Optimization Letters (2022) 16:1817–1833
https://doi.org/10.1007/s11590-021-01805-1

ORIGINAL PAPER

Single machine batch scheduling with non-increasing time slot costs

Junfeng Ren[1,2]

Received: 22 February 2021 / Accepted: 4 September 2021 / Published online: 27 September 2021

Abstract
This work studies batch scheduling problems in which non-increasing time slot costs are taken into consideration on a single machine. In these problems, each batch occupies some time slots, and the cost yielded by batch setup and job processing in a batch is dependent on the time slots. The objective is to minimize the sum of the total time slot cost and one of the traditional performance measures, including the total flow time, maximum lateness/tardiness, and weighted number of tardy jobs. For these objectives, it is necessary to determine not only the optimal job sequence and batch partition but also the optimal time slot allocation for the batches. For each problem, we find the structured properties of the optimal schedule and then propose an algorithm based on dynamic programming. All the presented algorithms are of polynomial time if the input size of the time slot costs is $O(K)$.

Keywords Polynomial-time algorithm · Time slot cost · Batch · Scheduling

1 Introduction

This paper investigates batch scheduling with non-increasing time slot costs on a single machine. The problem arises from the continuous casting stage in steel production, in which molten steel is turned into slabs on a continuous caster [13]. Before the continuous casting stage, molten steel is smelted in converters. The molten steel in the same converter is called a charge, which can be regarded as a job. These charges are refined and then processed on a continuous caster. The continuous caster needs to replace a tundish before the processing of a cast. A cast refers to a sequence of charges

✉ Junfeng Ren
rjf@hpu.edu.cn

1 Key Laboratory of Data Analytics and Optimization for Smart Industry (Northeastern University), Ministry of Education, Shenyang 110819, China

2 School of Mathematics and Information Science, Henan Polytechnic University, Jiaozuo 454000, China

that are processed consecutively into slabs. A cast can be regarded as a batch, and the time spent replacing the tundish can be regarded as the setup time before a batch. The production scheduling on the continuous caster is a serial batch scheduling. It is well known that steel production is energy intensive. When making optimal production scheduling in the current situation of rising energy prices, decision-makers should consider not only improving production efficiency but also saving energy or reducing energy costs. In steel plants, power supply companies supply electricity with time-of-use electricity prices, i.e., the electricity price varies for different time slots. There is a situation that often occurs; that is, during a production planning period (usually less than one day), the electricity price changes, but it does not increase over time. Thus, if the steel plant postpones the production tasks in the high price period to the low price period, the electricity costs can be significantly reduced. Considering the changes in electricity prices over time, finding the optimal batch scheduling that can improve production efficiency and reduce electricity costs in the continuous casting stage is the motivation for our study.

From the above practical scenario, the production planning period can be partitioned into multiple time slots with unit length. Because the power of the caster is fixed during the process of changing the tundish and producing the slab, the electricity cost generated by the operation of the machine in any time slot can be obtained according to the corresponding electricity price, and the electricity cost corresponding to the time slot is referred to as the TSC (time slot cost). To improve production efficiency and reduce electricity costs, we need to determine not only the optimal job sequence and batch partition but also the time slot allocation for the batches.

The problem herein is related to batch scheduling and TSC. A brief review of the scheduling studies related to the two subjects is presented as follows. Potts and Kovalyov [11] and Allahverdi et al. [1] provided extensive surveys of batch scheduling with various problems and models. Coffman et al. [4] considered the total flow time objective function and obtained the optimal scheduling by adopting their O(n) time algorithm to partition the SPT (shortest processing time first) job sequence into batches. For the maximum lateness minimization problem, Webster and Baker [16] provided an O(n^2) time algorithm to partition the EDD (earliest due date first) job sequence into batches to obtain an optimal schedule. Hochbaum and Landy [6] proposed a pseudopolynomial time algorithm for minimizing the weighted number of tardy jobs, and they indicated that the early jobs in the optimal schedule were sequenced in accordance with the EDD rule, followed by all of the tardy jobs. For more or recent studies related to batch scheduling, readers can refer to the literature [5, 7–10, 12, 13, 17].

Wan and Qi [14] first introduced the TSC into the scheduling model with individual processing. In their model, the planning horizon is partitioned into multiple time slots with unit length, and the operational costs vary over the time slots. With this consideration, the objective is to minimize the total TSC plus one of those traditional measures of scheduling performance, i.e., the total flow time, maximum lateness/tardiness, and weighted number of tardy jobs. In the case that the TSCs vary arbitrarily with time, they showed that all of the problems were strongly NP-hard. They provided polynomial-time algorithms under the condition that the TSCs are non-increasing. After that, some researchers followed their study. The problem of minimizing the total TSC plus the makespan was researched by Zhong and Liu [19]. Similarly, the general problem was determined to be strongly NP-hard. They provided an analysis of the special problem with non-increasing TSCs. The preemptive scheduling problems were investigated by Chen et al. [2]. They studied two problems: one was to minimize the makespan plus the total TSC on unrelated machines, and the other was to minimize the total weighted completion time plus the total TSC. Zhao et al. [18] followed the work on the weighted completion time; they solved the problems with the TSCs decreasing in three patterns, including decreasing convex, decreasing concave, and decreasing linear, and they proved that the problem with the TSCs decreasing in an arbitrary way was strongly NP-hard. Chen et al. [3] investigated the problem with outsourcing and variable TSCs on a single machine and two open-shop machines, respectively. Wang [15] considered machine calibration, which is necessary for running a job, in the scheduling problem with TSCs.

All the above scheduling studies that considered TSCs focused on individual processing, but there is no work involving batch scheduling. Therefore, to solve the problem derived from the continuous casting stage of steel production, we investigated batch scheduling problems with TSCs. For each problem, we determined the structured properties of the optimal schedule and proposed algorithms based on dynamic programming.

The remaining parts are arranged as below. The notations and detailed descriptions of the problems are provided in Sect. 2. Sections 3, 4, and 5 investigate the batch scheduling problems of minimizing the total TSC plus a traditional performance measure. More specifically, Sect. 3 discusses the total flow time, Sect. 4 focuses on maximum lateness/tardiness, and Sect. 5 considers the weighted number of tardy jobs. A brief conclusion is provided in Sect. 6.

