European Journal of Operational Research 295 (2021) 823–837

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

Solution methods for scheduling problems with sequence-dependent deterioration and maintenance events

Maxence Delorme [a,*], Manuel Iori [b], Nilson F. M. Mendes [b]

[a] Department of Econometrics and Operations Research, Tilburg University, the Netherlands
[b] Department of Sciences and Methods for Engineering, University of Modena and Reggio Emilia, Italy

ARTICLE INFO

Article history:
Received 6 July 2020
Accepted 2 March 2021
Available online 29 April 2021

Keywords:
Scheduling
Deterioration
Maintenance
Mathematical models
Metaheuristic

ABSTRACT

In this work, we study the problem of scheduling jobs and maintenance activities on a set of unrelated parallel machines, by considering that the processing time of a job increases according to a deterioration factor that depends both on the machine and on the set of jobs the machine has processed since its last maintenance. The objective we consider is to minimize the makespan. We introduce four mixed integer linear programming models, two of which using big-M constraints and the other two using an exponential number of variables. We also propose an iterated local search metaheuristic to tackle large size instances and we provide empirical evidence of the performance of the proposed approaches by means of extensive computational experiments.

© 2021 The Author(s). Published by Elsevier B.V.
This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/)

1. Introduction

Fatigue and deterioration might severely affect human and machine performances, causing an increase in the number of errors they make, in the quantity of resources they waste, or in the processing time of the jobs they perform. In order to alleviate this issue, work stops or maintenance events can be scheduled, so as to recover the full productivity of the agent and improve the overall performance of the system.

In this paper, we deal with the problem of scheduling jobs and maintenance activities on a set of unrelated parallel machines, where the processing time of each job on a given machine depends on the initial processing time of the job on the machine (i.e., its processing time if the machine was working at full performance) and on the deterioration level of the machine. The machines are not identical, which means that a job may have two different processing times and deterioration factors on two different machines. The duration of a maintenance also depends on the machine on which it is performed, but it is not impacted by the current deterioration level of the machine. The objective is to find a feasible schedule in which all jobs are processed and the makespan (i.e., the last completion time of a job) is minimized. This type of problems occurs, for example, in construction industry, where the difficulty of a given job impacts the workers' level of tiredness and thus increases the time they need to accomplish subsequent jobs, and in the cutting industry, where the hardness of a material deteriorates the cutting tools and increases the time required to cut other materials (see Ruiz-Torres, Paletta, & M'Hallah, 2017). In both cases, a short time in which the agent is not operating (such as a break for the worker or a maintenance operation for the cutting tools) is enough to restore full productivity.

In terms of contributions, we propose a linearization of the Mixed Integer Non-Linear Programming (MINLP) model originally proposed by Ruiz-Torres et al. (2017) and three additional Mixed Integer Linear Programming (MILP) models: an improvement of our first model that significantly reduces symmetry, a model based on the classical set covering formulation, and a model inspired by the arc flow formulation by Valério de Carvalho (1999). Our resulting models can be solved by invoking a standard MILP solver and obtain optimal solutions for small size instances. For larger instances, we propose a metaheuristic procedure based on the concept of *Iterated Local Search* (ILS). We show, trough extensive computational experiments, that we can obtain good quality solutions in a few seconds for a variety of instances, considering a large range of deterioration rates, processing times, and maintenance times.

The remainder of the paper is organised as follows. Section 2 provides a literature review on scheduling problems with deterioration and maintenance activities. We formally describe our problem in Section 3, and we introduce the mathematical models in Section 4. The ILS is presented in Section 5. The outcome of extensive computational experiments assessing

* Corresponding author.
E-mail address: m.delorme@tilburguniversity.edu (M. Delorme).

https://doi.org/10.1016/j.ejor.2021.03.067
0377-2217/© 2021 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/)

the quality of our approaches is reported in Section 6. Finally, we draw some conclusions and discuss interesting future research directions in Section 7.

2. Literature review

The literature on scheduling problems with deterioration can be split into two main groups, according to how the deterioration is estimated. The first group focuses on job deterioration and uses the paradigm that the jobs processed at a later stage require an additional time with respect to the jobs processed at an early stage (e.g, due to some physical properties). In such a case, we adopt the term *time-dependent* deterioration. The second group focuses on machine deterioration and adopts the paradigm that processing a job deteriorates some components of the machine, and this makes the processing time of subsequent jobs longer. In such a case, we adopt the term *sequence-dependent* deterioration.

