. However, Step 2 of AUX1 will be implemented in $O(n^2)$ time, leading to an overall time complexity of $O(n^4)$.

Note that Lemmas 2.2 and 2.3 and Theorem 2.4 still hold for Algorithm IMPROAUX1.

Lemma 2.5. *If Algorithm IMPROAUX1 generates a schedule σ at the last iteration, then σ is a Pareto optimal schedule for $1||(\sum_{j=1}^{n} C_j, f_{\max})$.*

Proof. The proof first appeared in [11]. We include it here for completeness.

By Lemma 1.1, it suffices to show that σ is optimal simultaneously for $1|f_{\max} \le f_{\max}(\sigma)|\sum_{j=1}^{n} C_j$ and $1|\sum_{j=1}^{n} C_j \le \sum_{j=1}^{n} C_j(\sigma)|f_{\max}$.

Since $f_{\max}(\sigma) < y$, by Theorem 2.4, it is evident that σ solves $1|f_{\max} \le f_{\max}(\sigma)|\sum_{j=1}^{n} C_j$ optimally.

Assume that π is optimal for $1|\sum_{j=1}^{n} C_j \le \sum_{j=1}^{n} C_j(\sigma)|f_{\max}$, not σ. It follows that $\sum_{j=1}^{n} C_j(\pi) \le \sum_{j=1}^{n} C_j(\sigma)$, $f_{\max}(\pi) < f_{\max}(\sigma)$ and thus π is also feasible for $1|f_{\max} \le f_{\max}(\sigma)|\sum_{j=1}^{n} C_j$. Therefore, we get $\sum_{j=1}^{n} C_j(\pi) \ge \sum_{j=1}^{n} C_j(\sigma)$. Actually, we get $\sum_{j=1}^{n} C_j(\pi) = \sum_{j=1}^{n} C_j(\sigma)$.

Let us compare σ and π backwardly looking for a difference between the jobs. Suppose that the first difference occurs at the k-th position, which is occupied by jobs J_i and J_j in σ and π, respectively. By the RLPTL-SC rule, we know that $p_i \ge p_j$.

If $p_i > p_j$, then we interchange J_i and J_j in π to get π'. Note that π' is feasible for $1|f_{\max} \le f_{\max}(\sigma)|\sum_{j=1}^{n} C_j$ and $\sum_{j=1}^{n} C_j(\pi') < \sum_{j=1}^{n} C_j(\pi) = \sum_{j=1}^{n} C_j(\sigma)$, contradicting the fact that σ is optimal for $1|f_{\max} \le f_{\max}(\sigma)|\sum_{j=1}^{n} C_j$.

Hence it must be true that $p_i = p_j$. Moreover, by the RLPTL-SC rule, $f_i(C_i(\sigma)) \le f_j(C_j(\pi))$. Thus, we can safely interchange J_i and J_j in π without affecting the cost of the schedule. Repetition of this argument shows that π can be safely transformed into σ, contradicting the assumption that σ is not optimal for $1|\sum_{j=1}^{n} C_j \le \sum_{j=1}^{n} C_j(\sigma)|f_{\max}$. Therefore, σ also solves $1|\sum_{j=1}^{n} C_j \le \sum_{j=1}^{n} C_j(\sigma)|f_{\max}$ optimally.

Hence, σ is a Pareto optimal schedule for $1||(\sum_{j=1}^{n} C_j, f_{\max})$.

Now, we are ready to describe the algorithm for constructing the Pareto set $\Omega(\mathcal{J})$ for $1||(\sum_{j=1}^{n} C_j, f_{\max})$ which consists of all Pareto optimal points together with the corresponding schedules.

Algorithm MAIN1:

Step 1. Initially, set $s = 0$, $y^s = +\infty$ and $\sigma_s = \emptyset$. Let $F^s = \{\mathcal{J}_1^s, \mathcal{J}_2^s, \ldots, \mathcal{J}_n^s\}$, where $\mathcal{J}_n^s = \mathcal{J}$ and $\mathcal{J}_i^s = \emptyset$ for $i = 1, 2, \ldots, n-1$. Let $\Omega(\mathcal{J}) = \emptyset$.