2 Problem description

At time zero, n independent jobs $J_1, J_2, \ldots, J_n$ are available, and they will be processed non-preemptively in batches $B_1, B_2, \ldots B_m$ on a single machine. For job J_j, let p_j, w_j, d_j, and C_j denote the processing time, weight, due date, and completion time, respectively. s is used to denote the setup time, which is needed before each batch processing. p_j, d_j, and s are integer numbers. Under the batch availability assumption, C_j is defined as the time when the processing of the last job in the batch that includes J_j is completed. The lateness and the tardiness can be determined as $L_j = C_j - d_j$ and $T_j = \max\{0, C_j - d_j\}$, respectively. If $C_j > d_j$, let $U_j = 1$ represent that job J_j is tardy. Otherwise, let $U_j = 0$ represent that job J_j is on-time.

All jobs should be scheduled within a planning horizon containing K time slots, in which each time slot has a unit length. To guarantee feasibility, it is assumed that $s + \sum_{j=1}^{n} p_j \leq K$. Because each time slot has a unit length, for the kth time slot, the starting time can be denoted as $k - 1$, and the ending time can be denoted as k. π_k is used to represent the cost yielded by using the kth time slot, and π_{J_j} is the total TSC of job J_j. It then follows that $\pi_{J_j} = \sum_{r=k+1}^{k+p_j} \pi_r$ if the processing of J_j starts at time k. Letting π_{B_i} denote the total TSC of batch B_i and π_{s_i} denote the TSC of the setup time $s_i = s$ for B_i; $\pi_{B_i} = \pi_{s_i} + \sum_{J_j \in B_i} \pi_{J_j}$. This study focuses on the batch scheduling problem with non-increasing TSCs, i.e., $\pi_1 \geq \pi_2 \geq \cdots \geq \pi_K$. Thus, if the scheduler delays the processing of some jobs, the total TSC can be reduced.

In scheduling problems, the total flow time (expressed as $\sum_{j=1}^{n} C_j$) represents the total inventory or holding costs, the maximum lateness and tardiness (expressed as $L_{\max} = \max_j\{L_j\}$ and $T_{\max} = \max_j\{T_j\}$, respectively) indicate the worst violation of the due dates, and the weighted number of tardy jobs (expressed as $\sum_{j=1}^{n} w_j U_j$) represents the total tardiness costs. These traditional performance measures reflect production efficiency or service level, and they are most commonly used as the objective function in various scheduling problems [4–6, 10, 14]. The time slots occupied by different schedules are probably different, as is the total TSC yielded by different schedules if the TSCs vary over time. In this case, it is necessary to consider the total TSC as a part of the total cost. Therefore, we study the following problems, of which the objectives are to minimize the sum of one of the above traditional performance measures and the total TSC.

P1: $1|\text{batch, non-inc-slotcost}|\sum_{j=1}^{n} C_j + \sum_{i=1}^{m} \pi_{B_i}$.

P2: $1|\text{batch, non-inc-slotcost}|L_{\max} + \sum_{i=1}^{m} \pi_{B_i}$ and $1|\text{batch, non-inc-slotcost}|T_{\max} + \sum_{i=1}^{m} \pi_{B_i}$.

P3: $1|\text{batch, non-inc-slotcost}|\sum_{j=1}^{n} w_j U_j + \sum_{i=1}^{m} \pi_{B_i}$.

In the triplet notations of the problems, non-inc-slotcost represents the non-increasing TSCs. For each problem, the joint decisions on the job sequence, the batch partition $B = (B_1, B_2, \ldots B_m)$, and the time slot allocation for the batches need to be determined to minimize the objective function.

3 $1|\text{batch, non-inc-slotcost}|\sum_{j=1}^{n} C_j + \sum_{i=1}^{m} \pi_{B_i}$

The objective of the problem discussed in this section is to minimize the sum of the total TSC and total flow time. We first propose some properties of an optimal schedule.

Lemma 1. For problem $1|\text{batch, non-inc-slotcost}|\sum_{j=1}^{n} C_j + \sum_{i=1}^{m} \pi_{B_i}$, there is an optimal schedule where any batch B_i has no idle time.

Proof Suppose that S is an optimal schedule where batch B_i has Δ units of idle time before processing job J_j. In batch B_i, we can postpone the setup and processing operations prior to job J_j by Δ units of time. In this new schedule S', the total flow time is not changed under the assumption of batch availability. Additionally, the total TSC does not increase because the TSCs are non-increasing. Hence, S' is also optimal. To summarize the above analysis, we can obtain an optimal schedule where any batch B_i has no idle time by repeating this procedure a finite number of times. $\square$

Remark 1. The property described in Lemma 1 is also valid for the optimal schedules of P2 and P3.

Lemma 2. For problem $1|\text{batch, non-inc-slotcost}|\sum_{j=1}^{n} C_j + \sum_{i=1}^{m} \pi_{B_i}$, an optimal job sequence can be obtained by sequencing the jobs in accordance with the SPT rule.

Proof Within each batch, the job sequence is immaterial under the assumption of batch availability. Suppose that S is an optimal schedule in which the jobs are sequenced in

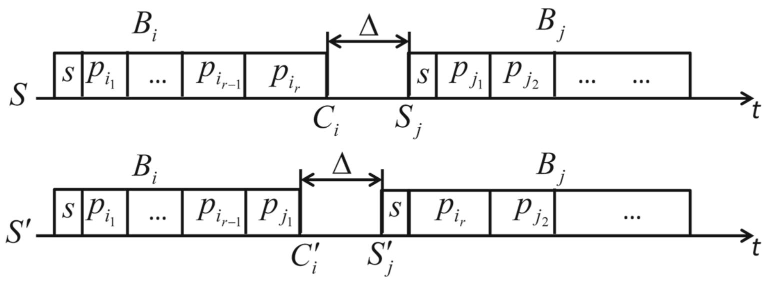

Fig. 1 Interchange of job J_{i_r} and job J_{j_1}

SPT order in each batch, batch $B_i = (J_{i_1}, J_{i_2}, \ldots, J_{i_r})$ precedes batch $B_j = (J_{j_1}, J_{j_2}, \ldots, J_{j_t})$ with $p_{i_r} > p_{j_1}$, and they are adjacent. The length of the idle time between B_i and B_j is denoted as Δ, and $\Delta \geq 0$.

Now, we interchange job J_{i_r} and job J_{j_1}. Let the processing of J_{j_1} start immediately after job $J_{i_{r-1}}$, and J_{i_r} be completed at the starting time of job J_{j_2}. Let S' denote this new schedule. This operation does not affect other batches except B_i and B_j. In schedules S and S', let C_i and C'_i denote the completion times of batch B_i, respectively, and let S_j and S'_j denote the starting times of batch B_j, respectively (see Fig. 1). $C'_i < C_i$, and the idle time has moved forward. S' and S have different objective values. The difference is

$$Z(S') - Z(S) = r(C'_i - C_i) + \sum_{k=C_i}^{S_j} \pi_k - \sum_{k=C'_i}^{S'_j} \pi_k < 0.$$

It is shown that the better schedule is S' rather than S, which contradicts the optimality of S.

