Journal of Combinatorial Optimization (2022) 44:207–222
https://doi.org/10.1007/s10878-021-00821-2

Parallel-machine scheduling of jobs with mixed job-, machine- and position-dependent processing times

Bartłomiej Przybylski[1]

Accepted: 28 September 2021 / Published online: 16 October 2021

Abstract
We consider a number of parallel-machine scheduling problems in which jobs have variable processing times. The actual processing time of each job is described by an arbitrary positive function of the position it holds on a machine. However, the function itself may additionally depend on the job or a machine this job was assigned to. Our aim is to find a schedule that minimizes the objectives of maximum completion time or the total completion time. We present a full set of polynomial solutions for the cases of jobs with no precedence constraints. We also show that the case of single-chained jobs may be not easier in general, but some polynomial results can be obtained, too.

Keywords Parallel-machine scheduling · Variable processing times · Position-dependent processing times · Makespan · Total completion time · Polynomial algorithms

1 Introduction

In many real-world scheduling problems one cannot assume that job processing times are fixed. Wright (1936) noticed that the time required to perform an aircraft production task decreases in conjunction with the worker's experience. Today, it is widely observed that the processing time of a job may vary in reaction to different environmental factors, such as the amounts of resources available, the starting time of the job, or the set of jobs performed earlier. In the latter case, it is often assumed that the actual processing time of a job depends either on its position in a schedule, or on the sum of basic processing times of jobs that have been already executed. If all the jobs are assumed to be identical, both these approaches are, in some sense, exchangeable. In this paper, the actual processing time of a job is described by a positive function of at

✉ Bartłomiej Przybylski
bap@amu.edu.pl

[1] Faculty of Mathematics and Computer Science, Adam Mickiewicz University, Uniwersytetu Poznańskiego 4, 61-614 Poznań, Poland

most three arguments: the index of the job, the machine to which the job is assigned, and the position it holds on this machine. We also assume that the jobs are either independent, or they need to be processed in a given order, i.e. the precedence constraints take form of a single chain.

Recently, one can observe a growing number of papers related to scheduling problems with the *learning effect*, where the processing time of a job decreases as the set of already executed jobs grows (see, e.g., Wang et al. 2020). A dual group of problems with the *aging effect* is also widely considered in literature (see, e.g., Lu et al. 2018; Liu et al. 2018). Having this in mind, we analyze a number of general scheduling problems where job processing times may, but do not have to give in to such a monotonic impact. In particular, we show polynomial algorithms for scheduling problems in which job processing times are described by any arbitrary positive and discrete function. The analyzed problems vary in: the set of factors that impact actual processing times of jobs (position, position and machine, position and job, or all the factors together), the precedence constraints among jobs (independent or chained jobs), and the objective function (maximum completion time or the total completion time).

In practice, it is often assumed that job processing times vary among jobs. As the machines may have different capabilities, the processing time may also depend on the machine chosen (scheduling on unrelated machines). However, the assumption that the actual processing time of a job freely depends on the position it holds on a machine, may seem artificial and impractical. As indicated above and further discussed in Sect. 2, position-dependent models are usually monotonic with respect to the job position. By assuming that job processing time may freely depend on its position, we obtain general results that can be successfully applied both for learning and aging models.

The contribution of the paper is two-fold. First, we analyze various parallel-machine scheduling problems where the actual processing times of independent jobs may freely depend on the three factors: machine, job, and its position on a machine. We present the full classification of computational complexity for these problems and their subproblems. It can be shown that most of such problems can be solved in polynomial time. Second, we show that although the case of single-chained jobs is not easier in general, some of the corresponding problems in this group can also be solved in polynomial time.

This paper is organized as follows. In Sect. 2, we briefly review main position-dependent learning and aging models. We also present main results in the area of job- and machine-dependent processing times. In Sect. 3, we formalize the problem considered in this paper. In Sect. 4, we present the full classification of the complexity of problems with independent jobs. Then, in Sect. 5, we show how the assumption that the jobs are chained influences the analyzed problems. We complete the paper by Sect. 6 which includes conclusions and remarks on future research.

2 Literature review

To the best of our knowledge, the first position-dependent model, where the actual job processing time depends on the number of jobs executed earlier, was proposed by Gawiejnowicz (1996). In this model, the actual processing time of the i-th job scheduled on the r-th position, $p_{i,r}$, depends on its basic processing time, $\bar{p}_i$, and the value of $\nu(r) \in [0, 1]$. In particular, Gawiejnowicz focused on makespan minimization on a single machine within the general $p_{i,r} = \bar{p}_i/\nu(r)$ model. Similar results—limited to the learning effect—were presented by Biskup (1999), who introduced the $p_{i,r} = \bar{p}_i r^a$ model, where $a < 0$ is a constant *learning index*. He considered single-machine scheduling problems to minimize the total flow time and the maximum deviation from the common due date. A few years later, Mosheiov and Sidney (2003) analyzed a more general case where the learning indices are job-dependent, that is $p_{i,r} = \bar{p}_i r^{a_i}$. All these results constituted a base for rich research in the area of position-dependent scheduling.

Another popular approach to position-dependent scheduling is based on the paper by DeJong (1957). Okołowski and Gawiejnowicz (2010) considered a group of parallel-machine scheduling problems within the model of generalized DeJong's learning effect. In this model, it is assumed that $p_{i,r} = \bar{p}_i[M + (1 - M)r^a]$, where $0 \leq M < 1$ is called the *incompressibility factor*. The same model was analyzed later by Ji et al. (2015).

The list of models, where job processing time depends on the number of jobs executed earlier, is still expanding. There are a few models that take into account that job processing times may be affected simultaneously by factors of different kind. For example, a number of models in which time-dependent job deterioration and DeJong's learning effect are mixed were analyzed by Zhang et al. (2018) and Sun et al. (2020). We refer the reader to Agnetis et al. (2014), Strusevich and Rustogi (2017), Azzouz et al. (2017), and Gawiejnowicz (2020) for more details on different scheduling models of the kind discussed in this section.