Step 2. Run Algorithm IMPROAUX1 to get a schedule σ_{s+1} with minimum total completion time among all schedules in $\Pi(\mathcal{J}, F^s, y^s)$. (Step 2 of IMPROAUX1 will be executed only once, just for σ_1. The reason is that for $s > 0$ σ_{s+1} is obtained by adjusting a series of schedules, starting with σ_s.) Let upon the completion of Algorithm IMPROAUX1 $F^{s+1} = \{\mathcal{J}_1^{s+1}, \mathcal{J}_2^{s+1}, \ldots, \mathcal{J}_n^{s+1}\}$ be obtained.

Step 3. If $\sigma_{s+1} \neq \emptyset$, then include $(\sum_{j=1}^{n} C_j(\sigma_{s+1}), f_{\max}(\sigma_{s+1}), \sigma_{s+1})$ into $\Omega(\mathcal{J})$. Set $y^{s+1} = f_{\max}(\sigma_{s+1})$. Set $s = s + 1$. Go to the next iteration (Step 2).

Step 4. If $\sigma_{s+1} = \emptyset$, then return $\Omega(\mathcal{J})$.

Lemma 2.6. *For $s = 0, 1, \ldots$, Algorithm MAIN1 ensures that $\Pi(\mathcal{J}, F^s, y^s) = \Pi\left(\mathcal{J}, F^0, y^s\right)$, where $F^0 = \{\emptyset, \emptyset, \ldots, \mathcal{J}\}$ and $y^0 = +\infty$.*

Proof. At the first iteration of Algorithm MAIN1, σ_1 and F^1 were obtained. By Lemma 2.3, we have: $\Pi\left(\mathcal{J}, F^1, y^0\right) = \Pi\left(\mathcal{J}, F^0, y^0\right)$. Since $y^1 = f_{\max}(\sigma_1) < y^0$, we have: $\Pi\left(\mathcal{J}, F^1, y^1\right) = \Pi\left(\mathcal{J}, F^0, y^1\right)$. By Lemma 2.3, we get: $\Pi\left(\mathcal{J}, F^2, y^1\right) = \Pi\left(\mathcal{J}, F^1, y^1\right) = \Pi\left(\mathcal{J}, F^0, y^1\right)$. Hence we get: $\Pi\left(\mathcal{J}, F^2, y^2\right) = \Pi\left(\mathcal{J}, F^0, y^2\right)$. Repeating the argument for all iterations we obtain $\Pi(\mathcal{J}, F^s, y^s) = \Pi\left(\mathcal{J}, F^0, y^s\right)$, $s = 0, 1, \ldots$.

We get the following theorem.

Theorem 2.7. *Algorithm MAIN1 solves $1||(\sum_{j=1}^{n} C_j, f_{\max})$ in $O(n^3)$ time.*

3. An $O(n^3)$-time algorithm for $1|r_j, p_j = p|(\sum_{j=1}^{n} C_j, f_{\max})$

In this section we will present an $O(n^3)$-time algorithm for $1|r_j, p_j = p|(\sum_{j=1}^{n} C_j, f_{\max})$.

Without causing confusion, in this section we re-use some notations and terminologies defined in the preceding section, such as $\sigma = (J_{\sigma(1)}, J_{\sigma(2)}, \cdots, J_{\sigma(n)})$, CSF $F = \{\mathcal{J}_1, \mathcal{J}_2, \ldots, \mathcal{J}_n\}$, $\Pi(\mathcal{J}, F, y)$, $F^0 = \{\emptyset, \emptyset, \ldots, \mathcal{J}\}$, to name a few.