Considering a given optimal schedule, we can reorder the jobs within each batch in SPT order. Then, for two arbitrary adjacent batches $B_i = (J_{i_1}, J_{i_2}, \ldots, J_{i_r})$ and $B_j = (J_{j_1}, J_{j_2}, \ldots, J_{j_t})$ (batch B_i precedes batch B_j), we have $p_{i_r} \leq p_{j_1}$. Consequently, we obtain an optimal SPT job sequence. □

According to Lemma 2, we first reorder the jobs in SPT order. Based on dynamic programming, we propose an algorithm to decide on the optimal partition of the SPT sequence into batches and the optimal time slot allocation for the batches.

Let $F(j, k)$ be the minimum total cost of the partial schedule including jobs J_1, $J_2, \ldots, J_j$ when the last job J_j is completed no later than time k. There are two cases for this partial schedule.

Case 1: J_j is completed at time k. In this case, let $H(j, k)$ denote the value of $F(j, k)$. Assume that J_h is the first job in the last batch in the partial schedule. Then, jobs $J_h, J_{h+1}, \ldots, J_j$ constitute the last batch and start their processing after a setup time s. According to Lemma 1, there exists no idle time in the last batch. For this batch, its starting time is $k - s - \sum_{r=h}^{j} p_r$, and the total cost is $(j - h + 1)k + \sum_{l=k-s-\sum_{r=h}^{j} p_r+1}^{k} \pi_l$. Then $H(j, k)$ can be recursively calculated by

$$H(j,k) = \min_{1\le h\le j}\left\{(j-h+1)k + \sum_{l=k-s-\sum_{r=h}^{j}p_r+1}^{k}\pi_l + F\left(h-1,k-s-\sum\nolimits_{r=h}^{j}p_r\right)\right\}. \quad (1)$$

Case 2: Job J_j is completed no later than time $k-1$.

Based on these two cases, the recursion of the dynamic programming can be given as

$$F(j,k) = \min\{H(j,k),\ F(j,k-1)\} \quad (2)$$

with the boundary conditions

$$F(j,k) = +\infty, \quad \text{if } k < s + \sum_{r=1}^{j} p_r. \quad (3)$$

The initial conditions are $F(0,k)=0$, for $k=1,2,\ldots,K$.

The total cost of the optimal schedule is obtained by $F^* = F(n,K)$. Retracing the solution backwards from the end, we can determine the optimal batch partition and time slot allocation.

Theorem 1 Problem $1|\text{batch, non-inc-slotcost}|\sum_{j=1}^{n} C_j + \sum_{i=1}^{m}\pi_{B_i}$ can be solved in $O(K^2)$ or $O(nK)$ time.

Proof The principle of dynamic programming and Lemmas 1–2 guarantee the correctness of the above algorithm. In the following, we analyze the time complexity. $O(n\log n)$ time is required by reordering the jobs in SPT order before implementing dynamic programming. The recursion in Eq. (2) takes $O(nK)$ time because there are $O(nK)$ states.

If the TSCs are input as K discrete values corresponding to the K time slots, the input size is $O(K)$. Then, preparing the values of $\sum_{l=a}^{b}\pi_l (1 \le a \le b \le K)$ before performing the recursion takes $O(K^2)$ time. In this case, the overall time complexity can be determined as $O(K^2)$ due to $n \le K$, and thus, the algorithm is of polynomial time.

If each TSC is provided as a function of time k with a closed form, the input size is $O(1)$. Then, obtaining each $\sum_{l=a}^{b}\pi_l$ value takes a constant time and requires no preparation. In this case, the overall time complexity can be determined as $O(nK)$, indicating that the algorithm is of pseudopolynomial time. □

Numerical example 1. Consider a four-job problem where the setup time is $s=1$, the processing times are $p_j=(3,2,1,1)$, the weights are $w_j=(10,15,3,8)$, the due dates are $d_j=(5,6,8,11)$, the number of time slots is $K=11$, and the TSCs are $\pi_k=(15,10,4,3,3,2,2,1,1,1,1)$.

Reorder the jobs in SPT order as J_4, J_3, J_2, J_1. Applying the above dynamic programming, the optimal batch partition and time slot allocation are obtained as $\{s, J_4, J_3\}_{(3-5)}$, $\{s, J_2, J_1\}_{(6-11)}$. $\{s, J_4, J_3\}_{(3-5)}$ represents the first batch, which occupies the 3rd to 5th time slots. $\{s, J_2, J_1\}_{(6-11)}$ represents the second batch, which occupies the 6th to 11th time slots. The total cost is $F^* = F(4, 11) = 50$. This solution shows that the first two time slots remain idle because their costs are much higher than those of the others. Without considering TSCs, by using the algorithm in paper [4], the optimal schedule is obtained as $\{\ s,\ J_4,\ J_3\}_{(1-3)}$,$\{\ s,\ J_2\}_{(4-6)}$,$\{\ s,\ J_1\}_{(7-10)}$. The total flow time is 22, and the total TSC is 42, so the total cost is 64. Comparing the optimal schedules in the two cases, we find that they have different batch partitions, and the optimal schedule considering TSCs can significantly reduce the total cost.

4 1|batch, non-inc-slotcost|$L_{\max} + \sum_{i=1}^{m} \pi_{B_i}$ and 1|batch, non-inc-slotcost|$T_{\max} + \sum_{i=1}^{m} \pi_{B_i}$

The objective of the problem discussed in this section is to minimize the sum of the total TSC and maximum lateness (or the maximum tardiness). We aim to present a unified algorithm to solve the two problems. Before that, we introduce the property of an optimal schedule as below.

Lemma 3. For problems 1|batch, non-inc-slotcost|$L_{\max} + \sum_{i=1}^{m} \pi_{B_i}$ and 1|batch, non-inc-slotcost|$T_{\max} + \sum_{i=1}^{m} \pi_{B_i}$, an optimal job sequence can be determined by sequencing the jobs in accordance with the EDD rule.