There are two main limitations of the models of scheduling with variable job processing times, especially those that are position-dependent. First, the majority of results are related to single-machine scheduling (see, e.g., Wang 2010; Wang and Wang 2013; Debczynski and Gawiejnowicz 2013 or Huang and Wang 2014). At the same time, one can find only limited number of polynomial results on parallel-machine scheduling of position-dependent jobs (see, e.g., Mosheiov and Sidney 2003; Mosheiov 2008 or Przybylski 2017). Second, most papers related to position-dependent scheduling focus on particular learning or aging models. In fact, there are only individual results independent of the form of the function describing the actual processing time of a job (e.g. Mosheiov 2008). Inspired by the latter paper, we assume that the actual processing time of a job may freely depend on the position it holds on a machine. In particular, we do not make any assumption on the form of such a relation, including its monotonicity. This way, our results can be successfully applied to almost all position-dependent models discussed in literature.

For many practical applications it is also justified to assume that the actual processing time of a job freely depends on a machine it is assigned to. The general problems of scheduling fixed-time jobs on unrelated machines to minimize the makespan or the total weighted completion time are known to be $\mathcal{NP}$-Hard. For this reason, approximation algorithms are proposed (see, e.g., Lenstra et al. 1990; Schulz and Skutella 2002 or Bansal et al. 2019). On the other hand, Bruno et al. (1974) showed that there exists a polynomial algorithm for the case of total completion time as the objective.

Having this in mind, we also assume that the actual processing time of a job may depend on a machine it is being assigned to. In the next section, we present the formal definition of this dependency.

3 Problem formulation

In this section, we formulate the set of parallel-machine scheduling problems with job-, machine-, and position-dependent processing times which are going to be discussed henceforth. Let us be given m parallel machines, $P_1, P_2, \ldots, P_m$, and n non-preemptable jobs, $J_1, \ldots, J_n$. The processing time of job J_i assigned to a given machine P_j is described by the $\varphi_i^j : \mathbb{N}_+ \rightarrow \mathbb{Q}_+$ function. The value of the function freely depends on the position r the job holds on the j-th machine. In other words, the processing time of the i-th job holding the r-th position on the j-th machine is equal to $p_{i,j,r} = \varphi_i^j(r)$. If the values of the functions are job-independent, they will be denoted by φ^j, and then it holds that $p_{j,r} = \varphi^j(r)$ for each job J_i. Similarly, if the values of the functions are machine-independent, they will be denoted by φ_i, and then it holds that $p_{i,r} = \varphi_i(r)$ for each feasible i. If the values of the functions are job- and machine-independent, they will be denoted by φ, and then it holds that $p_r = \varphi(r)$. The jobs may be either independent (i.e. with no precedence constraints), or chained. Our aim is to find an optimal schedule in the sense of at least one of the following objective functions: maximum completion time or total completion time. Using the traditional three-field notation (see Gawiejnowicz 2020 for a concise description), we can denote these problems by $\mathrm{P}m|p_{i,j,r} = \varphi_i^j(r)|C_{\max}$, $\mathrm{P}m|p_{i,j,r} = \varphi_i^j(r)|\sum C_i$, $\mathrm{P}m|\text{chain}, p_{i,j,r} = \varphi_i^j(r)|C_{\max}$, and $\mathrm{P}m|\text{chain}, p_{i,j,r} = \varphi_i^j(r)|\sum C_i$.

Throughout the paper we will assume that the set of φ_i^j functions is a part of the problem description, not the input, and that the values of such φ_i^j functions are precomputed and thus obtainable in constant time. This assumption helps us focus on the essence of the presented results. However, if the values of the φ_i^j functions were computed on the fly and could be obtained in polynomial time, then all the polynomial algorithms presented henceforth would remain polynomial.

4 Independent jobs

In this section, we consider problems in which position-dependent actual processing times are additionally job-dependent, machine-dependent, or mixed job- and machine-dependent, i.e. $p_{i,r} = \varphi_i(r)$, $p_{j,r} = \varphi^j(r)$, or $p_{i,j,r} = \varphi_i^j(r)$, respectively. However, we assume that there are no precedence constraints among jobs.

Let us start this section by observing that the $\mathrm{P}m|p_r = \varphi(r)|C_{\max}$ and $\mathrm{P}m|p_r = \varphi(r)|\sum C_i$ problems can be easily solved by assigning at most $\lceil n/m \rceil$ jobs to each of the machines in such a way that the numbers of jobs executed on any pair of machines do not differ by more than one. As it will be shown, in case of machine-dependent processing times, some of the problems can still be solved in polynomial time even if chain precedence constraints are defined. However, if the actual processing times are job-dependent, then they might be—as a special case—constant. As, for $m = 2$, this leads to the P2$||C_{\max}$ problem, the following proposition holds.

Proposition 1 *The $P2|p_{i,r} = \varphi_i(r)|C_{\max}$ problem is $\mathcal{NP}$-Hard.*

It is worth noticing that although the $\mathrm{P2}|p_{i,r} = \varphi_i(r)|C_{\max}$ problem is $\mathcal{NP}$-Hard, the $1|p_{i,r} = \varphi_i(r)|C_{\max}$ problem can be rewritten as the following balanced Assignment problem.

$$\min \quad \sum_{i=1}^{n}\sum_{r=1}^{n} \varphi_i(r) \cdot x_{i,r} \tag{1}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} x_{i,r} = 1 \qquad \text{for } r = 1, 2, \ldots, n \tag{2}$$

$$\sum_{r=1}^{n} x_{i,r} = 1 \qquad \text{for } i = 1, 2, \ldots, n$$

$$x_{i,r} \in \{0, 1\}. \tag{3}$$

Here, $x_{i,r}$ is an indicator variable that is equal to 1 if and only if job J_i holds the r-th position in a schedule. Condition (2) guarantees that only one job will be assigned to each of the positions, from 1 to n. Similarly, condition (3) ensures us that each job will be assigned to exactly one position. Finally, (1) represents the value of the $C_{\max}$ objective function for a given job-to-position assignment. As shown by Fredman and Tarjan (1987), a balanced Assignment problem can be solved in $O(n^3)$. We refer the reader to a book by Burkard et al. (2012) for a wide review on the Assignment problem.