Since $\sum_{j=1}^{n} C_j$ and $f_{\max}$ are both regular, we have:

Lemma 3.1. *In a feasible schedule $\sigma = (J_{\sigma(1)}, J_{\sigma(2)}, \cdots, J_{\sigma(n)})$, $S_{\sigma(1)} = r_{\sigma(1)}$, $S_{\sigma(i)} = \max\{r_{\sigma(i)}, S_{\sigma(i-1)} + p\}$, $i = 2, \ldots, n$.*

Auxiliary Problem: Find a schedule σ in $\Pi(\mathcal{J}, F, y)$ with minimum total completion time, i.e., $\sigma \in \Pi(\mathcal{J}, F, y)$ which solves $1|r_j, p_j = p, f_{\max} < y| \sum_{j=1}^{n} C_j$ (with F).

The auxiliary problem can be solved in $O(n^3 \log n)$ time by Algorithm AUX2.

Algorithm AUX2:

Modify Algorithm AUX1 by the RLRDL (Restricted Largest Release Date Last) rule: Replace "processing time" by "release date" in Steps 1 and 2 of AUX1, and in Step 3 set the start times of the jobs in σ^m by Lemma 3.1.

Below, we give an $O(n^3)$-time algorithm for the auxiliary problem, which will lead to an $O(n^3)$-time algorithm for $1|r_j, p_j = p|(\sum_{j=1}^{n} C_j, f_{\max})$. (Thus, we do not actually need Algorithm AUX2. We include it only for making this section a clear correspondence to the preceding section. Similarly for Algorithm AUX3 in the next section.)

Algorithm IMPROAUX2:

Step 1. Initially, set $m = 0$. Let $F^m = \{\mathcal{J}_1^m, \mathcal{J}_2^m, \ldots, \mathcal{J}_n^m\}$, where $\mathcal{J}_i^m = \mathcal{J}_i$, $i = 1, 2, \ldots, n$.

Step 2. Construct schedule σ^m: For $i = n, n-1, \ldots, 1$, assign the i-th position to the job with largest release date in $\bigcup_{h=i}^{n} \mathcal{J}_h^m \backslash \{J_{\sigma^m(n)}, J_{\sigma^m(n-1)}, \ldots, J_{\sigma^m(i+1)}\}$ (ties broken arbitrarily).

Step 3. Set the start times of the jobs in σ^m: Let $S_{\sigma(1)} = r_{\sigma(1)}$, $S_{\sigma(i)} = \max\{r_{\sigma(i)}, S_{\sigma(i-1)} + p\}$, $i = 2, \ldots, n$.

Step 4. Adjust F^m and σ^m: For $i = n, n-1, \ldots, 1$, check the inequality $f_{\sigma^m(i)}(C_{\sigma^m(i)}) < y$. If all inequalities hold, then return σ^m and the algorithm terminates. Otherwise, pick a job $J_{\sigma^m(i)}$ such that $f_{\sigma^m(i)}(C_{\sigma^m(i)}) \geq y$. Delete $J_{\sigma^m(i)}$ from its original set in F^m and insert it in $\mathcal{J}^m_{i-1}$. Then, adjust σ^m as follows. Let $E(i) = \{l | 1 \leq l \leq i-1 \wedge f_{\sigma^m(l)}(C_{\sigma^m(i)}) < y\}$. If $E(i) = \emptyset$, then return $\emptyset$. Otherwise, find the job with largest release date in $E(i)$, say $J_{\sigma^m(e)}$. Let $J_{\sigma^m(e)}$ be scheduled at the i-th position instead of $J_{\sigma^m(i)}$. Move backward over consecutive positions, starting from $i-1$ and ending by e, to find suitable positions for jobs $J_{\sigma^m(i)}, J_{\sigma^m(i-1)}, \ldots, J_{\sigma^m(e+1)}$. Let c denote the current position. Let J_x denote the current job, initially $J_{\sigma^m(i)}$. When $c > e$, if $r_{\sigma^m(c)} \geq r_x$, then $J_{\sigma^m(c)}$ and J_x keep unchanged and we continue with position $c-1$ and job J_x. Otherwise, let J_x be scheduled at the c-position, and continue with position $c-1$ and job $J_{\sigma^m(c)}$ (i.e., update the current job J_x to be $J_{\sigma^m(c)}$). When $c = e$, simply let J_x be scheduled at the c-position. Finally, let $F^{m+1} = F^m$, $\sigma^{m+1} = \sigma^m$ (F^m and σ^m have been adjusted already) and then set $m = m+1$. Go to Step 3.