Proof These two problems are similar; hence, we only consider the problem containing $L_{\max}$. As mentioned above, the job sequence is immaterial in each batch. Assuming that S is an optimal schedule in which the jobs are sequenced in EDD order in each batch, batch $B_i = (J_{i_1}, J_{i_2}, \ldots, J_{i_r})$ precedes batch $B_j = (J_{j_1}, J_{j_2}, \ldots, J_{j_t})$ with $d_{i_r} > d_{j_1}$, and they are adjacent. Let $L(B_i) = \max\{L_{i_1}, L_{i_2}, \ldots, L_{i_r}\}$ and $L(B_j) = \max\{L_{j_1}, L_{j_2}, \ldots, L_{j_t}\}$. The jobs have identical completion times in each batch, so $L(B_i) = L_{i_1}$ and $L(B_j) = L_{j_1}$. According to the number of jobs in B_i, there are two cases.

Case 1: $r = 1$, meaning that there is only one job J_{i_1} in batch B_i. We have $L_{i_1} < L_{j_1}$ because $C_{i_1} < C_{j_1}$ and $d_{i_1} > d_{j_1}$, then $L(B_i) < L(B_j)$. Without changing the time when batch B_j is completed, we move job J_{i_1} and insert it in an appropriate position after job J_{j_1} in batch B_j so that the job sequence still keeps EDD order in batch B_j (see Fig. 2 (a)). Then, the batch B_i is eliminated. This operation reduces the total TSC but does not change $L(B_j)$. The total cost is therefore reduced.

Case 2: $r > 1$, meaning that batch B_i contains more than one job. Similar to Case 1, remove job J_{i_r} from batch B_i and insert it in an appropriate position after job J_{j_1} in batch B_j so that the job sequence still keeps the EDD order in batch B_j (see Fig. 2b). This new schedule is denoted by S'. Let Δ, C_i, C'_i,S_j, and S'_j be defined the same as those in the proof of Lemma 2. After this operation, the total TSC has not increased, and the completion time of batch B_i has been reduced, meaning that $L(B_i)$ has been reduced. $L(B_j)$ in S' and S have identical values because the completion time of batch B_j is not changed, and $d_{i_r} > d_{j_1}$. Hence, schedule S' is also optimal.

We can obtain an optimal job sequence in accordance with the EDD rule by repeating the procedure in Case 1 or 2 a finite number of times. □

Now, the jobs are assumed to have been arranged according to the EDD rule as $d_1 \le d_2 \le \cdots \le d_n$. For problem 1|batch, non-inc-slotcost|$L_{\max} + \sum_{i=1}^{m} \pi_{B_i}$, a

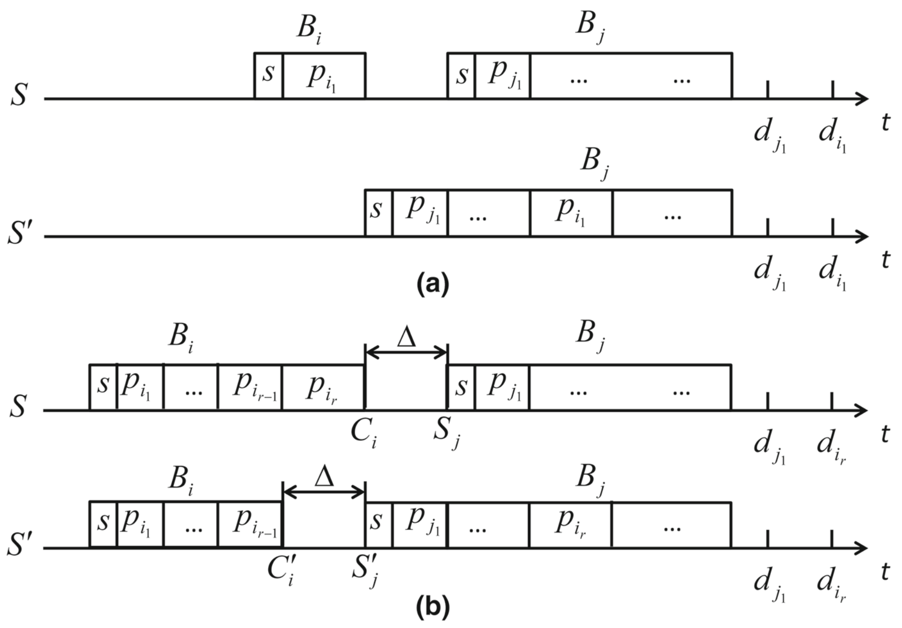

Fig. 2 Inserting job J_{i_r} into batch B_j: **a** Case 1 and **b** Case 2

dynamic programming algorithm is presented first to minimize the total TSC under constraint $L_{\max} \le 0$. $F(i, j, k)$ denotes the minimum total TSC of the partial schedule including jobs $J_1, J_2, \ldots, J_j$, where the last batch is composed of jobs $J_i, J_{i+1}, \ldots, J_j$, and where job J_j is completed no later than time k. Note that $\max\{L_i, L_{i+1}, \ldots, L_j\} = L_i$, and batch $J_i, J_{i+1}, \ldots, J_j$ should be processed as late as possible under constraint $L_{\max} \le 0$. $F(i, j, k)$ can then be recursively calculated by

$$F(i, j, k) = \begin{cases} F(i, j, k-1) & \text{if } k > d_i \\ \sum\limits_{l=k-s-\sum_{r=i}^{j} p_r+1}^{k} \pi_l + \min\limits_{1\le t\le i-1} F(t, i-1, k-s-\sum_{r=i}^{j} p_r) & \text{if } k \le d_i \end{cases} \tag{4}$$

with the boundary condition

$$F(i, j, k) = +\infty \text{ if} \begin{cases} i = 1 \text{ and } k < s + \sum_{r=1}^{j} p_r \\ i > 1 \text{ and } k < 2s + \sum_{r=1}^{j} p_r \end{cases}. \tag{5}$$

The initial conditions are

$$F(1,\ j,\ k) = \begin{cases} F(1,\ j,\ k-1) \text{ if } k > d_1 \text{ and } k > s + \sum_{r=1}^{j} p_r \\ \sum\limits_{l=k-s-\sum_{r=1}^{j} p_r+1}^{k} \pi_l \text{ if } s + \sum_{r=1}^{j} p_r \le k \le d_1\,. \\ +\infty \text{ otherwise} \end{cases} \tag{6}$$

Under the constraint $L_{\max} \le 0$, let $F(0)$ denote the minimum total TSC, and then

$$F(0) = \min_{1 \le i \le n} F(i,\ n,\ K). \tag{7}$$

The problem is infeasible if we get $F(0) = +\infty$. Because recursion Eq. (4) has $O(n^2 K)$ states, performing the above dynamic programming takes $O(n^2 K)$ time.

