Journal of Scheduling (2024) 27:227–237
https://doi.org/10.1007/s10951-022-00763-5

Single machine scheduling with step-learning

Matan Atsmony[1] · **Baruch Mor**[2] · **Gur Mosheiov**[1,3]

Accepted: 14 October 2022 / Published online: 17 December 2022

Abstract
In this paper, we study scheduling with step-learning, i.e., a setting where the processing times of the jobs started after their job-dependent learning-dates are reduced. The goal is to minimize makespan on a single machine. We focus first on the case that idle times between consecutive jobs are not allowed. We prove that the problem is NP-hard, implying that no polynomial-time solution exists and, consequently, propose a pseudo-polynomial time dynamic programming algorithm. An extensive numerical study is provided to examine the running time of the algorithm with different learning-dates and job processing time ranges. The special case of a common learning-date for all the jobs is also studied, and a (more efficient) pseudo-polynomial dynamic programming is introduced and tested numerically. In the last part of the paper, the more complicated setting in which idle times are allowed is studied. An appropriate dynamic programming is introduced and tested as well.

Keywords Scheduling · Single-machine · Step-learning · Job-dependent learning-dates · Dynamic programming

1 Introduction

In traditional scheduling theory, the processing times of the jobs were assumed to be constants dictated in advance. But as scheduling theory evolved, this concept was revised and many studies have proposed variable job processing times to reflect real-life circumstances. As claimed by Gawiejnowicz (2008, p. 49): "Scheduling with variable job processing times has numerous applications, e.g., in the modeling of the forging process in steel plants, manufacturing of pre-heated parts in plastic molding or in silverware production, finance management and scheduling maintenance or learning activities." The importance of variable job processing times is reflected in the comprehensive review of Gawiejnowicz (2020b) and in the recently published book of Gawiejnowicz (2020a), both summarizing the research conducted over the past four decades in the domain of time-dependent scheduling (deterioration effect and learning effect alike), where jobs' processing times depend on their starting time.

✉ Baruch Mor
baruchm@ariel.ac.il

[1] School of Business Administration, The Hebrew University, 91905 Jerusalem, Israel

[2] Department of Economics and Business Administration, Ariel University, 40700 Ariel, Israel

[3] Jerusalem College of Technology, Lev Academic Center, Jerusalem, Israel

The most prevalent aspects of variable job processing times are deterioration and/or learning effects. In this study, we concentrate on learning effects. The importance of the learning process is its implications for manufacturing routines. Empirical research has confirmed that learning-by-doing increases the productivity at the single worker level and moreover at the team level. The learning effects are manifested by enhanced productivity of the production system, decreased operational expense, and faster time-to-market, thus improving the sustainability of the business. A very recent paper by Azzouz et al. (2018) summarizes recent developments in the concept of learning effects and presents an overview of the significant learning models, a classification scheme for scheduling under learning effects and a mapping of the relations between major models. Recent published studies on learning effects include Gao et al. (2018), Wu et al. (2018), Fu et al. (2019), Geng et al. (2019), Mor et al. (2020), Mousavi et al. (2018), Renna et al. (2019), Sun et al. (2019), Wang et al., (2019a, 2019b), Yan et al. (2019), Azzouz et al. (2020) and Wu et al. (2020).

An important model for time-dependent job processing times is that of *step-deterioration*, which was first proposed by Mosheiov (1995), and suggested that the processing time of a job follows a step function of its starting time. More specifically, the actual processing times of the jobs that start after their deterioration-dates experience a step increase. The author focused on minimizing makespan with a single step-deterioration in a single- and multi-machine settings, presented NP-completeness justification, integer programming

formulation and provided a heuristic procedure. An abundance of studies was published subsequently, assuming common deterioration-date or job-specific deterioration-dates, various step-functions and machine settings; see Gawiejnowicz (2008) and Strusevich and Rustogi (2017). Recent papers addressing step-deterioration and other deterioration models include Cheng et al. (2018), Rostami (2018), Woo (2018), Ding et al. (2019), Liu et al. (2019), Miao and Zhang (2019), Pei et al. (2019), Sun and Geng (2019), Wang et al., (2019a, 2019b),Wu et al. (2019), Gawiejnowicz and Kurc (2020) and Soper and Strusevich (2020). Considering this wide research on step-deterioration, it is surprising that the complementary phenomenon, i.e., that of *step-learning* has yet to be discussed, let alone studied. Step-learning can be encountered in many real-life situations, e.g., when improved routines are assimilated in the production process, enhanced raw materials are utilized, faster equipment replaces outdated models, and, ultimately, when disruptive technology emerges and revolutionizes industry. We note that in many cases, there are no predicted times for the improvement procedures. However, the classical solid-state electronics based on the transistor technology which was introduced during the sixties of the previous century is a good example. This new technology replaced the vacuum tubes at predicted times and with predicted outcomes. Another example is flash memory, i.e., a non-volatile computer medium that can be electrically erased and reprogrammed anywhere and anytime, vs. ROM memory which was used previously and could only be programmed once and only at the original manufacturer facilities. Thus, the significance of incorporating step-learning into scheduling theory is two-fold, both theoretical and practical.

In the context of scheduling theory, there are two main approaches to formulating learning-effects, i.e., time-dependent or position-dependent job processing times. In this paper, we focus on a single-machine scheduling setting with both *time-dependent* and *job-dependent* step-learning effect. The assumption is that each job has its own Learning-Date (LD), such that if the starting time of a job is equal to or greater than this value, its actual processing time is reduced by a job-dependent factor. In the first model studied here, *no idle time* between consecutive jobs is permitted, an assumption which is justified in numerous manufacturing systems, due to the cost of stopping and renewing the production process. The objective function is minimum makespan. The problem is proved to be NP-Hard, and a pseudo-polynomial dynamic programming (DP) algorithm is introduced and tested. Our numerical tests indicate that medium-size and large problems (of up to 175 jobs) are solved in reasonable running time. We also study the special case in which all the jobs share a Common Learning-Date, denoted CONLD. This problem is NP-Hard as well, and a more efficient dynamic programming is proposed, as reflected in our numerical tests. In the last model we study, idle times between consecutive jobs are permitted. For this more complex setting (which can clearly lead to smaller makespan values), a more complicated pseudo-polynomial dynamic programming algorithm is proposed. In this case, the computational effort is much larger, and the running time required for solving smaller problems (of up to 70 jobs) increases significantly.

The paper is organized as follows. In Sect. 2, we present the notation and formulation. Section 3 is dedicated to the case of job-dependent learning-dates with no idle time between consecutive jobs. First, we prove that the problem is NP-hard, and then we introduce the DP algorithm. The special case of a common learning-date is studied in Sect. 4. In Sect. 5, we introduce the DP for the setting that idle times are permitted. In Sect. 6, we report the results of our numerical tests for all three DPs. Section 7 contains concluding remarks and topics for challenging future research.