In Steps 2 and 4 of Algorithm IMPROAUX2, the jobs are chosen by the RLRDL rule. Step 2 is executed only once (for constructing σ^0 in $O(n \log n)$ time). For $m > 0$, σ^m is obtained by adjusting σ^{m-1}. The overall running time of IMPROAUX2 is $O(n^3)$. Since we do not combine the Smallest Cost rule to choose a job in case of ties, Lemma 2.5 does not hold for IMPROAUX2.

Note that Lemmas 2.2 and 2.3 and Theorem 2.4 still hold for Algorithm IMPROAUX2.

Now, we are ready to describe the algorithm for constructing the Pareto set $\Omega(\mathcal{J})$ for $1|r_j, p_j = p|(\sum_{j=1}^n C_j, f_{\max})$, which is subtly different from Algorithm MAIN1.

MAIN2:

Step 1. Initially, set $s = 0$, $y^s = +\infty$ and $\sigma_s = \emptyset$. Let $F^s = \{\mathcal{J}^s_1, \mathcal{J}^s_2, \ldots, \mathcal{J}^s_n\}$, where $\mathcal{J}^s_n = \mathcal{J}$ and $\mathcal{J}^s_i = \emptyset$ for $i = 1, 2, \ldots, n-1$. Let $\Omega(\mathcal{J}) = \emptyset$, $k = 0$.

Step 2. Run Algorithm IMPROAUX2 to get a schedule σ_{s+1} with minimum total completion time among all schedules in $\Pi(\mathcal{J}, F^s, y^s)$. (Step 2 of IMPROAUX2 will be executed only once.) Let upon the completion of Algorithm IMPROAUX2 $F^{s+1} = \{\mathcal{J}^{s+1}_1, \mathcal{J}^{s+1}_2, \ldots, \mathcal{J}^{s+1}_n\}$ be obtained.

Step 3. If $\sigma_{s+1} \neq \emptyset$, then:
(i) Set $y^{s+1} = f_{\max}(\sigma_{s+1})$.
(ii) If $\sum_{j=1}^n C_j(\sigma_s) < \sum_{j=1}^n C_j(\sigma_{s+1})$ and $s > 0$, then set $k = k+1$ and $\pi_k = \sigma_s$. Include $(\sum_{j=1}^n C_j(\pi_k), f_{\max}(\pi_k), \pi_k)$ into $\Omega(\mathcal{J})$.
(iii) Set $s = s+1$. Go to the next iteration (Step 2).

Step 4. If $\sigma_{s+1} = \emptyset$, then set $k = k+1$ and $\pi_k = \sigma_s$. Include $(\sum_{j=1}^n C_j(\pi_k), f_{\max}(\pi_k), \pi_k)$ into $\Omega(\mathcal{J})$ and return $\Omega(\mathcal{J})$.

Algorithms MAIN1 and MAIN2 invoke IMPROAUX1 and IMPROAUX2 respectively. By Lemma 2.5, each feasible schedule returned by IMPROAUX1 is a Pareto optimal schedule. Therefore, there are exactly $|\Omega(\mathcal{J})| + 1$ iterations in Algorithm MAIN1, where $|\Omega(\mathcal{J})|$ is the cardinality of the Pareto set. Algorithm MAIN1 finds only Pareto optimal schedules. On the other hand, since Lemma 2.5 does not hold for IMPROAUX2, Algorithm MAIN2 may perform more iterations than $|\Omega(\mathcal{J})| + 1$. (The number of iterations in Algorithm MAIN2 is still bounded by n^2.) The algorithm can find schedules which are not Pareto optimal, but it never misses pareto optimal schedules.

Lemma 2.6 still holds for Algorithm MAIN2. We get the following theorem.