The $1|p_{i,r} = \varphi_i(r)|\sum C_i$ problem can also be rewritten as a variant of the Assignment problem. Assume that n jobs are executed on a single machine, one after another, with their processing times equal to $p_1, p_2, \ldots, p_n$, respectively. Then,

$$\begin{aligned}\sum C_i &= p_1 + (p_1 + p_2) + \cdots + (p_1 + p_2 + \cdots + p_n)\\ &= n \cdot p_1 + (n-1) \cdot p_2 + \cdots + 2 \cdot p_{n-1} + p_n.\end{aligned}$$

For this reason, the objective takes form of

$$\min \quad \sum_{i=1}^{n}\sum_{r=1}^{n} (n - r + 1) \cdot \varphi_i(r) \cdot x_{i,r}. \tag{4}$$

As n is a constant, one can introduce a new function, $\bar{\varphi}_i(r) = (n - r + 1) \cdot \varphi_i(r)$, and rewrite the objective as

$$\min \quad \sum_{i=1}^{n}\sum_{r=1}^{n} \bar{\varphi}_i(r) \cdot x_{i,r}.$$

which is equivalent to (1). Thus, the $1|p_{i,r} = \varphi_i(r)|\sum C_i$ problem can be solved in polynomial time, too. What is more interesting, the latter result can be generalized to a case of parallel machines, mixed position-, job- and machine-dependent processing times, and $\sum C_i$ objective function as follows.

Let us start with a two-machine case in which $p_{i,j,r} = \varphi_i^j(r)$. Consider an instance of the ASSIGNMENT problem, denoted by I_c. In this instance, we are given n jobs which should be assigned to both a position and a machine. However, we assume *a priori* that exactly c jobs will be assigned to the first machine. This means that the remaining $n-c$ jobs will be assigned to the second machine. Given an instance I_c and any corresponding feasible schedule T_c with no idle times between jobs, we have

$$\begin{aligned}\sum C_i(T_c) &= \sum_{r=1}^{c}\sum_{q=1}^{r}\varphi^1_{[q]}(q) + \sum_{r=1}^{n-c}\sum_{q=1}^{r}\varphi^2_{[q]}(q)\\ &= \sum_{r=1}^{c}(c-r+1)\cdot\varphi^1_{[r]}(r) + \sum_{r=1}^{n-c}(n-c-r+1)\cdot\varphi^2_{[r]}(r) \qquad (5)\end{aligned}$$

where $\varphi^j_{[r]}$ is a function associated with the job holding the r-th position on the j-th machine. Figure 1 shows an example of a corresponding bipartite graph for $n = 5$ and $c = 2$. The weights of the edges correspond to the impact of a job-to-machine-to-position assignment on the final value of the $\sum C_i$ objective function, based on (5). With the reasoning similar to the one presented for (4), instance I_c can be formulated as the following ILP program for the ASSIGNMENT problem.

$$\min \quad \sum_{i=1}^{n}\left[\sum_{r=1}^{c}(c-r+1)\cdot\varphi_i^1(r)\cdot x_{i,1,r} + \sum_{r=1}^{n-c}(n-c-r+1)\cdot\varphi_i^2(r)\cdot x_{i,2,r}\right] \qquad (6)$$

$$\text{s.t.} \quad \sum_{i=1}^{n} x_{i,1,r} = 1 \qquad \text{for } r = 1,2,\ldots,c \qquad (7)$$

$$\sum_{i=1}^{n} x_{i,2,r} = 1 \qquad \text{for } r = 1,2,\ldots,n-c \qquad (8)$$

$$\sum_{r=1}^{c} x_{i,1,r} + \sum_{r=1}^{n-c} x_{i,2,r} = 1 \qquad \text{for } i = 1,2,\ldots,n \qquad (9)$$

$$x_{i,j,r} \in \{0,1\}. \qquad (10)$$

Here, conditions (7–8) guarantee that exactly one job will be assigned to each of the c positions on the first machine, and each of the $n-c$ positions on the second machine. Similarly, condition (9) ensures us that each job will be assigned to exactly one spot.

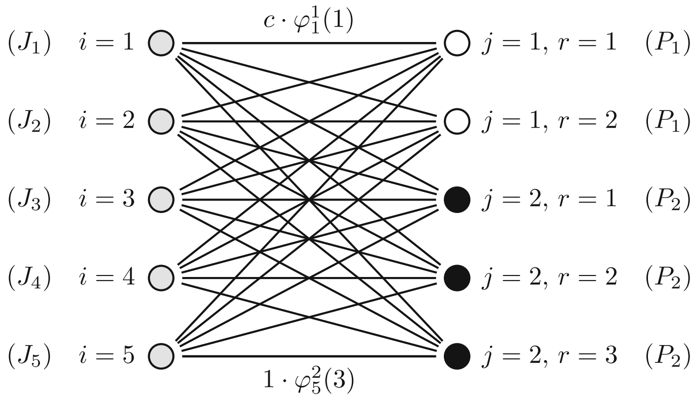

Fig. 1 The I_2 instance with 5 jobs

As we can now solve the case with fixed number of jobs assigned to the first machine, we may construct an enumerating algorithm for the general problem, presented as Algorithm 1.