1.1 Notations and formulation

Formally, a set $\mathcal{J}$ containing n jobs is to be processed on a single machine, with the jobs ready for processing at time zero and no idle times and no preemption allowed. The basic (maximal) processing time of job $j : j \in \mathcal{J}$, is denoted by u_j and the reduced (minimal) processing time is denoted by v_j, such that $u_j \geq v_j$ and $u_j,\ v_j \in Z^+$. We also denote by $u_{\max} = \max_{j\in\mathcal{J}}\{u_j\}$, the maximal basic processing time among all jobs.

For a given schedule, the starting time of job $j : j \in \mathcal{J}$, is denoted by S_j and the completion time of job $j : j \in \mathcal{J}$, is denoted by C_j. In this study, we focus on the makespan, i.e., the completion time of the last job to leave the production line, defined as $C_{\max} = \max_{j\in\mathcal{J}}\{C_j\}$.

We denote by $LD_j \in Z^+$ the learning-date of job $j : j \in \mathcal{J}$, and by $LD_{\min} = \min_{j\in\mathcal{J}}\{LD_j\}$, the minimal learning-date in set $\mathcal{J}$. If the starting time of job j is strictly less than LD_j, then its processing time is u_j, whereas if the starting time is equal to or greater than LD_j, its processing time is v_j. Eventually, the actual processing time of job j is defined as.

$$p_j = \begin{cases} u_j,\ S_j < LD_j \\ v_j,\ S_j \geq LD_j \end{cases},\ j \in \mathcal{J}$$

If the processing of job j starts before its job-dependent learning-date, LD_j, it is regarded as an early job, and otherwise if the processing starts at or after the LD_j, it is considered a late job. Consequently, we denote by $\mathcal{J}^E$ and $\mathcal{J}^L$, the subsets of early and late jobs, respectively, such that $\mathcal{J} = \mathcal{J}^E \cup \mathcal{J}^L$ and $\mathcal{J}^E \cap \mathcal{J}^L = \varnothing$. In the first setting studied here no idle time between consecutive jobs is permitted (and this constraint is denoted by *No-Idle*).

Utilizing the standard 3-field formulation of scheduling problems (Graham et al., 1979), the problem studied here, denoted by $\mathcal{Q}1$, is:

$$\mathcal{Q}\mathbf{1}:\ 1\big|LD_j,\ p_j \in \{u_j, v_j : u_j \geq v_j\},\ No-Idle\big|C_{\max}.$$

We then focus on the special case of a common learning-date (CONLD), i.e., $LD_j = LD,\ j \in \mathcal{J}$. For this setting, the subset of jobs that start their production before LD are regarded as early jobs, whereas the subset of jobs that start their production exactly at or after LD are considered as late jobs. The CONLD problem, denoted $\mathcal{Q}\mathbf{2}$, is the following:

$$\mathcal{Q}\mathbf{2}:\ 1\big|LD_j = LD,\ p_j \in \{u_j, v_j : u_j \geq v_j\},\ No-Idle\big|C_{\max}.$$

In the last setting studied here, all the features of problem $\mathcal{Q}1$ are relevant excluding the *No-Idle* time constraint. Hence, the problem, denoted $\mathcal{Q}3$ is the following:

$$\mathcal{Q}\mathbf{3}:1\big|LD_j,\ p_j \in \{u_j, v_j : u_j \geq v_j\}\big|C_{\max}$$

1.2 Minimizing makespan with job-dependent learning-dates and no idle-time

We first prove that Problem $\mathcal{Q}1$ is NP-Hard. In fact, we prove that even the special case of a common learning date (i.e., Problem $\mathcal{Q}2$) is NP-hard.

Theorem 1 Problem $\mathcal{Q}1$ is NP-Hard even for a common learning-date.

Proof Assume that all the jobs share a common learning-date denoted by LD. □

We formulate the problem as an integer linear program (ILP). Let X_j be a binary variable: $X_j = 1$ if job j starts processing at or after LD; $X_j = 0$ otherwise.

$u_j - v_j$ is the reduction in the processing time of job j due to learning (i.e., if job j starts processing at or after LD; $j = 1, \ldots, n$).

The objective function is minimum makespan, given by: $\sum_{j=1}^{n} u_j - \sum_{j=1}^{n} X_j(u_j - v_j)$, or, equivalently, maximum reduction in processing time: $\sum_{j=1}^{n} X_j(u_j - v_j)$.

Thus, the ILP for minimizing makespan on a single machine with step-learning and a common learning-date is:

$$\max \sum_{j=1}^{n} X_j(u_j - v_j)$$
$$s.t.\ \sum_{j=1}^{n} (1 - X_j)u_j \geq LD$$
$$X_j \text{ binary};\ j = 1, \ldots, n.$$

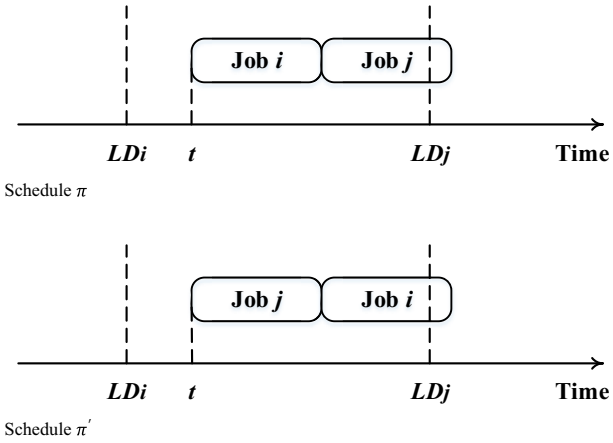

Fig. 1 Schedules π and $\pi\prime$ in the proof of property 1

This formulation is equivalent to:

$$\max \sum_{j=1}^{n} X_j(u_j - v_j)$$
$$s.t.\ \sum_{j=1}^{n} u_j \sum_{j=1}^{n} X_j u_j \geq LD$$
$$X_j \text{ binary};\ j = 1, \ldots, n.$$

Define $W = \sum_{j=1}^{n} u_j - LD$ and $P_j = u_j - v_j$
We obtain:

$$\max \sum_{j=1}^{n} X_j P_j$$
$$s.t. \sum_{j=1}^{n} X_j u_j \leq W$$
$$X_j \text{ binary};\ j = 1, \ldots, n.$$

The latter formulation is that of the well-known NP-Hard knapsack problem. □

In this section we introduce a pseudo-polynomial DP algorithm (denoted $\boldsymbol{DP}1$), thus establishing that $\mathcal{Q}1$ is NP-hard in the ordinary sense. In order to do this, we prove in the following a number of properties of an optimal schedule by performing adjacent pairwise interchange.

Property 1 An optimal schedule exists such that all the early jobs are scheduled prior to the late jobs.