Theorem 3.2. *Algorithm MAIN2 solves* $1|r_j, p_j = p|(\sum_{j=1}^{n} C_j, f_{\max})$ *in* $O(n^3)$ *time.*

4. An $O(n^3)$-time algorithm for $1|p_j = p|(\sum_{j=1}^{n} w_jC_j, f_{\max})$

We now modify the algorithm developed in Section 2 to solve $1|p_j = p|(\sum_{j=1}^{n} w_jC_j, f_{\max})$.

Auxiliary Problem: Find a schedule σ in $\Pi(\mathcal{J}, F, y)$ with minimum total weighted completion time, i.e., $\sigma \in \Pi(\mathcal{J}, F, y)$ which solves $1|p_j = p, f_{\max} < y|(\sum_{j=1}^{n} w_jC_j, f_{\max})$ (with F).

The auxiliary problem can be solved in $O(n^3 \log n)$ time by Algorithm AUX3, or in $O(n^3)$ time by Algorithm IMPROAUX3.

Algorithm AUX3:

Modify Algorithm AUX1 as follows: Replace the RLPTL rule by the RSWL (Restricted Smallest Weight Last) rule. That is, replace “Max-heap” by “Min-heap” in Step 1, and replace “processing time” by “weight” in Steps 1 and 2.

Algorithm IMPROAUX3:

Modify Algorithm IMPROAUX1 by the RSWL-SC (RSWL-Smallest Cost) rule: Replace “processing time” by “weight” in Steps 2 and 4 of IMPROAUX1. Moreover, in Step 4, replace “When $c > e$, if $p_{\sigma^m(c)} > p_x$, or $p_{\sigma^m(c)} = p_x$ and $f_{\sigma^m(c)}(C_{\sigma^m(c)}) \leq f_x(C_{\sigma^m(c)})$” by “When $c > e$, if $w_{\sigma^m(c)} < w_x$, or $w_{\sigma^m(c)} = w_x$ and $f_{\sigma^m(c)}(C_{\sigma^m(c)}) \leq f_x(C_{\sigma^m(c)})$”.

Note that Lemmas 2.2, 2.3, 2.5 and Theorem 2.4 still hold for Algorithm IMPROAUX3.

Now,we obtain the algorithm for $1|p_j = p|(\sum_{j=1}^{n} w_jC_j, f_{\max})$.

MAIN3:

Modify Algorithm MAIN1 as follows. Replace “IMPROAUX1” by “IMPROAUX3” in Step 2.

we get:

Theorem 4.1. *Algorithm MAIN3 solves* $1|p_j = p|(\sum_{j=1}^{n} w_jC_j, f_{\max})$ *in* $O(n^3)$ *time.*

5. Discussion

In this paper we investigated the Pareto optimization scheduling problem on a single machine to minimize total weighted completion time and maximum cost simultaneously. We improved the earlier results by presenting $O(n^3)$-time algorithms for three important special cases of the problem. The algorithms provide trade-off solutions (i.e., Pareto optimal schedules) which can balance the two criteria to reach an acceptable compromise.

The algorithms are useful in situations when the processing times (or release dates, or weights) do not too much differ from each other, or in situations when the majority of the jobs has equal processing times (or release dates, or weights). Such a situation occurs also for problems with uncertain processing times (or release dates, or weights), e.g., when we only know that the processing times (or release dates, or weights) take values from a small interval around a common value.

For future research, it is interesting to extend this work to other criteria such as two min-sum criteria, or a combination of a min-sum and a maximum cost. Furthermore, the parallel machines or batch scheduling Pareto optimization problems are also meaningful.

Acknowledgments

We thank the editor and reviewers for their helpful suggestions. This work is supported by Natural Science Foundation of Shandong Province China (No. ZR2020MA030).

Conflict of interest

The authors declare there is no conflict of interest.