Algorithm 1

Input: The set of $\varphi_i^j(r)$ values for $i, r \in \{1, 2, \dots, n\}$ and $j \in \{1, 2\}$

Output: An optimal schedule for the $P2|p_{i,j,r} = \varphi_i^j(r)|\sum C_i$ problem

```
1: $T \leftarrow$ An empty schedule
2: $C \leftarrow \infty$
3: for $c \leftarrow 0, 1, \dots, n$ do
4:     CREATE the $I_c$ instance of the ASSIGNMENT problem.
5:     FIND an optimal schedule $T_c$ for $I_c$ based on the (6–10) program.
6:     if $\sum C_i(T_c) < C$ then
7:         $T \leftarrow T_c$
8:         $C \leftarrow \sum C_i(T_c)$
9: return $T$
```

Theorem 1 *Algorithm 1 solves the $P2|p_{i,j,r} = \varphi_i^j(r)|\sum C_i$ problem in $O(n^4)$ time.*

Proof In order to find an optimal schedule for the $P2|p_{i,j,r} = \varphi_i^j(r)|\sum C_i$ problem, we consider all the possible numbers of jobs c assigned to the first machine. Then, we select the c' value that guarantees the lowest value of the objective function. Thus, the algorithm must lead to an optimal schedule. Notice that the optimal c' may be any value between 0 to n, as the actual processing times of jobs are also machine-dependent. For example, if $\varphi_i^1(r) < \frac{1}{n}$ and $\varphi_i^2(r) \geq 1$ for each i and r, then $c' = n$ (in the optimal schedule all n jobs are assigned to the first machine). Similarly, if $\varphi_i^1(r) \geq 1$ and $\varphi_i^2(r) < \frac{1}{n}$ for each i and r, then $c' = 0$ (in the optimal schedule all n jobs are assigned to the second machine). Notice that if the processing times were machine-independent, then it would be enough to consider only the cases in which

$0 \leq c \leq \lfloor n/2 \rfloor$. Indeed, given the optimal schedule T_c inside the **for** loop (line 3), one could immediately obtain the optimal schedule T_{n-c} with the same value of the objective function, simply by swapping the machines.

As the **for** loop (line 3) is executed $n+1$ times, and each time an optimal assignment for I_c is found in $O(n^3)$, the total required number of operations is $O(n^4)$. □

Now, let us consider the general P$m|p_{i,j,r} = \varphi_i^j(r)|\sum C_i$ problem. One can notice that the same approach can be applied here. Indeed, let $AP(m,n)$ be a number of Assignment problems that need to be solved in order to find an optimal solution for m machines and n jobs. It holds that

$$AP(m,n) = \left\{ \begin{array}{ll} 1, & \text{for } m = 1 \\ 1 + \sum_{c=0}^{n-1} AP(m-1, n-c), & \text{for } m > 1 \end{array} \right\} = O(n^{m-1}).$$

This means that the P$m|p_{i,j,r} = \varphi_i^j(r)|\sum C_i$ problem can be solved in the time of $O(n^{m-1} \cdot n^3) = O(n^{m+2})$. The same asymptotic result can be justified by the following reasoning. Given a fixed sequence of non-negative numbers of jobs $(n_1, n_2, \ldots, n_m)$ that have to be assigned to each of the machines, one can find an optimal solution by solving the Assignment problem. The number of $(n_1, n_2, \ldots, n_m)$ sequences such that $\sum_{j=1}^m n_j = n$ is defined by $\binom{n+m-1}{n} = O(n^{m-1}/(m-1)!)$, for a fixed m. Thus, the P$m|p_{i,j,r} = \varphi_i^j(r)|\sum C_i$ problem can be solved in $O(n^{m+2})$.

We will finish this section by considering the P2$|p_{j,r} = \varphi^j(r)|C_{\max}$ and the P2$|p_{j,r} = \varphi^j(r)|\sum C_i$ problems. Let us observe that for any schedule T the values of $C_{\max}(T)$ and $\sum C_i(T)$ depend on the number of jobs executed on each of the machines only. For example, if in the T_c schedule there are c jobs assigned to machine P_1 and $n-c$ jobs assigned to machine P_2, then

$$C_{\max}(T_c) = \max \left\{ \sum_{r=1}^{c} \varphi^1(r), \sum_{r=1}^{n-c} \varphi^2(r) \right\}$$

and

$$\sum C_i(T_c) = \sum_{r=1}^{c} (c - r + 1) \cdot \varphi^1(r) + \sum_{r=1}^{n-c} (n - c - r + 1) \cdot \varphi^2(r).$$

For this reason, the optimal schedule can be obtained enumeratively by finding, in $O(n)$ time, the values of

$$c' = \underset{0 \leq c \leq n}{\arg\min} \max \left\{ \sum_{r=1}^{c} \varphi^1(r), \sum_{r=1}^{n-c} \varphi^2(r) \right\}$$

and

$$c'' = \underset{0 \le c \le n}{\arg\min} \left\{ \sum_{r=1}^{c} (c - r + 1) \cdot \varphi^1(r) + \sum_{r=1}^{n-c} (n - c - r + 1) \cdot \varphi^2(r) \right\},$$

respectively, and then assigning any c' jobs to machine P_1 and the remaining $n - c'$ jobs to machine P_2 in the case of the $C_{\max}$ objective function, and any c'' jobs to machine P_1 and the remaining $n - c''$ jobs to machine P_2 in the case of the $\sum C_i$ objective function.

In general, the $\mathrm{P}m|p_{j,r} = \varphi^j(r)|C_{\max}$ and $\mathrm{P}m|p_{j,r} = \varphi^j(r)|\sum C_i$ problems can be solved by greedily assigning each of n jobs to a machine that guarantees the lowest $C_{\max}$ at the moment of assignment, as shown in Algorithm 2.

Algorithm 2

Input: The set of $\varphi^j(r)$ values for $r \in \{1, 2, \dots, n\}$ and $j \in \{1, 2, \dots, m\}$
Output: An optimal schedule for the $\mathrm{P}m|p_{j,r} = \varphi^j(r)|C_{\max}$ and $\mathrm{P}m|p_{j,r} = \varphi^j(r)|\sum C_i$ problems

```
1: $T \leftarrow$ An empty schedule for $m$ identical parallel machines
2: $C \leftarrow [0, 0, \dots, 0]$          ▷ A $1 \times m$ vector (number of jobs executed on each machine)
3: $P \leftarrow [\varphi^1(1), \varphi^2(1), \dots, \varphi^m(1)]$          ▷ Potential $C_{\max}$ values for each of the machines
4: for $i \leftarrow 1, 2, \dots, n$ do
5:     FIND any index $p \in \{1, 2, \dots, n\}$ such that $P[p] = \min(P)$.
6:     In schedule $T$, ASSIGN job $J_i$ to machine $P_p$ so it is executed as soon as possible.
7:          ▷ The job is executed in the $\big[P[p] - \varphi^p(C[p] + 1), P[p]\big)$ interval.
8:     $C[p] \leftarrow C[p] + 1$
9:     $P[p] \leftarrow P[p] + \varphi^p(C[p] + 1)$
10: return $T$
```