Scheduling problems with time-dependent deterioration have been investigated since the late Eighties. To the best of our knowledge, the first study was presented by Gupta and Gupta (1988), and focused on a single machine problem in which the duration of a job depends on its starting time. The authors mentioned relevant applications in chemical and metallurgical processes, where the temperature of the material cools down if it is not used immediately, requiring a longer time to be processed. This seminal study was followed in the next years by Kunnathur and Gupta (1990), who proposed algorithms based on dynamic programming and branch-and-bound, by Browne and Yechiali (1990), who analyzed the effects of different deterioration schemes and derived optimal scheduling policies that minimize either the expected makespan or its variance, by Mosheiov (1991, 1994, 1996), who outlined theoretical properties on the optimal job sequence, and by Kubiak and van de Velde (1998), who studied the case in which the supplementary time caused by deterioration is bounded.

The problem where multiple parallel machines are available was studied by Mosheiov (1995, 1998), who proved that makespan minimization with linear deterioration is an $\mathcal{NP}$-hard problem if there are at least two machines. He also introduced several compact MILP formulations and heuristic algorithms. More recently, Ji and Cheng (2008) proposed a polynomial-time approximation scheme for the case in which the number of machines is fixed.

The problem where the deterioration function works through steps (i.e., the processing time of a job changes only if it is processed after a given deadline) was studied by Cheng and Ding (2001) for the single machine case, and by Leung, Ng, and Cheng (2008) and Lalla-Ruiz and Voß (2016) for the multiple machine case.

Surveys on scheduling with time-dependent deterioration were proposed by Alidaee and Womer (1999) and Cheng, Ding, and Lin (2004). We also refer the reader to the recent survey proposed by Gawiejnowicz (2020) for an up-to-date overview of this class of problems.

To the best of our knowledge, the first paper on scheduling with sequence-dependent deterioration was proposed by Ruiz-Torres, Paletta, and Pérez (2013), who studied the unrelated parallel machine case with the objective of makespan minimization. They showed that the problem is $\mathcal{NP}$-hard and proposed an MINLP model and a simulated annealing algorithm. A few years later, Santos and Arroyo (2017) proposed an ILS and an ILS combined with a random variable neighborhood descent algorithm, and showed that their algorithms computationally outperformed the one proposed in Ruiz-Torres et al. (2013) on both small size instances (50 jobs and 10 machines) and large size instances (150 jobs and 20 machines). In the same period, Araújo, Dhein, and Fampa (2017) linearized the model proposed in Ruiz-Torres et al. (2013) and improved its performance so that it was able to solve exactly instances with up to 50 jobs and 10 machines. More recently, Ding, Shen, Lü, and Peng (2019) studied a similar problem in which the deterioration level of a machine when processing a job depends on the deterioration factor of the job itself in addition to those of the jobs already processed. The authors studied the objectives of minimizing the makespan and minimizing the weighted completion time. They proposed an ejection chain algorithm and tested it on instances with up to 50 jobs and 10 machines.

Maintenance activities were also studied by Kuo and Yang (2008), Zhao and Tang (2010), Yang (2011), and Yang, Cheng, Yang, and Hsu (2012) in the context of scheduling with *position-dependent* deterioration. Position-dependent deterioration can be considered as a special case of sequence-dependent deterioration in which the additional time required to process a job depends only on the number of jobs that were processed since the last maintenance (and not on the type of jobs, as in the sequence-dependent case). While Kuo and Yang (2008) and Zhao and Tang (2010) focused on the single machine problem, Yang (2011) and Yang et al. (2012) dealt with the multiple machine case. In all these problems, as in ours, a maintenance activity restores the full productivity of a machine.

We note, in addition, that the term “maintenance activity” is widely used in the scheduling literature, sometimes with different meanings with respect to the one we adopt in our work. The term might refer, for example, to a mandatory operation (see, e.g., Nesello, Subramanian, Battarra, & Laporte, 2018b) or to a *rate modifying* (rm) operation that changes the processing time of subsequent jobs (either by speeding them up or slowing them down).

A large stream of the literature is dedicated to rm activities. To the best of our knowledge, the concept was introduced by Lee and Leon (2001) to describe a scheduling behavior in an electronic assembly line. In their single machine scheduling problem, each job had two possible processing times, depending on whether it was scheduled before or after the rm activity. The problem was then to decide the job ordering and the position of the rm activity, if necessary. This study was extended by Mosheiov and Sidney (2003), who addressed a similar problem with the addition of precedence constraints. They also studied the problem with the addition of a learning effect, which can be seen as the opposite of a position-dependent deterioration (i.e., processing a job shortens the duration of every subsequent job). Some recent studies, as Wang and Li (2017) and Lu, Liu, Pei, Thai, and Pardalas (2018), consider the case in which the rm activity has an execution time that linearly depends on its starting time. We refer the interested reader to the recent book by Strusevich and Rustogi (2017) for further details on rm activities.