Next, the problem of minimizing the total TSC is taken into consideration under the constraint $L_{\max} \le t$. This problem is a generalization of the one under the constraint $L_{\max} \le 0$. For each job, a new due date is introduced and expressed as $d'_j = d_j + t$. Then, the constraint $L_{\max} \le t$ is converted to $L_{\max} \le 0$, and the problem under $L_{\max} \le t$ is equivalent to that under $L_{\max} \le 0$. Therefore, for any given t, we can solve the problem under $L_{\max} \le t$ by using the dynamic programming algorithm presented above.

Let $F(t)$ represent the minimum total TSC for the total TSC minimization problem under the constraint $L_{\max} \le t$. If this problem has an optimal schedule S_t, then $F(t)$ is equal to a finite value, and there are two cases:

Case 1: $L_{\max} = t$, namely, the maximum lateness in schedule S_t is equal to t. For problem $1|\text{batch, non-inc-slotcost}|L_{\max} + \sum_{i=1}^{m} \pi_{B_i}$, schedule S_t is feasible, and the objective value is $t + F(t)$.

Case 2: $L_{\max} = \tilde{t} < t$, namely, the maximum lateness in schedule S_t is less than t. S_t is also optimal for the total TSC minimization problem under the constraint $L_{\max} \le \tilde{t}$; then, $F(t) = F(\tilde{t})$. For problem $1|\text{batch, non-inc-slotcost}|L_{\max} + \sum_{i=1}^{m} \pi_{B_i}$, the objective value of S_t is $\tilde{t} + F(\tilde{t})$.

Based on the two cases, the optimal objective value of $1|\text{batch, non-inc-slotcost}|$ $L_{\max} + \sum_{i=1}^{m} \pi_{B_i}$ must be in the set $\{t + F(t) | t = t_0,\ t_0 + 1,\ \ldots,\ K - d_1\}$, in which t_0 is the minimum value of t satisfying that the total TSC minimization problem has a feasible schedule under the constraint $L_{\max} \le t_0$. Therefore, we can use the above algorithm based on dynamic programming to obtain all the $F(t)$ values by enumerating the t values for $t = t_0,\ t_0 + 1,\ \ldots,\ K - d_1$. The optimal objective value is

$$F_L^* = \min_t \{t + F(t) | t = t_0,\ t_0 + 1,\ \ldots,\ K - d_1\}. \tag{8}$$

Note that t_0 may be less than 0 because $L_{\max}$ may be negative but $T_{\max}$ is non-negative. Thus, the optimal objective value of $1|\text{batch, non-inc-slotcost}|T_{\max} + \sum_{i=1}^{m} \pi_{B_i}$ is

$$F_T^* = \min_t\{t + F(t)|t = 0, 1, \ldots, K - d_1\} \tag{9}$$

Theorem 2. The problems $1|\text{batch, non-inc-slotcost}|L_{\max} + \sum_{i=1}^m \pi_{B_i}$ and $1|\text{batch, non-inc-slotcost}|\ T_{\max} + \sum_{i=1}^m \pi_{B_i}$ can be solved in $O(n^2K^2)$ time.

Proof The algorithms for the problems have been given above. Next, we analyze its time complexity. First, we need to spend $O(n \log n)$ time reordering the jobs in EDD order. For $1 \leq a \leq b \leq K$, preparing the $\sum_{l=a}^b \pi_l$ values will take $O(K^2)$ time, if necessary. For a given t, we use dynamic programming to obtain $F(t)$ in time $O(n^2K)$. Then, after enumerating the different values of t in $O(K)$ iterations, we can solve the two problems using Eqs. (8) and (9), respectively. Therefore, the overall time complexity can be determined as $O(n^2K^2)$. □

Similar to the algorithm for $1|\text{batch, non-inc-slotcost}|\sum_{j=1}^n C_j + \sum_{i=1}^m \pi_{B_i}$, the above algorithm is of polynomial time when the TSCs are input as K discrete values. It is a pseudopolynomial time algorithm when each TSC is provided as a function of time k.

Numerical example 2. Considering the four-job problem in Sect. 3, the jobs have been sequenced in EDD and it can be deduced that $K - d_1 = 11 - 5 = 6$. Applying the above algorithm, we obtain $F(6) = 14$, $F(5) = 17$, $F(4) = 18$, $F(3) = 18$, $F(2) = 27$, $F(1) = 28$, $F(0) = +\infty$. Therefore, $F_T^* = F_L^* = \{t + F(t)|, t = 0, 1, \ldots, 6\} = 6 + F(6) = 20$, the maximum lateness and tardiness problems have identical optimal solutions. Retracing the solution backwards from the end, we obtain the optimal solution in which there is only one batch as $\{s, J_1, J_2, J_3, J_4\}_{(4-11)}$, and the first three time slots are idle.

5 $1|\text{batch, non-inc-slotcost}|\sum_{j=1}^{n} w_jU_j + \sum_{i=1}^m \pi_{B_i}$

The objective of the problem discussed in this section is to minimize the sum of the total TSC and the weighted number of tardy jobs. The property of an optimal schedule is introduced below.

Lemma 4. For problem $1|\text{batch, non-inc-slotcost}|\sum_{j=1}^{n} w_jU_j + \sum_{i=1}^m \pi_{B_i}$, an optimal sequence of on-time jobs can be determined by sequencing the on-time jobs in accordance with the EDD rule.

The proof is omitted because it is similar to that for Lemma 3.

In terms of tardy jobs, there are two general scenarios. In the first scenario, customers allow some jobs to be rejected or outsourced to another manufacturer, so the manufacturer will reject processing the tardy jobs to save costs. The weight w_j represents the penalty cost or outsourcing cost caused by job J_j being rejected. The tardy jobs do not yield any TSC. In the second scenario, the customer requires that all the jobs must be processed by the manufacturer and cannot be rejected or outsourced. In this case, the weight w_j represents the tardiness cost of job J_j. Because the tardiness costs are fixed, the tardy jobs can be backlogged and processed at the end of the schedule to economize the total TSC.

We consider these two scenarios in the following section. According to Lemma 4, all jobs are assumed to have been arranged according to the EDD rule as $d_1 \leq d_2 \leq \cdots \leq d_n$.

5.1 Tardy jobs are rejected

For $i \geq 1$, $F(i, j, k)$ denotes the minimum total cost of jobs J_1, J_2, ..., J_j, where J_i is the first on-time job in the last batch of the partial sequence $J_1, J_2, \ldots, J_j$, and the last on-time job in $J_i, J_{i+1}, \ldots, J_j$ is completed no later than time k. Two cases are presented as follows.