Theorem 2 *Algorithm 2 solves the $\mathrm{P}m|p_{j,r} = \varphi^j(r)|C_{\max}$ and $\mathrm{P}m|p_{j,r} = \varphi^j(r)|\sum C_i$ problems in $O(m + n \log m)$ time.*

Proof One can notice that for a single job it is optimal to select a machine that guarantees the lowest completion time of this job. Assume that $n > 1$ is the lowest number of jobs for which Algorithm 2 leads to a suboptimal schedule.

Let T_n be a schedule generated by Algorithm 2 for n jobs and let T'_n be an optimal schedule for the same instance. It must hold that $C_{\max}(T'_n) < C_{\max}(T_n)$, because T_n is suboptimal. Let us remove from schedule T_n any job for which the completion time is equal to $C_{\max}(T_n)$. Call the resulting schedule T_{n-1}.

By the construction of Algorithm 2, T_{n-1} is an optimal schedule of $n - 1$ jobs. For that reason it must hold that $C_{\max}(T_{n-1}) \le C_{\max}(T'_n) < C_{\max}(T_n)$. At the same time, there exists at least one machine P_j such that the number of jobs assigned to it in the T_{n-1} schedule is lower than in the T'_n schedule. After assigning a new job to that machine we obtain a new schedule, S_n. Again, by the construction of the algorithm, it must hold that $C_{\max}(T_n) \le C_{\max}(S_n)$. Consequently, $C_{\max}(T_n) \le C_{\max}(S_n) \le C_{\max}(T'_n) < C_{\max}(T_n)$ which is a contradiction. A similar reasoning can be provided for the $\sum C_i$ objective function.

Table 1 Complexity of considered problems with independent jobs

Proc. times ↓	$C_{\max}$			$\sum C_i$		
No. of m. →	1	2	m	1	2	m
$p_r = \varphi(r)$	$O(n)$	$O(n)$	$\boldsymbol{O(n)}$	$O(n)$	$O(n)$	$\boldsymbol{O(n)}$
$p_{j,r} = \varphi^j(r)$	$O(n)$	$\boldsymbol{O(n)}$	$\boldsymbol{O(m + n \log m)}$	$O(n)$	$\boldsymbol{O(n)}$	$\boldsymbol{O(m + n \log m)}$
$p_{i,r} = \varphi_i(r)$	$\boldsymbol{O(n^3)}$	**NPH**	NPH	$\boldsymbol{O(n^3)}$	$O(n^4)$	$O(n^{m+2})$
$p_{i,j,r} = \varphi_i^j(r)$	$O(n^3)$	NPH	NPH	$O(n^3)$	$\boldsymbol{O(n^4)}$	$\boldsymbol{O(n^{m+2})}$

Before the **for** loop (line 4), one needs to perform $O(m)$ operations. In the **for** loop, the number of operations is bounded by $O(nm)$. However, if one implements P as a priority queue using a strict Fibonacci heap (Brodal et al. 2012), this result can be improved. Building a heap P at the beginning would still require $O(m)$ operations, but finding $\min(P)$ could be done in $O(1)$ time, and increasing a key in the P heap would require $O(\log m)$ time. For that reason the number of operations required by the **for** loop (line 4) can be reduced to $O(n \log m)$, and thus the total number of operations is $O(m + n \log m)$. □

We present a summary of the results discussed up to this moment in Table 1. The results proved in this section are printed in bold. Other results in the table follow either from their generalizations (if they are polynomial) or from their special cases (in case of $\mathcal{NP}$-Hardness).

5 Chained jobs

In the previous section, we analyzed the $Pm|p_{i,j,r} = \varphi_i^j(r)|C_{\max}$ and $Pm|p_{i,j,r} = \varphi_i^j(r)|\sum C_i$ problems of scheduling jobs with no precedence constraints. Now, we will assume that the jobs need to be executed in a given order, i.e. that they are chained.

It might be observed that when one schedules jobs with fixed processing times, the problems in which jobs are single-chained are usually easier than the ones with independent jobs. For example, the $P||C_{\max}$ problem is $\mathcal{NP}$-Hard, while the $P|\text{chain}|C_{\max}$ problem can be solved in linear time. It is also true for more general $R||C_{\max}$ and $R|\text{chain}|C_{\max}$ problems.

However, it seems that the case of chained jobs is generally harder than the case of independent jobs when it comes to mixed job-, machine- and position-dependent processing times. Despite this, in this section we will present polynomial algorithms for the $Pm|\text{chain}, p_{j,r} = \varphi^j(r)|C_{\max}$ and $Pm|\text{chain}, p_{j,r} = \varphi^j(r)|\sum C_i$ problems. We will analyze these problems one after another.

Let us begin with the $P2|\text{chain}, p_{j,r} = \varphi^j(r)|C_{\max}$ problem. Assume that we are given n chained jobs, i.e. $J_1 \to J_2 \to, \cdots \to J_n$. One can observe that in any optimal schedule related to the P2$|\text{chain}, p_{j,r} = \varphi^j(r)|C_{\max}$ problem jobs must be executed one after another, without unnecessary idleness. That means that the value of the $C_{\max}$ objective function is basically the sum of the actual processing times of all jobs. Moreover, the objective value does not depend on the exact job-to-machine assignment, but rather on the number of jobs assigned to the first (and consequently, to the second) machine. If one assigns c jobs to the first machine, then the sum of their actual processing times is equal to $\sum_{r=1}^{c} \varphi^1(r)$. As there are n jobs in total, the remaining $n - c$ jobs must be assigned to the second machine, with sum of their actual processing times equal to $\sum_{r=1}^{n-c} \varphi^2(r)$. This means that for any schedule T_c with exactly c jobs assigned to the first machine and with no idle times between consecutive jobs, we have

$$C_{\max}(T_c) = \sum_{r=1}^{c} \varphi^1(r) + \sum_{r=1}^{n-c} \varphi^2(r).$$

These considerations lead us to the linear algorithm for the P2|chain, $p_{j,r} = \varphi^j(r)|C_{\max}$ problem, presented as Algorithm 3.