Proof Consider an optimal schedule π in which job j follows job i, job i is late and job j is early. (If job i in π starts at time t, it follows that $LD_i < t < S_j = t + v_i < LD_j$.) We create a schedule π' by swapping these two jobs; see Fig. 1. It is clear that in π' job i (which is scheduled later) remains late, and job j (which is scheduled earlier) remains early. Hence, the total processing time of job i and j is unchanged, implying that π' is optimal as well. □

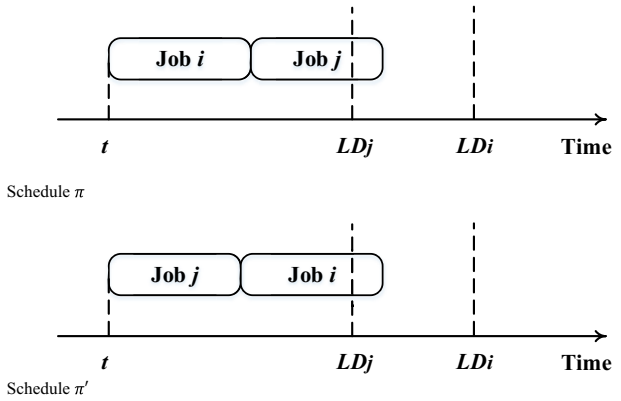

Fig. 2 Schedules π and $\pi\prime$ in the proof of property 2

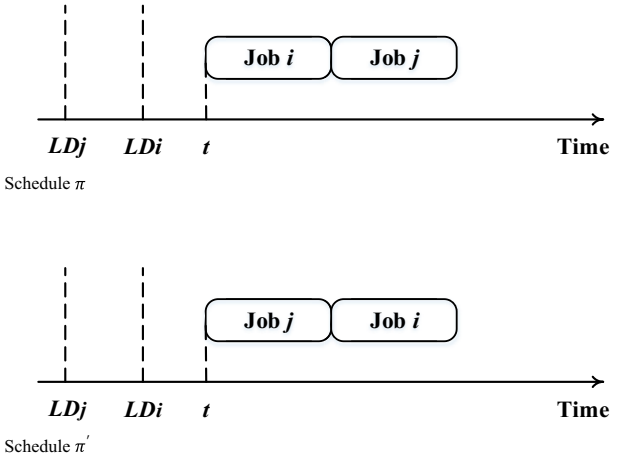

Fig. 3 Schedules π and $\pi\prime$ in the proof of property 3

Property 2 An optimal schedule exists such that all the early jobs are scheduled according to the Earliest Learning-Date first (ELD) rule.

Proof Consider an optimal schedule π in which job j follows job i, both jobs are early, and $LD_j < LD_i$. (If job i in π starts at time t, it follows that $S_j = t + u_i < LD_j < LD_i$.) We create a schedule π' by swapping these two jobs; see Fig. 2. It is clear that in π' job j (which is scheduled later in π) remains early. Job i is scheduled later in $\pi\prime$, and there are two options: (i) job i becomes a late job and its processing is reduced to v_i, (ii) job i remains an early job. In case (i) the total processing time of jobs i and j is reduced and therefore the makespan of π' is strictly smaller than that of π. (Note however that in π', job j is early and i is late). In case (ii) the total processing time of jobs i and j is unchanged, implying that π' is optimal as well. □

Property 3 An optimal schedule exists such that all the late jobs are scheduled according to the Earliest Learning-Date first (ELD) rule.

Proof Consider an optimal schedule π in which job j follows job i, both jobs are late, and $LD_j < LD_i$. (If job i in π starts at time t, it follows that $LD_j < LD_i \leq t$.) We create a schedule π' by swapping these two jobs; see Fig. 3. It is clear that in π' both jobs i and j remain late. Hence, in π' their total processing time is unchanged, implying that π' is optimal as well. □

Based on the above properties, an optimal schedule exists such that all the early jobs are scheduled prior to all the late jobs, and both sets are ordered according to ELD. Hence, we begin by sorting the jobs in an ELD order. At a superficial glance, Problem $\mathcal{Q}1$ is similar to 0/1 Knapsack, but this is true only if the last early job is guaranteed to be completed at a specific learning-date (equivalent to the knapsack size). Unfortunately, this is not guaranteed in our case. The last early job may be a *crossover* job, i.e., it may start strictly before its learning-date and completes after it. This phenomenon adds considerably to the complexity of $\mathcal{Q}1$, since each feasible completion time of the early set results in a different set of late jobs and therefore diverse values of makespan can be obtained. Consequently, as potentially each job can be the crossover job, we have to check all feasible schedules, where the last early job is completed not earlier than $LD_{\min}$ (recall that $LD_{\min} = \min_{j\in\mathcal{J}}\{LD_j\}$), and not later than $T^E_{\max} \equiv \max_{j\in\mathcal{J}}\{LD_j + u_j - 1\}$, i.e., in the interval $[LD_{\min}, T^E_{\max}]$.

Let $T^E \in [LD_{\min}, T^E_{\max}]$ denote the plausible completion time of all feasible early subsets. Since all feasible values need to be checked, we set T^E to an integer value in this interval and execute $\boldsymbol{DP}1$ for this T^E value. In each iteration of $\boldsymbol{DP}1$, a single job is handled and the updated makespan is computed accordingly. If job j, $j \in \mathcal{J}$, is early, then the makespan remains unchanged (since this job is completed not later than T^E), whereas if job j is late, the updated makespan is increased by v_j (job j is scheduled after T^E). The optimal solution is achieved by $\boldsymbol{DP}1$ that obtains the minimal makespan among all executions.

We define the following state variables:

j-the number of jobs already handled in the *partial* set $\mathcal{J} = \{1, \ldots, j\}$, $1 \leq j \leq n$.

t-the completion time of the last early job contained in the *partial* subset $\mathcal{J}^E : \mathcal{J}^E \subseteq \mathcal{J}$, $0 \leq t \leq T^E$.

Let $f_{T^E}(j, t)$ denote the optimal makespan of the partial schedule of jobs $1, \ldots, j$, given that the current completion time of the last early job is t. In each execution of $\boldsymbol{DP}1$, we set $f(0, 0) = T^E$, and in each iteration of $\boldsymbol{DP}1$, we have to decide between the above-mentioned two complementary options:

- If job j is early, i.e., $S_j(=t-u_j) < LD_j$ and $t \le T^E$, then job j can be either appended to subset $\mathcal{J}^E$, i.e., assigned to the last position in this subset with actual processing time $p_j = u_j$, or assigned to the last position in subset $\mathcal{J}^L$ with $p_j = v_j$. If we decide that the job is early, the makespan is not updated and otherwise, the makespan is incremented by v_j.
- Otherwise (if job j is late, i.e., $S_j \ge LD_j$), it can only be added to set $\mathcal{J}^L$ with $p_j = v_j$ and the makespan is incremented accordingly.