Case 1: The last on-time job in $J_i, J_{i+1}, \ldots, J_j$ is completed at time k. The value of $F(i, j, k)$ is denoted as $f^1(i, j, k)$ if J_j $(j \geq i)$ is an on-time job. If J_j $(j > i)$ is a tardy job, the value of $F(i, j, k)$ is denoted as $f^2(i, j, k)$.

Case 2: The last on-time job in $J_i, J_{i+1}, \ldots, J_j$ is completed no later than time $k - 1$.

According to the above definition, we have

$$F(i, j, k) = \begin{cases} \min\{F(i, j, k-1),\ f^1(i, j, k),\ f^2(i, j, k)\} \text{ for } i, k \geq 1; \\ +\infty \text{ for } i \geq 1 \text{ and } k = 0. \end{cases} \tag{10}$$

For $i = 0$ and an arbitrary value of k, $F(i, j, k)$ represents the summation of the tardy penalties when the jobs J_1, J_2, ..., J_j are all tardy and rejected, i.e.,

$$F(0, j, k) = \sum_{r=1}^{j} w_r. \tag{11}$$

Dynamic programming recursions are proposed for calculating $f^1(i, j, k)$ and $f^2(i, j, k)$. For $f^1(i, j, k)$, $k \leq d_i$ is a necessary condition for J_i to be an on-time job under batch availability. Note that J_j is an on-time job. If $i < j$, the last on-time job before job J_j is completed at time $k - p_j$. If $i = j$, a setup time s is required before processing J_j, and all the on-time jobs before the current batch are completed no later than $k - s - p_j$. Therefore, we have

$$f^1(i, j, k) = \begin{cases} \sum_{l=k-p_j+1}^{k} \pi_l + \min\{f^1(i, j-1, k-p_j),\ f^2(i, j-1, k-p_j)\} & \text{if } k \leq d_i \text{ and } i < j \\ \sum_{l=k-s-p_j+1}^{k} \pi_l + \min\limits_{0 \leq r \leq i-1} F(r, i-1, k-s-p_j) & \text{if } k \leq d_i \text{ and } i = j \end{cases} \tag{12}$$

with the boundary conditions $f^1(i, j, k) = +\infty$, if $k > d_i$ or $k < s + p_j$.

For $f^2(i, j, k)$, job J_j incurs a tardy penalty. We have

$$f^2(i, j, k) = w_j + \min\{f^1(i, j-1, k), f^2(i, j-1, k)\} \text{ if } k \le d_i \text{ and } i < j \tag{13}$$

with the boundary conditions $f^2(i, j, k) = +\infty$ if $k > d_i$, $i = j$, or $k < s + p_i$. The initial conditions are

$$f^1(1, 1, k) = \begin{cases} \sum_{l=k-s-p_1}^{k} \pi_l & \text{if } k \le d_1 \\ +\infty & \text{if } k > d_1 \end{cases} \tag{14}$$

and $f^2(1, 1, k) = +\infty$.

The optimal objective value can be calculated by $F^* = \min\limits_{0 \le i \le n} F(i, n, K)$. The above recursion equations have $O(n^2K)$ states, so dynamic programming can be implemented in $O(n^2K)$ time.

Theorem 3. Problem $1|\text{batch, non-inc-slotcost}|\sum_{j=1}^{n} w_j U_j + \sum_{i=1}^{m} \pi_{B_i}$ can be solved in $O(\max(n^2K, K^2))$ or $O(n^2K)$ time in the case that tardy jobs are rejected.

Proof The correctness of the above algorithm is obvious, so we only analyze the time complexity. $O(n \log n)$ time is required by reordering the jobs in EDD order. If the TSCs are input as K discrete values, preparing the $\sum_{l=a}^{b} \pi_l$ values will take $O(K^2)$ time for $1 \le a \le b \le K$. Then, we have to implement dynamic programming in $O(n^2K)$ time. Therefore, the overall complexity can be determined as $O(\max(n^2K, K^2))$, indicating that the algorithm is of polynomial time.

If each TSC is provided as a function of time k, then obtaining each $\sum_{l=a}^{b} \pi_l$ value takes a constant time and requires no preparation. In this case, the algorithm is of pseudopolynomial time, and its overall complexity can be determined as $O(n^2K)$. $\square$

Numerical example 3. Consider the four-job problem in Sect. 3. Applying the above algorithm, the total cost is obtained as $F^* = F(4, 4, 11) = 23$. Retracing the solution backwards from the end, we obtain the optimal batch partition and time slot allocation as $\{s, J_2\}_{(4-6)}$, $\{s, J_3\}_{(7-8)}$, and $\{s, J_4\}_{(10-11)}$. This solution shows that job J_1 is a rejected job, and the first three time slots and the $9th$ time slot are idle.

5.2 Tardy jobs are backlogged

Now, the tardy jobs are considered to be backlogged, and their processing needs to be completed by time K. In a schedule, let T denote the ending time of the last job. Then, we give the property of an optimal schedule.

Lemma 5 All of the jobs whose due dates are not less than T ($d_j \ge T$) and all of the tardy jobs constitute the last batch in an optimal schedule.

Proof All the jobs whose due dates are not less than T are on-time jobs without a tardiness penalty. Therefore, they can be processed as late as possible to reduce the total TSC. For the same purpose, all tardy jobs can also be processed as late as possible due to their fixed tardiness penalties. Furthermore, the two types of jobs should be combined to constitute the last batch to reduce the number of setups. Therefore, the lemma is valid. □

For all tardy jobs, let t' represent their total processing time. Now, we focus on the problem with the given t' and T. Let $c(0 \le c \le n)$ represent the number of jobs satisfying $d_j \ge T$. These on-time jobs are $J_{n-c+1}, J_{n-c+2}, \ldots, J_n$ in the EDD job sequence. According to Lemma 5, $J_{n-c+1}, J_{n-c+2}, \ldots, J_n$ and all the tardy jobs constitute the last batch, and the total processing time is $P{=}t' + \sum_{j=n-c+1}^{n} p_j$. Considering the setup time s, the total TSC of the last batch is $\sum_{l=T-s-P+1}^{T} \pi_l$. The on-time jobs in the partial sequence $J_1, J_2, \ldots, J_{n-c}$ are completed no later than $T' = T - s - P$. In this case, we consider the remaining problem of how to determine the tardy jobs, the partition of the on-time jobs in sequence $J_1, J_2, \ldots, J_{n-c}$ into batches, and the time slot allocation for batches. It is similar to the problem in Sect. 5.1. Thus, we now modify the dynamic programming in Sect. 5.1 to handle the remaining problem as follows.