Algorithm 3

Input: The set of $\varphi^j(r)$ values for $r \in \{1, 2, \ldots, n\}$ and $j \in \{1, 2\}$
Output: An optimal schedule for the P2|chain, $p_{j,r} = \varphi^j(r)|C_{\max}$ problem

```
Step 1. Find
            $c' = \arg\min_{c} \left\{ \sum_{r=1}^{c} \varphi^1(r) + \sum_{r=1}^{n-c} \varphi^2(r) \right\},$
        where $0 \le c \le n$.
Step 2. Assign $c'$ jobs to the first machine and the remaining $n - c'$ jobs to the second machine.
```

Algorithm 3 is based on the observation, that there are only $n + 1$ ways to write down a positive integer n as a sum of two non-negative integers. In general, for m machines, there are $O(n^{m-1}/(m-1)!)$ ways to assign n identical jobs to m machines (see the discussion on the P$m|p_{i,j,r} = \varphi_i^j(r)|\sum C_i$ problem). So, the $O(n^{m-1}/(m-1)!)$ algorithm, presented as Algorithm 4, can be applied for the Pm|chain, $p_{j,r} = \varphi^j(r)|C_{\max}$ problem.

Algorithm 4

Input: The set of $\varphi^j(r)$ values for $r \in \{1, 2, \ldots, n\}$ and $j \in \{1, 2, \ldots, m\}$
Output: An optimal schedule for the Pm|chain, $p_{j,r} = \varphi^j(r)|C_{\max}$ problem

```
Step 1. Find
            $(n'_1, n'_2, \ldots, n'_m) = \arg\min_{(n_1, n_2, \ldots, n_m)} \left\{ \sum_{r=1}^{n_1} \varphi^1(r) + \sum_{r=1}^{n_2} \varphi^2(r) + \cdots + \sum_{r=1}^{n_m} \varphi^m(r) \right\},$
        where $n_j \ge 0$ and $\sum_{j=1}^{m} n_j = n$.
Step 2. Assign any $n'_j$ jobs to machine $P_j$, for $1 \le j \le m$.
```

Fig. 2 Notation for a set of chains

$$\begin{tikzcd}
J_{1,1} \arrow[r] & J_{1,2} \arrow[r] & J_{1,3} \arrow[r] & \cdots \arrow[r] & J_{1,n_1} & \\
J_{2,1} \arrow[r] & J_{2,2} \arrow[r] & J_{2,3} \arrow[r] & J_{2,3} \arrow[r] & \cdots \arrow[r] & J_{2,n_2} \\
\cdots & & & & & \\
J_{m,1} \arrow[r] & J_{m,2} \arrow[r] & \cdots \arrow[r] & J_{m,n_m} & &
\end{tikzcd}$$

Is is worth noticing, based on the results by Przybylski (2017), that if the φ function is non-increasing, then the Pm|chains, $p_r = \varphi(r)|C_{\max}$ problem with more than one chain of jobs to be scheduled can be solved in $O(n)$.

In case of the P2|chain, $p_{j,r} = \varphi^j(r)|C_{\max}$ problem, one can find an optimal schedule by answering a simple question of *how many jobs should be assigned to each of two machines?* Given an optimal number c' of jobs that should be assigned to machine P_1, one of $\binom{n}{c'}$ optimal schedules can be obtained by choosing any c' jobs from the original chain of n jobs. However, in case of the $\sum C_i$ objective function, there are two questions to be answered. First, *how many jobs should be assigned to each of two machines*, and second, *which jobs should it be*. Thus, the P2|chain, $p_{i,j,r} = \varphi_i^j(r)|\sum C_i$ problem cannot be reformulated as the Assignment problem, even if we assume that the number of jobs executed on each of the machines is constant. This is so, because jobs need to be executed in a given order. As a consequence, the actual processing time of a job J_i in any feasible schedule directly depends on how many jobs from the $\{J_1, J_2, \ldots, J_{i-1}\}$ set were assigned to each of the machines.

It turns out that the Pm|chain, $p_{j,r} = \varphi^j(r)|\sum C_i$ problem can still be solved by a polynomial algorithm that uses, as a subroutine, a known algorithm for the 1|chains|$\sum C_i$ problem. For the latter case, assume that we are given a set of m chains of jobs, containing $n_1, n_2, \ldots, n_m$ jobs, respectively, where $\sum_{j=1}^m n_j = n$. Denote those jobs as presented in Fig. 2. Without loss of generality, we may assume that if $n_j = 0$ for some j, then the chain is empty and the index j is omitted.

Given such a set of chains, one can find a single-machine schedule that is optimal in the sense of the $\sum C_i$ objective function, based on Algorithm 5 by Conway et al. (1967). This algorithm requires $O(n^2)$ time and is presented in its original form, i.e. as the list of steps.

Algorithm 5

Input: The set of chains of jobs, as described in Fig. 2, with the processing time of each job $J_{j,k}$ equal to $p_{j,k}$
Output: An optimal schedule for the 1|chains|$\sum C_i$ problem

Step 1. For each job $J_{j,k}$ compute

$$x_{j,k} = \frac{\sum_{l=1}^{k} \bar{p}_{j,l}}{k}.$$

Step 2. For each chain $j = 1, \ldots, m$ compute $y_j = \min_k x_{j,k}$ and $h_j = \arg\min_k x_{j,k}$.
Step 3. Choose j' such that $y_{j'} \leq y_j$ for all possible values of j and place the first $h_{j'}$ jobs from chain j' in the schedule.
Step 4. Neglecting all scheduled jobs, reindex the remaining jobs and recompute $y_{j'}$ and $h_{j'}$.
Step 5. Repeat Steps 3 and 4 until all the jobs are scheduled.