Based on the above, we present the following formal recursion:

Algorithm ***DP***1

$$f_{T^E}(j,t)=\begin{cases}\min\begin{cases}f_{T^E}(j-1,t-u_j)\\ f_{T^E}(j-1,t)+v_j\end{cases}, & t\ge u_j \text{ and } S_j<LD_j \text{ and } t\le T^E\\ f_{T^E}(j-1,t)+v_j, & t<u_j \text{ or } S_j\ge LD_j\\ \infty, & t>T^E \text{ or } \left(j=n \text{ and } t\ne T^E\right)\end{cases}$$

In the first line, i.e., if $S_j < LD_j$ *and* $t \le T^E$, then job j is early and can be added either to the set $\mathcal{J}^E$ or to the set $\mathcal{J}^L$. The second line reflects the case that job j is late and therefore must be added to set $\mathcal{J}^L$. The infinite cost in the third line reflects two infeasible cases: (i) the completion time of job j exceeds T^E, and (ii) the completion time of the last early job is not equal to T^E. The latter condition guarantees no idle time between the last early job and the first late job.

The boundary conditions are:

$f_{T^E}(0,\,0) = T^E$; The initial makespan value is the completion time of the last early job, T^E.

$f_{T^E}(0,\,t) = \infty,\ 1 \le t \le T^E$; For $j = 0$, a positive t value is infeasible.

$f_{T^E}(j,\,0) = f_{T^E}(j-1,\,0) + v_j,\ j = 1, \ldots, n$; There are no early jobs, and therefore job j is late.

The optimal solution (for a given T^E value) is given by $f_{T^E}(n,\,T^E)$.

Comment: Note that if at the end of the process, $t < T^E$ (i.e., the final value of t is strictly smaller than T^E), then it is clear that this solution is never optimal since there is a gap between the actual completion time of the last early job (t), and the starting time of the first late job (T^E), and the same schedule with no such gap is always better. Moreover, this schedule is not feasible due to the no-idle-time constraint.

The global optimum is: $f^* = \min\{f_{T^E}(n,\,T^E);\ LD_{\min} \le T^E \le T^E_{\max}\}$.

Theorem 2 Problem $\mathcal{Q}1$ is solved in $O\left(n\left(T^E_{\max}\right)^2\right)$ time.

Proof The recursive function in ***DP***1 is calculated for every job $j : 1 \le j \le n$, and $t : 0 \le t \le T^E$, $LD_{\min} \le T^E \le T^E_{\max}$ resulting in maximal running time of $O\left(nT^E_{\max}\right)$. ***DP***1 is executed for every integer value in the interval $\left[LD_{\min},\ T^E_{\max}\right]$, implying that the total computational effort is $O\left(n\left(T^E_{\max}\right)^2\right)$. □

Numerical Example 1 Consider an 8-job problem, where jobs are already sequenced in ELD order.

The job-dependent learning-dates are: LD_j = (130, 132, 135, 137, 147, 155, 176, 218), thus $LD_{\max} = \max_{j\in\mathcal{J}}\{LD_j\} = 218$.

The maximal (u_j) and minimal (v_j) processing times were generated uniformly in the intervals [25, 50] and [1, 25], respectively.

The generated processing times are: u_j = (31, 33, 33, 47, 30, 47, 29, 32) and v_j = (11, 21, 18, 15, 6, 19, 4, 5), implying that $u_{\max} = \max_{j\in\mathcal{J}}\{u_j\} = 47$ and $T^E_{\max} = \max_{j\in\mathcal{J}}\{LD_j + u_j - 1\} = 249$.

Executing ***DP***1, we achieved the following optimal solution with $\left(T^E\right)^* = 145 < 249 = T^E_{\max}$.

The resulting ordered set of early jobs is $\mathcal{J}^E$ = (2, 3, 6, 8), implying that (one) optimal sequence is $\mathcal{J}^*$ = (2, 3, 6, 8, 1, 4, 5, 7).

With regard to set $\mathcal{J}^*$, the actual job processing times are p_j = (33, 33, 47, 32, 11, 15, 6, 4).

We note that in this case there is no crossover job as the completion time of the last early job is $C_8 = 145 = \left(T^E\right)^*$, and the optimal makespan is $C^*_{\max} = 181$.

1.3 Minimizing makespan with a common learning-date

As proven in Sect. 3, Problem $\mathcal{Q}1$ is NP-hard even if $LD_j = LD,\ j : j \in \mathcal{J}$, thus $\mathcal{Q}2$ is NP-hard. We introduce a DP algorithm (denoted ***DP***2), implying that $\mathcal{Q}2$ is likewise ordinary NP-hard. Unlike ***DP***1, in this special case there is no need for an initial sorting and we only have to check the completion time of the last early job in a much smaller interval around the learning-date, i.e., $[LD-1,\ LD-1+u_{\max}]$.

As above, let $T^E_{\max} = LD + u_{\max} - 1$ denote the maximal completion time of all early subsets, and let $T^E \in \left[LD - 1,\ T^E_{\max}\right]$ denote the possible completion time of all feasible early subsets. Again, we have to check all feasible values in the interval $[LD - 1,\ LD + u_{\max} - 1]$, and therefore set T^E to an integer value in this interval, and execute $\boldsymbol{DP}2$ for each such value. Similar to $\boldsymbol{DP}1$, in each iteration of $\boldsymbol{DP}2$, a single job is handled and the updated makespan is computed accordingly. If job $j : j \in \mathcal{J}$ is early then the makespan remains unchanged, whereas if job j is late, the updated makespan is increased by v_j. The optimal solution is achieved by $\boldsymbol{DP}2$ that obtains the minimal makespan among all $u_{\max} + 1$ iterations.

While $\boldsymbol{DP}2$ is much more efficient than $\boldsymbol{DP}1$ (see below), the definitions of the state variables, the return function and the recursion are almost identical. For $\boldsymbol{DP}2$, we define the same state variables:

j-the number of jobs already handled in the *partial* set $\mathcal{J} = \{1, \ldots, j\}$, $1 \le j \le n$.

t-the completion time of the last early job contained in the *partial* subset $\mathcal{J}^E : \mathcal{J}^E \subseteq \mathcal{J}$, $0 \le t \le T^E$.

The definition of the return function, $f(j, t)$ is identical to that given in Sect. 3: it denotes the optimal makespan of the partial schedule of jobs $1, \ldots, j$, given that the current completion time of the last early job is t. In each execution of $\boldsymbol{DP}2$, we set $f(0, 0) = T^E$ and in each iteration of the algorithm, we have to decide between two options:

- If job j is early, then job j can be either appended to subset $\mathcal{J}^E$, or assigned to any position in subset $\mathcal{J}^L$. If job j is added to $\mathcal{J}^E$, the makespan is not updated and otherwise, it increases by v_j.
- If job j is late, it can only be added to set $\mathcal{J}^L$, and the makespan increases by v_j.