Considering the partial sequence $J_1, J_2, \ldots, J_j$, $(j \le n - c)$, for $i \ge 1$, $F(i, j, k, t, T)$ represents the minimum total cost of $J_1, J_2, \ldots, J_j$ excluding the total TSC of all tardy jobs, where

1. the last batch in the partial sequence is formed by all on-time jobs in $J_i, J_{i+1}, \ldots, J_j$,
2. job J_i is the first on-time job in the last batch in $J_1, J_2, \ldots, J_j$,
3. the last on-time job in $J_i, J_{i+1}, \ldots, J_j$ is completed no later than time k,
4. the total processing time of all tardy jobs in sequence $J_1, J_2, \ldots, J_j$ is t,
5. the last batch of the entire sequence $J_1, J_2, \ldots, J_n$ is completed at time T.

There exist two cases:

Case 1: The last on-time job in $J_i, J_{i+1}, \ldots, J_j$ is completed at time k. The value of $F(i, j, k, t, T)$ is denoted as $f^1(i, j, k, t, T)$ if $J_j (j \ge i)$ is an on-time job and $f^2(i, j, k, t, T)$ if $J_j (j > i)$ is a tardy job.

Case 2: The last on-time job in $J_i, J_{i+1}, \ldots, J_j$ is completed no later than time $k - 1$.

By definition, we have.

$$F(i, j, k, t, T) = \begin{cases} \min\{F(i, j, k-1, t, T),\ f^1(i, j, k, t, T),\ f^2(i, j, k, t, T)\} \text{ for } i, k \ge 1 \\ +\infty \text{ for } i \ge 1 \text{ and } k = 0 \end{cases}. \quad (15)$$

In addition, we consider the case in which all jobs of $J_1, J_2, \ldots, J_j$ are tardy, i.e., they are all processed in the last batch of the entire schedule. Setting $i = 0$; if $t = \sum_{r=1}^{j} p_r$, let $F(i, j, k, t, T)$ represent the sum of the tardiness penalties of $J_1, J_2, \ldots, J_j$. Then, we have

$$F(0, j, k, t, T) = \begin{cases} \sum_{r=1}^{j} w_r \text{ if } t = \sum_{r=1}^{j} p_r \\ +\infty \text{ otherwise} \end{cases}. \quad (16)$$

Dynamic programming recursions are proposed for calculating $f^1(i, j, k, t, T)$ and $f^2(i, j, k, t, T)$ as below. For $f^1(i, j, k, t, T)$, we have

$$f^1(i, j, k, t, T) = \begin{cases} \sum_{l=k-p_j+1}^{k} \pi_l + \min\{f^1(i, j-1, k-p_j, t, T), f^2(i, j-1, k-p_j, t, T)\} & \text{if } k \le d_i \text{ and } i < j \\ \sum_{l=k-s-p_j+1}^{k} \pi_l + \min_{0 \le r \le i-1} F(r, i-1, k-s-p_j, t, T) & \text{if } k \le d_i \text{ and } i = j \end{cases} \tag{17}$$

with the boundary conditions

$$f^1(i, j, k, t, T) = +\infty \quad \text{if} \begin{cases} k > d_i \\ k < s + p_j \\ t < 0 \end{cases}. \tag{18}$$

For $f^2(i, j, k, t, T)$, job J_j incurs a tardiness penalty. It yields

$$f^2(i, j, k, t, T) = w_j + \min\{f^1(i, j-1, k, t-p_j, T), f^2(i, j-1, k, t-p_j, T)\} \text{ if } k \le d_i \text{ and } i < j \tag{19}$$

with the boundary conditions

$$f^2(i, j, k, t, T) = +\infty \quad \text{if} \begin{cases} k > d_i \\ k < s + p_i \\ t < p_j \\ i = j \end{cases}. \tag{20}$$

The initial conditions are

$$f^1(1, 1, k, t, T) = \begin{cases} \sum_{l=k-s-p_1}^{k} \pi_l & \text{if } s + p_1 \le k \le d_1 \text{ and } t = 0 \\ +\infty & \text{otherwise} \end{cases}. \tag{21}$$

and $f^2(1, 1, k, t, T) = +\infty$.

Given the value of t' and T, the minimum total cost of jobs $J_1, J_2, \ldots, J_{n-c}$ excluding the total TSC of all the tardy jobs is determined by

$$\min_{0 \le i \le n-c} F(i, n-c, T', t', T). \tag{22}$$

Adding the total TSC (i.e. $\sum_{l=T-s-P+1}^{T} \pi_l$) consumed by the last batch of the entire schedule to the above formula, we can obtain the total cost of the entire schedule as

$$\min_{0 \le i \le n-c} F(i, n-c, T', t', T) + \sum_{l=T-s-P+1}^{T} \pi_l. \tag{23}$$

Under the assumption that the last job is completed at time T, by enumerating different values of t' ($0 \leq t' \leq \sum_{r=1}^{n-c} p_r$) in $O(K)$ iterations, the problem can be solved by

$$F(T) = \min_{0\leq t'\leq \sum_{r=1}^{n-c} p_r} \{ \min_{0\leq i\leq n-c} F(i, n-c, T', t', T) + \sum_{l=T-s-P+1}^{T} \pi_l \} \tag{24}$$

If $F(T) = +\infty$, it shows that the problem is infeasible for time T.

Based on the above discussions, the problem $1|batch, non{-}inc{-}slot\cos t|\sum_{j=1}^{n} w_j U_j + \sum_{i=1}^{m} \pi_{B_i}$ in the case that tardy jobs are backlogged can be solved by enumerating different values of T in $O(K)$ iterations. Specifically, it can be solved by

$$F^* = \min\{F(T)|s + \sum_{r=1}^{n} p_r \leq T \leq K\} \tag{25}$$

To obtain the optimal solution, performing the above dynamic programming requires $O(n^2K^3)$ time. Therefore, the following theorem is obtained, of which the proof is omitted because it is similar to that of Theorem 3.

Theorem 4. Problem $1|batch, non{-}inc{-}slot\cos t|\sum_{j=1}^{n} w_j U_j + \sum_{i=1}^{m} \pi_{B_i}$ can be solved in $O(n^2K^3)$ time for the case in which tardy jobs are backlogged.

Numerical example 4. For the four-job problem in Sect. 3, $8 \leq T \leq 11$. Applying the above algorithm, we obtain $F(8) = 65$, $F(9) = 51$, $F(10) = 37$ and $F(11) = 28$. By Eq. (25), the total cost is obtained as $F^* = \min\{F(T)|8 \leq T \leq 11\} = F(11) = 28$. Retracing the solution backwards from the end, we obtain the optimal batch partition and time slot allocation as $\{s, J_2, J_3\}_{(3-6)}, \{s, J_1, J_4\}_{(7-11)}$. This solution shows that the first two time slots are idle.