Algorithm 5 can be adapted to the Pm|chain, $p_{j,r} = \varphi^j(r)|\sum C_i$ problem in the following way. Given an instance of the latter problem, with a chain of length n, we imagine any feasible schedule. In such a schedule n_j jobs are executed on machine P_j, with their actual processing times equal to $\varphi^j(1), \varphi^j(2), \ldots, \varphi^j(n_j)$, respectively. This is so for each $1 \le j \le m$. As no two jobs are executed at the same time, this can be viewed as a single-machine scheduling problem, in which we schedule m independent chains of jobs with fixed processing times to minimize the value of $\sum C_i$.

Let us be given an instance of the Pm|chain, $p_{j,r} = \varphi^j(r)|\sum C_i$ problem. Also, let $(n_1, n_2, \ldots, n_m)$ be a sequence of non-negative integers such that $\sum_{j=1}^m n_j = n$. For such a sequence, we generate an instance of the 1|chains|$\sum C_i$ problem, in which we have at most m chains of jobs. The j-th chain consists of n_j jobs, $J_{j,1} \rightarrow J_{j,2} \rightarrow \ldots \rightarrow J_{j,n_j}$, with their fixed processing times equal to $\varphi^j(1), \varphi^j(2), \ldots, \varphi^i(n_j)$, respectively. Given a single-machine schedule that provides the lowest value of $\sum C_i$ within all the $(n_1, n_2, \ldots, n_m)$ sequences, a corresponding m-machine optimal schedule with the same value of $\sum C_i$ can be generated, as shown in Algorithm 6.

Algorithm 6

Input: The set of $\varphi^j(r)$ values for $r \in \{1, 2, \ldots, n\}$ and $j \in \{1, 2, \ldots, m\}$
Output: An optimal schedule for the Pm|chain, $p_{j,r} = \varphi^j(r)|\sum C_i$ problem

```
1: $T_s \leftarrow$ An empty single-machine schedule
2: $\mathcal{J}_s \leftarrow \{\}$
3: $C_s \leftarrow \infty$
4: for each sequence $(n_1, n_2, \ldots, n_m)$ such that $\sum_{j=1}^m n_j = n$ do
5:     $\mathcal{J} \leftarrow \{\}$
6:     for $j \leftarrow 1, 2, \ldots m$ do
7:         CREATE a chain of jobs $J_{j,1} \rightarrow J_{j,2} \rightarrow \cdots \rightarrow J_{j,n_j}$ such that $\bar{p}_{j,k} = \varphi^j(k)$.
8:         ADD the $J_{j,1} \rightarrow J_{j,2} \rightarrow \cdots \rightarrow J_{j,n_j}$ chain to $\mathcal{J}$
9:     FIND an optimal single-machine schedule $T$ for $\mathcal{J}$, based on Alg. 5.
10:    if $\sum C_i(T) < C_s$ then
11:        $T_s \leftarrow T$
12:        $\mathcal{J}_s \leftarrow \mathcal{J}$
13:        $C_s \leftarrow \sum C_i(T)$
14: $T \leftarrow$ An empty schedule for $m$ identical parallel machines
15: for $i \leftarrow 1, 2, \ldots, n$ do
16:     $J_{j,k} \leftarrow$ The $i$-th job in the $T_s$ schedule, as represented in the $\mathcal{J}_s$ set
17:                                   ▷ The processing time of job $J_{j,k}$ is equal to $\varphi^j(k)$.
18:     ASSIGN the $i$-th job from the original chain of jobs to machine $P_j$.
19:                                   ▷ The processing time of this job will be equal to $\varphi^j(k)$.
20: return $T$
```

Notice that inside of Algorithm 6, we perform Algorithm 5 $O(n^{m-1}/(m-1)!)$ times. As the latter algorithm needs $O(n^2)$ operations, the first **for** loop (line 4) requires $O(n^{m+1}/(m-1)!)$ operations in total. The second **for** loop (line 15) can be performed in $O(n)$ time. So, Algorithm 6 requires a total running time of $O(n^{m+1})$.

6 Conclusions

We considered a set of parallel-machine scheduling problems with jobs that are position-dependent. However, the actual processing time of a job might also freely depend on the job itself and a machine it had been assigned to. We analyzed the cases of independent and chained jobs, with maximum completion time and total completion time as the objective functions.

As the $Pm|p_{i,j,r} = \varphi_i^j(r)|C_{\max}$ problem is $\mathcal{NP}$-Hard in general, we proposed a number of polynomial algorithms that solve some of its special cases. Those special cases have been obtained by reducing the number of different φ functions. In case of the $Pm|p_{i,j,r} = \varphi_i^j(r)|\sum C_i$ problem, we presented a full set of polynomial algorithms for it an its subproblems. Then, for the case of chained jobs, we presented a series of polynomial algorithms for the $Pm|\text{chain}, p_{j,r} = \varphi^j(r)|C_{\max}$ and $Pm|\text{chain}, p_{j,r} = \varphi^j(r)|\sum C_i$ problems.

Although the presented results are theoretical, they can be used in real life. As the analyzed model is very general, it covers various job-, machine-, and position-dependent models—including the ones that might have not been discussed yet. For example, the presented results are true for classic learning and aging models (e.g. $p_{i,r} = p_i \cdot r^{a_i}$, where $a_i < 0$ or $a_i > 0$, respectively). They also remain valid when DeJong's learning effect is considered. Our results can be a base for further research on parallel-machine scheduling of variable-time jobs for which no assumptions on the model of positional-dependency are made.

There are still some open questions related to the problems discussed in this paper. Namely, we now focus on answering the questions of whether the Pm|chain, $p_{i,j,r} = \varphi_i^j(r)|C_{\max}$ and Pm|chain, $p_{i,j,r} = \varphi_i^j(r)|\sum C_i$ problems are polynomially-solvable. It is known that the answer is positive for job processing times that are only job- and machine-dependent. However, for position-dependent jobs, the answer requires some further analysis.