The options in the recursion are based on the starting time of the current job (S_j), the completion time of the last early job (T^E), and the *common* learning-date (LD):

Algorithm $\boldsymbol{DP}2$

$$f_{T^E}(j, t) = \begin{cases} \min\begin{cases} f_{T^E}(j-1,\ t-u_j) \\ f_{T^E}(j-1,\ t)+v_j \end{cases}, & t \ge u_j \text{ and } S_j < LD_j \text{ and } t \le T^E \\ f_{T^E}(j-1,\ t)+v_j, & t < u_j \text{ or } S_j \ge LD \\ \infty, & t > T^E \text{ or } \left(j = n \text{ and } t \ne T^E\right) \end{cases}$$

As in $\boldsymbol{DP1}$, Option 1 reflects the case of an early job (which can either be added to the set $\mathcal{J}^E$ or to the set $\mathcal{J}^L$), Option 2 reflects the case that the job is late (and is added to set $\mathcal{J}^L$), and the ∞ value in Option 3 avoids the infeasible cases.

The boundary conditions (similar to the above) are:

$$f_{(T^E)}(0,\ 0) = T^E,$$
$$f_{(T^E)}(0,\ t) = \infty,\ 1 \le t \le T^E,$$
$$f_{(T^E)}(j,\ 0) = f_{(T^E)}(j-1,\ 0) + v_j,\ j = 1, \ldots, n;$$

The optimal solution (for a given T^E value) is given by $f_{T^E}(n,\ T^E)$, and the global optimum is: $f^* = \min\{f_{T^E}(n,\ T^E);\ LD - 1 \le T^E \le T^E_{\max}\}$.

Theorem 3 Problem $\mathcal{Q}2$ is solved in $O\left(nu_{\max}T^E_{\max}\right)$ time**.**

Proof The recursive function of $\boldsymbol{DP}2$ is calculated for every job $j : 1 \le j \le n$, and $t : 0 \le t \le T^E$, $LD - 1 \le T^E \le T^E_{\max}$ resulting in maximal running time of $O\left(nT^E_{\max}\right)$. $\boldsymbol{DP}2$ is executed for every integer value in the interval $[1,\ u_{\max}]$, implying that the total computational effort is $O\left(nu_{\max}T^E_{\max}\right)$. □

Numerical Example 2 Consider an 8-job problem, where $LD = 121$ and the maximal (u_j) and minimal (v_j) processing times were generated uniformly in the intervals [25, 50] and [1, 25], respectively.

The generated processing times are u_j = (30, 25, 48, 41, 37, 33, 50, 44) and v_j = (8, 15, 9, 10, 18, 14, 2, 15), implying that $u_{\max} = \max\limits_{j\in\mathcal{J}}\{u_j\} = 50$ and $T^E_{\max} = LD + u_{\max} = 171$.

$\boldsymbol{DP}2$ was executed 50 times, and the optimal solution was achieved when T^E was set to $\left(T^E\right)^* = 125 < 171 = T^E_{\max}$.

The resulting ordered set of early jobs is $\mathcal{J}^E$ = (1, 2, 5, 6), implying that (one) optimal sequence is $\mathcal{J}^*$ = (1, 2, 5, 6, 3, 4, 7, 8).

With regard to set $\mathcal{J}^*$, the actual job processing times are p_j = (30, 25, 37, 33, 9, 10, 2, 15).

The crossover job is $j = 6$, with starting time $S_6 = 92 < 121 = LD$ and completion time.

$$C_6 = \left(T^E\right)^* = 125.$$

We conclude that the optimal makespan is $C^*_{\max} = 161$.

1.4 Minimizing makespan with job-dependent learning-dates allowing idle-times

In this section we introduce a pseudo-polynomial dynamic programming algorithm for problem $\mathcal{Q}3$, i.e., for the setting in which idle times between consecutive jobs are allowed.

The return function $f_{T^E}(j,\ t_1,\ t_2)$ is the minimal makespan for scheduling jobs $j,\ j+1,\ \ldots,\ n$, given that the current completion time of the early jobs is t_1, the current completion time of the late jobs is t_2, and the maximal completion time of the early jobs is T^E.

The recursive function is the following:

Algorithm $\boldsymbol{DP}3$

$$f_{T^E}(j,\ t_1,\ t_2) = \begin{cases} h(j,\ t_1,\ t_2), & if\ t_1 + u_j > T^E\ or\ t_1 > LD_j \\ \min\left\{f_{T^E}\left(j,\ t_1 + u_j,\ t_2\right),\ h(j,\ t_1,\ t_2)\right\}, & \text{otherwise} \end{cases}$$

where:

$$h(j,\ t_1,\ t_2) = \begin{cases} f_{T^E}\left(j+1,\ t_1,\ LD_j + v_j\right) + LD_j + v_j - t_2, & \text{if } LD_j \geq t_2 \\ f_{T^E}\left(j+1,\ t_1,\ t_2 + v_j\right) + v_j, & \text{otherwise} \end{cases}$$

Note that Property 1 (an optimal schedule exists such that all the early jobs are scheduled prior to the late jobs), as well as Properties 2 and 3 (an optimal schedule exists such that all the early jobs and all the late jobs are scheduled according to ELD) remain valid. Hence, the jobs are initially sorted in a non-decreasing order of LD_j and are renumbered accordingly. In each iteration of the proposed DP (denoted $\boldsymbol{DP}3$), as in the previous algorithms, a single job is handled, and it is either scheduled to be early (if possible), or late. In the former case (earliness is feasible), the job will be scheduled either as early as possible (i.e., with no idle time), or the job is delayed and is processed after some idle time to start exactly at its learning date. We use again the definition of $T^E_{max} = \max\limits_{j\in\mathcal{J}}\left\{LD_j + u_j - 1\right\}$, and denote by $T^E \in \left[0,\ T^E_{\max}\right]$, the completion time of all feasible early subsets. Then, we execute $\boldsymbol{DP}3$ for any T^E in this interval, and select the solution with the minimal makespan value. Note that unlike the previous DPs, $\boldsymbol{DP}3$ is a *backward* procedure.

The state variables are:

j-The index of the next job to handle (i.e., the remaining jobs are $\{j,\ j+1,\ \ldots,\ n\}$), $1 \leq j \leq n$.

t_1-The current completion time of the jobs scheduled to be early (clearly, $t_1 \leq \min\left\{\sum_{i=1}^{j-1} u_i,\ T^E\right\}$),

t_2-The current completion time of the jobs scheduled to be late. [$T^E \leq t_2 \leq \max\limits_{1\leq i\leq j-1}\left\{LD_j\right\} + \sum_{i=1}^{j-1} v_i$, because (i) T^E is the completion time of the early jobs, and the late jobs are executed only after the early jobs, and (ii) the completion time of the late jobs cannot be larger than $\max\limits_{1\leq i\leq j-1}\left\{LD_j\right\} + \sum_{i=1}^{j-1} v_i$, because this implies a solution which is not optimal due to unnecessary idle times.]