References

1. H. Hoogeveen, Multicriteria scheduling, *Eur. J. Oper. Res.*, **167** (2005), 592–623. https://doi.org/10.1016/j.ejor.2004.07.011
2. D. Jones, S. Firouzy, A. Labib, A. V. Argyriou, Multiple criteria model for allocating new medical robotic devices to treatment centres, *Eur. J. Oper. Res.*, **297** (2022), 652–664. https://doi.org/10.1016/j.ejor.2021.06.003
3. F. F. Ostermeier, On the trade-offs between scheduling objectives for unpaced mixed-model assembly lines, *Int. J. Prod. Res.*, **60** (2022), 866–893. https://doi.org/10.1080/00207543.2020.1845914
4. P. M. Kumar, G. C. Babu, A. Selvaraj, M. Raza, A. K. Luhach, V. G. Daaz, Multi-criteria-based approach for job scheduling in industry 4.0 in smart cities using fuzzy logic, *Soft Comput.*, **25** (2021), 12059–12074.
5. V. T'Kindt, J. C. Billaut, Multicriteria scheduling: theory, models and algorithms, second edition, Springer Verlag, Berlin, 2006.
6. P. Brucker, Scheduling algorithms, fifth edition, Springer, 2007.
7. G. Steiner, P. Stephenson, Pareto optima for total weighted completion time and maximum lateness on a single machine, *Discrete Appl. Math.*, **155** (2007), 2341–2354. https://doi.org/10.1016/j.dam.2007.06.012
8. J. K. Lenstra, A. R. Kan, P. Brucker, Complexity of machine scheduling problems, *Ann. Discrete Math.*, **1** (1977), 343–362. https://doi.org/10.1016/S0167-5060(08)70743-X
9. L. N. V. Wassenhove, F. Gelders, Solving a bicriterion scheduling problem, *Eur. J. Oper. Res.*, **4** (1980), 42–48. https://doi.org/10.1016/0377-2217(80)90038-7
10. T. C. John, Tradeoff solutions in single machine production scheduling for minimizing flow time and maximum penalty, *Comput. Oper. Res.*, **16** (1989), 471–479. https://doi.org/10.1016/0305-0548(89)90034-8

11. J. A. Hoogeveen, S. L. V. D. Velde, Minimizing total completion time and maximum cost simultaneously is solvable in polynomial time, *Oper. Res. Lett.*, **17** (1995), 205–208. https://doi.org/10.1016/0167-6377(95)00023-D

12. Y. Gao, J. J. Yuan, A note on Pareto minimizing total completion time and maximum cost, *Oper. Res. Lett.*, **43** (2015), 80–82. https://doi.org/10.1080/01576895.2014.1000811

13. Y. Gao, J. J. Yuan, Pareto minimizing total completion time and maximum cost with positional due indices, *J. Oper. Res. Soc. China*, **3** (2015), 381–387. https://doi.org/10.1007/s40305-015-0083-1

14. C. He, H. Lin, X. M. Wang, Single machine bicriteria scheduling with equal-length jobs to minimize total weighted completion time and maximum cost, *4OR-Q. J. Oper. Res.*, **12** (2014), 87–93.

15. A. A. Lazarev, D. I. Arkhipov, F. Werner, Scheduling jobs with equal processing times on a single machine: minimizing maximum lateness and makespan, *Optim. Lett.*, **11** (2016), 165–177. https://doi.org/10.1007/s11590-016-1003-y

16. J. A. Hoogeveen, Single-machine scheduling to minimize a function of two or three maximum cost criteria, *J. Algorithms*, **21** (1996), 415–433. https://doi.org/10.1006/jagm.1996.0051

17. S. A. Kravchenko, F. Werner, Parallel machine problems with equal processing times: A survey, *J. Scheduling*, **14** (2011), 435–444. https://doi.org/10.1007/s10951-011-0231-3

18. H. Emmons, A note on a scheduling problem with dual criteria, *Nav. Res. Log.*, **22** (1975), 615–616. https://doi.org/10.1002/nav.3800220317

19. W. E. Smith, Various optimizers for single-stage production, *Nav. Res. Log.*, **3** (1956), 59–66. https://doi.org/10.1002/nav.3800030106

20. T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Introduction to algorithms, third edition, MIT press, 2009.

21. F. Koehler, S. Khuller, Optimal batch schedules for parallel machines, in *Proceedings of the 13th International Conference on Algorithms and Data Structures*, Springer-VerlagBerlin, Heidelberg, 2013.