Acknowledgements The author would like to thank anonymous reviewers for their detailed comments, which greatly helped to improve the quality of the paper.

References

Agnetis A, Billaut JC, Gawiejnowicz S, Pacciarelli D, Soukhal A (2014) Multiagent scheduling. Models and algorithms. Springer. https://doi.org/10.1007/978-3-642-41880-8

Azzouz A, Ennigrou M, Ben Said L (2017) Scheduling problems under learning effects: classification and cartography. Int J Prod Res 56(4):1642–1661. https://doi.org/10.1080/00207543.2017.1355576

Bansal N, Srinivasan A, Svensson O (2019) Lift-and-round to improve weighted completion time on unrelated machines. SIAM J Comput STOC16-138–STOC16–159. https://doi.org/10.1137/16M1099583
Biskup D (1999) Single-machine scheduling with learning considerations. Eur J Oper Res 115(1):173–178. https://doi.org/10.1016/S0377-2217(98)00246-X
Brodal GS, Lagogiannis G, Tarjan RE (2012) Strict Fibonacci heaps. In: Proceedings of the forty-fourth annual ACM symposium on theory of computing, association for computing machinery, STOC'12, pp 1177–1184. https://doi.org/10.1145/2213977.2214082
Bruno J, Coffman E Jr, Sethi R (1974) Scheduling independent tasks to reduce mean finishing time. Commun ACM 17:382–387
Burkard R, DellAmico M, Martello S (2012) Assignment problems. Society for industrial and applied mathematics. https://doi.org/10.1137/1.9781611972238
Conway R, Maxwell W, Miller L (1967) Theory of scheduling. Addison-Wesley Publishing Company
Debczynski M, Gawiejnowicz S (2013) Scheduling jobs with mixed processing times, arbitrary precedence constraints and maximum cost criterion. Comput Indus Eng 64(1):273–279. https://doi.org/10.1016/j.cie.2012.10.010
DeJong J (1957) The effects of increasing skill on cycle time and its consequences for time standards. Ergonomics 1(1):51–60. https://doi.org/10.1080/00140135708964571
Fredman M, Tarjan R (1987) Fibonacci heaps and their uses in improved network optimization algorithms. J ACM 34(3):596–615. https://doi.org/10.1145/28869.28874
Gawiejnowicz S (1996) A note on scheduling on a single processor with speed dependent on a number of executed jobs. Inform Process Lett 57(6):297–300. https://doi.org/10.1016/0020-0190(96)00021-X
Gawiejnowicz S (2020) Models and algorithms of time-dependent scheduling. Springer. https://doi.org/10.1007/978-3-662-59362-2
Huang X, Wang MZ (2014) Single machine group scheduling with time and position dependent processing times. Optim Lett 8(4):1475–1485. https://doi.org/10.1007/s11590-012-0535-z
Ji M, Yao D, Yang Q, Cheng T (2015) Machine scheduling with DeJongs learning effect. Comput Indus Eng 80:195–200. https://doi.org/10.1016/j.cie.2014.12.009
Lenstra J, Shmoys S, Tardos É (1990) Approximation algorithms for scheduling unrelated parallel machines. Math Program 46(1–3):259–271
Liu X, Lu S, Pei J, Pardalos PM (2018) A hybrid VNS-HS algorithm for a supply chain scheduling problem with deteriorating jobs. Int J Prod Res 56(17):5758–5775. https://doi.org/10.1080/00207543.2017.1418986
Lu S, Liu X, Pei J, Thai M, Pardalos P (2018) 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. https://doi.org/10.1016/j.asoc.2018.02.018
Mosheiov G (2008) Minimizing total absolute deviation of job completion times: extensions to position-dependent processing times and parallel identical machines. J Oper Res Soc 59:1422–1424. https://doi.org/10.1057/palgrave.jors.2602480
Mosheiov G, Sidney J (2003) Scheduling with general job-dependent learning curves. Eur J Oper Res 147(3):665–670. https://doi.org/10.1016/S0377-2217(02)00358-2
Okołowski D, Gawiejnowicz S (2010) Exact and heuristic algorithms for parallel-machine scheduling with DeJongs learning effect. Comput Indus Eng 59(2):272–279. https://doi.org/10.1016/j.cie.2010.04.008
Przybylski B (2017) Precedence constrained parallel-machine scheduling of position-dependent jobs. Optimi Lett 11(7):1273–1281. https://doi.org/10.1007/s11590-016-1075-8
Schulz A, Skutella M (2002) Scheduling unrelated machines by randomized rounding. SIAM J Disc Math 15(4):450–469. https://doi.org/10.1137/S0895480199357078
Strusevich V, Rustogi K (2017) Scheduling with time-changing effects and rate-modifying activities. Springer. https://doi.org/10.1007/978-3-319-39574-6
Sun L, Ning L, Huo J (2020) Group scheduling problems with time-dependent and position-dependent DeJongs learning effect. Math Prob Eng 1–8. https://doi.org/10.1155/2020/5161872
Wang JB (2010) Single-machine scheduling with a sum-of-actual-processing-time-based learning effect. J Oper Res Soc 61(1):172–177. https://doi.org/10.1057/jors.2008.146
Wang JB, Wang JJ (2013) Single-machine scheduling with precedence constraints and position-dependent processing times. Appl Math Modelll 37(3):649–658. https://doi.org/10.1016/j.apm.2012.02.055
Wang JB, Gao M, Wang JJ, Liu L, He H (2020) Scheduling with a position-weighted learning effect and job release dates. Eng Optim 52(9):1475–1493. https://doi.org/10.1080/0305215X.2019.1664498

Wright T (1936) Factors affecting the cost of airplanes. J Aeron Sci 3(4):122–128. https://doi.org/10.2514/8.155

Zhang X, Liao L, Zhang W, Cheng T, Tan Y, Ji M (2018) Single-machine group scheduling with new models of position-dependent processing times. Comput Indus Eng 117:1–5. https://doi.org/10.1016/j.cie.2018.01.002