$h(j,\ t_1,\ t_2)$ reflects the contribution of job j to the makespan value, when it is scheduled to be late: the first line relates to the case with idle time (i.e., job j is delayed to start at time LD_j, and in this case the makespan increases by $LD_j + v_j - t_2$), and the second line relates to the case without idle time (i.e., job j is already late, it starts as early as possible, and the makespan increases by v_j).

The recursive function reflects the following cases: (i) job j cannot be scheduled early, either because it exceeds the upper bound T^E on the completion time of the early jobs, or because its learning date has passed; (ii) job j can be scheduled early or late, and the minimal of the above two options is chosen. Note that if job j is scheduled early, it is better to schedule it without idle time.

The boundary conditions are:

$$f_{T^E}(n,\ t_1,\ t_2) = \begin{cases} h(n,\ t_1,\ t_2), & \text{if } t_1 + u_n > T^E\ or\ t_1 > LD_n \\ 0, & \text{otherwise} \end{cases}$$

$$h(n,\ t_1,\ t_2) = \begin{cases} LD_n + v_n - t_2, & \text{if } LD_n \geq t_2 \\ v_n, & \text{otherwise} \end{cases}$$

The boundary conditions relate to the case where the last job is handled, and then it is better to schedule it to be early if possible, or late otherwise.

The optimal solution for a specific value of T^E is obtained by $f^*_{T^E} = f_{T^E}\left(1,\ 0,\ T^E\right) + T^E$, which is the optimal makespan for scheduling jobs $1,\ \ldots,\ n$ with an initial zero value for the completion time of the early jobs (which is bounded by T^E) and an initial value of T^E for the completion time of the late jobs.

The optimal solution for problem $\mathcal{Q}3$ is therefore: $\min\limits_{T^E\in\left[0,\ T^E_{\max}\right]} f^*_{T^E}$.

Theorem 4 Problem $\mathcal{Q}3$ is solved in $O(n\left(T^E_{max}\right)^2(LD_{max}+\sum v_j))$ time.

Proof j is bounded by n, t_1 is bounded by T^E, and t_2 is bounded by $LD_{\max}+\sum v_j$. In addition, T^E is bounded by $T^E_{\max}$ and $\boldsymbol{DP}3$ is executed $T^E_{\max}$ times. It follows that the total running time is $O\left(n\left(T^E_{\max}\right)^2\left(LD_{\max}+\sum v_j\right)\right)$. □

Numerical Example 3 Consider an 8-job problem, where jobs are sequenced in ELD order. The job-dependent learning-dates are: LD_j = (44, 47, 60, 86, 90, 137, 185, 211), implying that $LD_{\max}=\max\limits_{j\in\mathcal{J}}\left\{LD_j\right\}=211$.

As in Example 1, the maximal (u_j) and minimal (v_j) processing times were generated uniformly in the intervals [25, 50] and [1, 25], respectively.

The resulting processing times are: u_j = (40, 49, 37, 45, 40, 31, 46, 44) and v_j = (34, 32, 24, 22, 22, 21, 34, 20). It follows that $u_{\max}=\max\limits_{j\in\mathcal{J}}\left\{u_j\right\}=49$ and $T^E_{\max}=\max\limits_{j\in\mathcal{J}}\left\{LD_j+u_j-1\right\}=254$.

In the optimal solution achieved by $\boldsymbol{DP}3$, $\left(T^E\right)^*=86<254=T^E_{\max}$.

The ordered set of the early jobs is $\mathcal{J}^E=(1,7)$, implying that (one) optimal sequence is $\mathcal{J}^*=(1,7,2,3,4,5,6,8)$.

The actual job processing times in this optimal sequence are p_j = (40, 46, 32, 24, 22, 22, 21, 20), and the optimal makespan is $C^*_{\max}=231$.

Note that this optimal solution contains an idle time between job 6 (in position 7) and job 8 (in position 8). Job 6 is completed at $C_6=207$, and if job 8 starts at this point, it is an early job and its processing time is$u_8=44$. It is clearly better to create idle time and delay the starting time of job 8 to its learning date:$LD_8=211$. In this case its processing time is reduced to $v_8=20$, which reduces the makespan value to $C^*_{\max}=231$.

[For the sake of complete exposition, we provide below the optimal solution achieved by $\boldsymbol{DP}1$ for the same input:

$$\left(T^E\right)^*=77<254=T^E_{max},\ \mathcal{J}^E=(6,7),$$
$$\mathcal{J}^*=(6,7,1,2,3,4,5,8),$$
$$p_j=(31,46,34,32,24,22,22,20).$$

The optimal makespan (when idle times are not allowed) is $C^*_{\max}=231$, i.e., the same makespan value which was achieved by ***DP3***.]

1.5 Numerical study

We performed numerical tests in order to evaluate the running time of algorithms $\boldsymbol{DP}1$, $\boldsymbol{DP}2$ and $\boldsymbol{DP}3$**,** as a function of the input parameters.

For $\boldsymbol{DP}1$, random instances were generated with $n=50$, 75, 100, 125, 150 and 175 jobs. The maximal processing times $\left(u_j\right)$ were generated uniformly in the intervals [10, 20] and [20, 30], and the minimal processing times $\left(v_j\right)$ were generated uniformly in the intervals [1, 10] and [10, 20], respectively. For each instance, the sum of the maximal processing times was computed, i.e., $P=\sum_{j\in\mathcal{J}}u_j$. The job-dependent learning-dates $\left(LD_j\right)$ were generated uniformly in the interval $[0,\ \lfloor\omega P\rfloor]$, where ω is a tightness factor. We considered $\omega=0.05$, 0.15 and 0.25, to produce various ranges of learning dates. For each combination of n, intervals of u_j and v_j, and ω, 20 instances were generated and solved. (Thus, a total of 720 instances were generated and solved by DP1.) [The programs were implemented in Python, and were executed on a Lenovo 2.00 GHz, Intel Core i7 with 16 GB RAM Memory].

The average and worst case running times of $\boldsymbol{DP}1$ are reported in Table 1 (for $u_j\in[10,20]$ and $v_j\in[1,10]$), and Table 2 (for $u_j\in[20,30]$ and $v_j\in[10,20]$). As expected, the running times increase as the actual processing times increase. Also, the running times increase as a function of the ω values, since larger tightness factors lead to a larger proportion of early jobs. The results validate that the proposed DP performs well in solving medium and even large instances. We note that the worst case running time of $\boldsymbol{DP}1$ with $n=175$, $u_j\in[20,30]$, $v_j\in[10,20]$ and $\omega=0.25$ did not exceed 200 s (see Table 2).

The same input parameters were used for the evaluation of $\boldsymbol{DP}2$. Again, 720 instances were generated and solved (as 20 instances were generated for each combination of n, the intervals of u_j and v_j, and ω). The average and worst case running times obtained by $\boldsymbol{DP}2$ are reported in Table 3 (when $u_j\in[10,20]$ and $v_j\in[1,10]$) and Table 4 (when $u_j\in[20,30]$ and $v_j\in[20,10]$). The running times are much smaller for this special case of a common learning date. The worst case running time of $\boldsymbol{DP}2$ with $n=175$ did not exceed 11 s (see Table 4).