6 Conclusions

Single machine batch scheduling problems with non-increasing TSCs arise from the continuous casting stage of steel production considering the time-of-use electricity prices. In this real production, one charge of molten steel is regarded as a job, a cast is regarded as a batch, and the time required to replace the tundish is regarded as the setup time. In a production planning period where electricity prices are non-increasing, all the TSCs can be obtained according to the machine power and electricity prices. To improve production efficiency and reduce electricity costs, the objectives are to minimize the sum of the total TSC and one traditional measure (such as the total flow time, maximum lateness or tardiness, and weighted number of tardy jobs). For the first two traditional measures, we first determine the optimal job sequence, and then the corresponding dynamic programming is proposed for determining the batch partition and the time slot allocation. For the weighted number of tardy jobs, two cases are taken into account, i.e., tardy jobs are rejected or backlogged. To minimize the objectives of these two cases, we first determine the optimal sequence of the on-time jobs and then give the corresponding dynamic programming for determining the tardy jobs, the batch partition of the on-time job, and the time slot allocation. All the algorithms presented in this paper are of polynomial time in the case that the TSC's input size is $O(K)$.

For future research, the batch scheduling problem with non-increasing TSCs can be extended to the case with release times because the jobs arrive dynamically in many real productions. In addition, problems with parallel machine environments or flowshop environments are also worth studying, as they often appear in modern industrial manufacturing systems.

Funding No funding was received to assist with the preparation of this manuscript.

Code availability Not applicable.

Declarations

Conflict of interest The authors have no conflicts of interest to declare that are relevant to the content of this article.

Availability of data and material Not applicable.

References

1. Allahverdi, A., Ng, C.T., Cheng, T.C.E., Kovalyov, M.Y.: A survey of scheduling problems with setup times or costs. Eur J Oper Res **187**, 985–1032 (2008). https://doi.org/10.1016/j.ejor.2006.06.060
2. Chen, L., Megow, N., Rischke, R., Stougie, L., Verschae, J.: Optimal Algorithms and a PTAS for Cost-Aware Scheduling. In: Italiano, G.F., Pighizzini, G., Sannella, D.T. (eds.) Mathematical foundations of computer science 2015, pp. 211–222. Springer, Berlin, Heidelberg (2015)
3. Chen, R., Qin, L., Tang, G.: Scheduling with outsourcing and variable time slot costs. J Math **35**, 1068–1074 (2015)
4. Coffman, E.G., Yannakakis, M., Magazine, M.J., Santos, C.: Batch sizing and job sequencing on a single machine. Ann Oper Res **26**, 135–147 (1990). https://doi.org/10.1007/BF02248589
5. Ghosh, J.B., Gupta, J.N.D.: Batch scheduling to minimize maximum lateness. Oper Res Lett **21**, 77–80 (1997). https://doi.org/10.1016/S0167-6377(97)00028-X
6. Hochbaum, D.S., Landy, D.: Scheduling with batching: minimizing the weighted number of tardy jobs. Oper. Res. Lett. **16**, 79–86 (1994). https://doi.org/10.1016/0167-6377(94)90063-9
7. Lu, S., Liu, X., Pei, J.T., Thai, M.M., Pardalos, P.: A hybrid ABC-TS algorithm for the unrelated parallel-batching machines scheduling problem with deteriorating jobs and maintenance activity. Appl Soft Comput **66**, 168–182 (2018). https://doi.org/10.1016/j.asoc.2018.02.018
8. Lu, S., Pei, J., Liu, X., Pardalos, P.M.: A hybrid DBH-VNS for high-end equipment production scheduling with machine failures and preventive maintenance activities. J. Comput. Appl. Math. **384**, 113195 (2021). https://doi.org/10.1016/j.cam.2020.113195
9. Lu, S., Pei, J., Liu, X., Qian, X., Mladenovic, N., Pardalos, P.M.: Less is more: variable neighborhood search for integrated production and assembly in smart manufacturing. J Sched. **23**, 649–664 (2020). https://doi.org/10.1007/s10951-019-00619-5
10. Mosheiov, G., Oron, D., Ritov, Y.: Minimizing flow-time on a single machine with integer batch sizes. Oper. Res. Lett. **33**, 497–501 (2005). https://doi.org/10.1016/j.orl.2004.09.007

11. Potts, C.N., Kovalyov, M.Y.: Scheduling with batching: A review. European J. Oper. Res. **120**, 228–249 (2000). https://doi.org/10.1016/S0377-2217(99)00153-8
12. Quadt, D., Kuhn, H.: Batch scheduling of jobs with identical process times on flexible flow lines. Int. J. Prod. Econ. **105**, 385–401 (2007). https://doi.org/10.1016/j.ijpe.2004.04.013
13. Tang, L., Wang, G.: Decision support system for the batching problems of steelmaking and continuous-casting production. Omega **36**, 976–991 (2008). https://doi.org/10.1016/j.omega.2007.11.002
14. Wan, G., Qi, X.: Scheduling with variable time slot costs. Naval Res. Logistics. **57**, 159–171 (2010). https://doi.org/10.1002/nav.20393
15. Wang, K.: Calibration scheduling with time slot cost. Theoret. Comput. Sci. **821**, 1–14 (2020). https://doi.org/10.1016/j.tcs.2020.03.018
16. Webster, S., Baker, K.R.: Scheduling Groups of Jobs on a Single Machine. Oper. Res. **43**, 692–703 (1995). https://doi.org/10.1287/opre.43.4.692
17. Yuan, J.J., Lin, Y.X., Cheng, T.C.E., Ng, C.T.: Single machine serial-batching scheduling problem with a common batch size to minimize total weighted completion time. Int. J. Prod. Econ. **105**, 402–406 (2007)
18. Zhao, Y., Qi, X., Li, M.: On scheduling with non-increasing time slot cost to minimize total weighted completion time. J Sched. **19**, 759–767 (2016). https://doi.org/10.1007/s10951-015-0462-9
19. Zhong, W., Liu, X.: A Single Machine Scheduling Problem with Time Slot Costs. In: Qian, Z., Cao, L., Su, W., Wang, T., Yang, H. (eds.) Recent Advances in Computer Science and Information Engineering, pp. 677–681. Springer, Berlin Heidelberg, Berlin, Heidelberg (2012)