The complexity of $\boldsymbol{DP}3$ is significantly larger than those of $\boldsymbol{DP}1$ and $\boldsymbol{DP}2$. Therefore, much smaller instances were solved in our numerical tests. The numbers of jobs considered were: $n=10,20,\ldots,70$. As above, the maximal processing times $\left(u_j\right)$ were generated uniformly in the interval [10, 20] and [20, 30], and the minimal processing times $\left(v_j\right)$ were generated uniformly in the interval [1, 10] and [10, 20], respectively, and the tightness factors were: $\omega=0.05$ and 0.15. As above, for each combination of n, intervals of u_j and v_j, and ω, 20 instances were generated and solved. Thus, a total of 560 instances were solved by DP3, and the results are reported in Table 5 (where $u_j\in[10,20]$ and $v_j\in[1,10]$) and Table 6 (when $u_j\in[20,30]$ and $v_j\in[10,20]$). The tables reflect the very high complexity of $\boldsymbol{DP}3$ (see Theorem 4): the running times increase dramatically as a function

Table 1 Results of $\boldsymbol{DP}1$ for $u_j \in [10, 20]$, $v_j \in [1, 10]$ Average and worst case running times (seconds)

n	u_j	v_j	$\omega = 0.05$		$\omega = 0.15$		$\omega = 0.25$	
			Average run time [s]	Worst case run time [s]	Average run time [s]	Worst case run time [s]	Average run time [s]	Worst case run time [s]
50	[10, 20]	[1, 10]	0.1	0.1	0.6	0.7	1.6	1.7
75			0.3	0.3	1.9	2.2	5.3	5.9
100			0.7	0.7	4.8	5.5	12.5	13.5
125			1.2	1.3	9.2	10.0	25.3	27.3
150			2.0	2.2	15.7	17.0	42.8	45.2
175			3.2	3.5	25.3	27.1	68.2	72.6

Table 2 Results of $\boldsymbol{DP}1$ for $u_j \in [20, 30]$, $v_j \in [10, 20]$. Average and worst case running times (seconds)

n	u_j	v_j	$\omega = 0.05$		$\omega = 0.15$		$\omega = 0.25$	
			Average run time [s]	Worst case run time [s]	Average run time [s]	Worst case run time [s]	Average run time [s]	Worst case run time [s]
50	[20, 30]	[10, 20]	0.3	0.3	1.7	1.8	4.5	5.1
75			0.8	0.9	5.7	6.0	15.4	18.4
100			1.8	2.0	13.4	14.4	35.8	37.1
125			3.4	3.5	25.9	27.6	71.0	74.9
150			5.7	6.0	44.9	46.5	122.1	127.0
175			8.9	9.4	71.6	74.2	191.6	198.3

Table 3 Results of $\boldsymbol{DP}2$ for $u_j \in [10, 20]$, $v_j \in [1, 10]$. Average and worst case running times (seconds)

n	u_j	v_j	$\omega = 0.05$		$\omega = 0.15$		$\omega = 0.25$	
			Average run time [s]	Worst case run time [s]	Average run time [s]	Worst case run time [s]	Average run time [s]	Worst case run time [s]
50	[10, 20]	[1, 10]	0.0	0.1	0.1	0.2	0.2	0.4
75			0.1	0.2	0.3	0.5	0.3	0.7
100			0.1	0.2	0.3	0.7	0.8	1.4
125			0.2	0.5	0.8	1.3	0.9	2.0
150			0.3	0.6	0.9	1.9	1.4	3.1
175			0.5	0.8	1.2	2.3	2.0	4.4

Table 4 Results of $\boldsymbol{DP}2$ for $u_j \in [20, 30]$, $v_j \in [10, 20]$. Average and worst case running times (seconds)

n	u_j	v_j	$\omega = 0.05$		$\omega = 0.15$		$\omega = 0.25$	
			Average run time [s]	Worst case run time [s]	Average run time [s]	Worst case run time [s]	Average run time [s]	Worst case run time [s]
50	[20, 30]	[10, 20]	0.1	0.2	0.3	0.5	0.4	0.8
75			0.2	0.3	0.6	1.1	1.1	2.1
100			0.3	0.6	1.2	2.0	1.9	3.3
125			0.5	1.0	1.3	3.2	3.5	6.4
150			0.8	1.6	3.0	4.8	4.8	8.3
175			1.1	2.2	4.6	6.5	6.8	11.0

Table 5 Results of $\boldsymbol{DP}3$ for $u_j \in [10, 20]$, $v_j \in [1, 10]$. Average and worst case running times (seconds)

n	u_j	v_j	$\omega = 0.05$		$\omega = 0.15$	
			Average run time [s]	Worst case run time [s]	Average run time [s]	Worst case run time [s]
10	[10, 20]	[1, 10]	0.1	0.1	0.2	0.4
20			0.7	1.0	3.0	4.0
30			2.6	3.5	12.5	16.3
40			7.0	8.2	38.7	48.8
50			15.7	18.6	95.2	117.0
60			29.0	32.6	194.6	237.3
70			48.2	56.8	347.6	404.0

Table 6 Results of $\boldsymbol{DP}3$ for $u_j \in [20, 30]$, $v_j \in [10, 20]$. Average and worst case running times (seconds)

n	u_j	v_j	$\omega = 0.05$		$\omega = 0.15$	
			Average run time [s]	Worst case run time [s]	Average run time [s]	Worst case run time [s]
10	[20, 30]	[10, 20]	0.7	0.9	1.9	2.6
20			5.2	5.8	22.1	26.5
30			18.8	22.2	103.2	121.2
40			50.7	57.5	309.3	364.8
50			120.5	133.4	772.1	856.1
60			235.4	267.4	1534.3	1745.5
70			399.3	435.5	2819.5	3264.4

of the intervals from which the processing times were generated, and as a function of the tightness factor. Note that the average running time required for solving a 70-job problem with $u_j \in [20, 30]$, $v_j \in [10, 20]$ and $\omega = 0.15$ exceeds 2800 s.

2 Conclusions

This study focused on step-learning, i.e., the very realistic phenomenon that the processing times of the jobs starting after their (job-dependent) learning-dates, are reduced. We concentrated first on minimizing makespan on a single machine for the setting that idle times between consecutive jobs are not allowed, proved that the problem is NP-hard, and, subsequently, proposed a pseudo-polynomial time dynamic programming algorithm. The special case of a common learning-date for all the jobs was also studied, and an appropriate (more efficient) DP was introduced. Then, we introduced a more complicated pseudo-polynomial dynamic programming for the case that idle times between consecutive jobs are allowed. Our extensive numerical tests on all algorithms validated that the first two proposed DPs are efficient for solving real-life instances comprised of hundreds of jobs, whereas the third DP is limited to much smaller instances.

A challenging question for future research—is there a more efficient exact solution algorithm for problem $\mathcal{Q}3$ (allowing idle times between jobs)? Other interesting and challenging topics might be dedicated to the extensions: either to multi-step-learning, or to multi-machine settings.

Acknowledgements The second author was supported by the Israel Science Foundation (Grant No. 884/22). The third author was supported by the Israel Science Foundation (Grant No. 2505/19) and by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation – Project Number 452470135).

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

Azzouz, A., Ennigrou, M., & Ben Said, L. (2018). Scheduling problems under learning effects: Classification and cartography. *International Journal of Production Research, 56*(4), 1642–1661.

Azzouz, A., Pan, P. A., Hsu, P. H., Lin, W. C., Liu, S., Said, L. B., & Wu, C. C. (2020). A two-stage three-machine assembly scheduling problem with a truncation position-based learning effect. *Soft Computing, 24*, 10515–10533.

Cheng, M., Xiao, S., Luo, R., & Lian, Z. (2018). Single-machine scheduling problems with a batch-dependent aging effect and variable maintenance activities. *International Journal of Production Research, 56*(23), 7051–7063.

Ding, J., Shen, L., Lü, Z., & Peng, B. (2019). Parallel machine scheduling with completion-time-based criteria and sequence-dependent deterioration. *Computers & Operations Research, 103*, 35–45.

Fu, Y., Zhou, M., Guo, X., & Qi, L. (2019). Artificial-molecule-based chemical reaction optimization for flow shop scheduling problem with deteriorating and learning effects. *IEEE Access, 7*, 53429–53440.

Gao, F., Liu, M., Wang, J. J., & Lu, Y. Y. (2018). No-wait two-machine permutation flow shop scheduling problem with learning effect, common due date and controllable job processing times. *International Journal of Production Research, 56*(6), 2361–2369.

Gawiejnowicz, S. (2008). Time-dependent scheduling. Springer Science & Business Media.

Gawiejnowicz, S. (2020a). *Models and algorithms of time-dependent scheduling* (p. 538). Springer.

Gawiejnowicz, S. (2020b). A review of four decades of time-dependent scheduling: Main results, new topics, and open problems. *Journal of Scheduling, 23*(1), 3–47.

Gawiejnowicz, S., & Kurc, W. (2020). New results for an open time-dependent scheduling problem. *Journal of Scheduling, 23*(6), 733–744.

Geng, X. N., Wang, J. B., & Bai, D. (2019). Common due date assignment scheduling for a no-wait flowshop with convex resource allocation and learning effect. *Engineering Optimization, 51*(8), 1301–1323.

Graham, R. L., Lawler, E. L., & Lenstra, J. K. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics, 5*, 287–326.

Liu, W., Yao, Y., & Jiang, C. (2019). Single-machine resource allocation scheduling with due-date assignment, deterioration effect and position-dependent weights. *Engineering Optimization, 52*(4), 701–714.

Miao, C., & Zhang, Y. (2019). Scheduling with step-deteriorating jobs to minimize the makespan. *Journal of Industrial & Management Optimization, 15*(4), 1955–1964.

Mor, B., Mosheiov, G., & Shapira, D. (2020). Flowshop scheduling with learning effect and job rejection. *Journal of Scheduling, 23*(6), 631–641.

Mosheiov, G. (1995). Scheduling jobs with step-deterioration; minimizing makespan on a single-and multi-machine. *Computers & Industrial Engineering, 28*(4), 869–879.

Mousavi, S. M., Mahdavi, I., & Rezaeian, J. (2018). An efficient bi-objective algorithm to solve re-entrant hybrid flow shop scheduling with learning effect and setup times. *Operational Research International Journal, 18*, 123–158.

Pei, J., Wang, X., Fan, W., Pardalos, P. M., & Liu, X. (2019). Scheduling step-deteriorating jobs on bounded parallel-batching machines to maximise the total net revenue. *Journal of the Operational Research Society, 70*(10), 1830–1847.

Renna, P. (2019). Flexible job-shop scheduling with learning and forgetting effect by multi-agent system. *International Journal of Industrial Engineering Computations, 10*(4), 521–534.

Rostami, M., Nikravesh, S., & Shahin, M. (2018). Minimizing total weighted completion and batch delivery times with machine deterioration and learning effect: A case study from wax production. *Operational Research International Journal*. https://doi.org/10.1007/s12351-018-0373-6

Soper, A. J., & Strusevich, V. A. (2020). Refined conditions for V-shaped optimal sequencing on a single machine to minimize total completion time under combined effects. *Journal of Scheduling, 23*(6), 665–680.

Strusevich, V. A., & Rustogi, K. (2017). *Scheduling with time-changing effects and rate-modifying activities*. Springer International Publishing.

Sun, X., & Geng, X. N. (2019). Single-machine scheduling with deteriorating effects and machine maintenance. *International Journal of Production Research, 57*(10), 3186–3199.

Sun, X., Geng, X. N., Wang, J. B., & Liu, F. (2019). Convex resource allocation scheduling in the no-wait flowshop with common flow allowance and learning effect. *International Journal of Production Research, 57*(6), 1873–1891.

Wang, H., Huang, M., & Wang, J. (2019a). An effective metaheuristic algorithm for flowshop scheduling with deteriorating jobs. *Journal of Intelligent Manufacturing, 30*(7), 2733–2742.

Wang, J. B., Liu, F., & Wang, J. J. (2019b). Research on m-machine flow shop scheduling with truncated learning effects. *International Transactions in Operational Research, 26*(3), 1135–1151.

Woo, Y. B., & Kim, B. S. (2018). Matheuristic approaches for parallel machine scheduling problem with time-dependent deterioration and multiple rate-modifying activities. *Computers & Operations Research, 95*, 97–112.

Wu, C. C., Azzouz, A., Chung, I. H., Lin, W. C., & Ben Said, L. (2019). A two-stage three-machine assembly scheduling problem with deterioration effect. *International Journal of Production Research, 57*(21), 6634–6647.

Wu, C. C., Wang, D. J., Cheng, S. R., Chung, I. H., & Lin, W. C. (2018). A two-stage three-machine assembly scheduling problem with a position-based learning effect. *International Journal of Production Research, 56*(9), 3064–3079.

Wu, C. C., Zhang, X., Azzouz, A., Shen, W. L., Cheng, S. R., Hsu, P. H., & Lin, W. C. (2020). Metaheuristics for two-stage flow-shop assembly problem with a truncation learning function. *Engineering Optimization*. https://doi.org/10.1080/0305215X.2020.1757089

Yan, P., Wang, J. B., & Zhao, L. Q. (2019). Single-machine bi-criterion scheduling with release times and exponentially time-dependent learning effects. *Journal of Industrial & Management Optimization, 15*(3), 1117–1131.