The first work in which sequence-dependent deterioration and maintenance activities were studied together was proposed by Ruiz-Torres et al. (2017). In their paper, the deterioration level of a machine when processing a job depends only on the types of jobs that the machine has processed since its last maintenance. The authors studied the case of identical machines. They introduced an MINLP model to describe the problem and proposed some constructive heuristics to find good quality solutions. In our work, which extends the one by Ruiz-Torres et al. (2017), we propose new mathematical models and metaheuristic algorithms for scheduling problems with sequence-dependent deterioration and maintenance events. All our techniques are valid for the general case of unrelated machines. A preliminary version of our work containing the model of Section 4.2 and a prior ILS implementation was presented in Mendes and Iori (2019).

3. Problem description

Let $J = \{1, 2, \ldots, n\}$ be a set of independent jobs, all available at the beginning of the working horizon, to be processed on a

Table 1
Ideal processing times, delay factors, and maintenance times for Example 1.

machine	t_i	ideal processing time (p_{ij})				delay factor (d_{ij})			
		$j=1$	$j=2$	$j=3$	$j=4$	$j=1$	$j=2$	$j=3$	$j=4$
$i=1$	2	10	20	10	30	1.2	1.1	1.1	1.2
$i=2$	5	20	10	10	30	1.1	1.2	1.1	1.1

set $M = \{1, 2, \ldots, m\}$ of unrelated parallel machines subject to deterioration. Let p_{ij} be the ideal processing time of job j ($j \in J$) on machine i ($i \in M$), that is, the processing time when the machine is fully operative, so either at the beginning of the time horizon or right after a maintenance event has occurred. Let also $d_{ij} \geq 1$ be the *delay factor* caused by the deterioration of machine i after processing job j. As the delay factors are multiplicative, let $\delta_{ij} \geq 1$ be the *accumulated delay factor* due to the deterioration of machine i because of all the jobs it has processed before j since the last maintenance. This sequence-dependent deterioration (which is also made clear by means of Example 1 at the end of this section) can be described as follows:

- The actual processing time of job j on machine i is equal to $p_{ij}\delta_{ij}$.
- The accumulated delay factor caused by the deterioration of machine i after processing job j is equal to $\delta_{ij}d_{ij}$.

It can be observed that the accumulated delay factor at the start of job j (i.e., δ_{ij}) is equal to the product of the delay factors of all jobs processed before j since the last maintenance.

Let t_i be the duration of a maintenance event that returns machine i to its fully operative state (i.e., $\delta_{ij} = 1$, for all j processed right after a maintenance activity). A single machine cannot process a job and perform a maintenance activity at the same time. In addition, preemption is not allowed, so a machine cannot be interrupted while it is processing a job to perform a maintenance activity. There is no theoretical limit on the number of maintenance activities that can be performed on a single machine (even if in practice every relevant schedule has at most $n-1$ maintenance activities), nor on the number of maintenance activities that can be performed in parallel on all machines.

Each job must be assigned to a machine in such a way that the makespan (i.e., the maximum completion time among all jobs) is minimized. By summing the actual processing times of the jobs and the maintenance times, we can easily compute the completion time of each machine and the makespan. The insertion of idle times on the machine schedules cannot decrease the makespan because all jobs are available at time 0 and there are no precedence constraints. Thus, there is always an optimal solution without idle times.

According to the three-field notation by Graham, Lawler, Lenstra, and Kan (1979), this problem can be denoted as R|Sdd,mnt|C_{max}, where "R" stands for unrelated parallel machines, "Sdd" for sequence-dependent deterioration, "mnt" for maintenance, and "C_{max}" for makespan minimization. The R|Sdd,mnt|C_{max} is strongly $\mathcal{NP}$−hard because it generalizes the well-known R||C_{max}, already proven to be strongly $\mathcal{NP}$-hard in Pinedo (2016). In the following, we introduce an example that will be resumed in the next sections to outline the behavior of our models.

Example 1. Let us consider the following instance with two machines and four jobs, whose ideal processing times, delay factors, and maintenance durations are displayed in Table 1.

As shown in Fig. 1, jobs 1 and 3 are scheduled on machine 1 separated by a maintenance activity (represented in white in the figure). Job 4 is scheduled first on machine 2, followed by job 2. The actual processing time of job 2 is 11 instead of 10 because of the deterioration caused by job 4 (represented by hashed lines in the figure). The optimal makespan is 41. Note that we included a maintenance on machine 1 between job 1 and job 3 for descriptive purposes. One could also skip the maintenance and process job 3 first, followed by job 1, and obtain a load of 21 units on machine 1. In any case, the optimal makespan would remain 41 as the workload on machine 2 is unchanged.

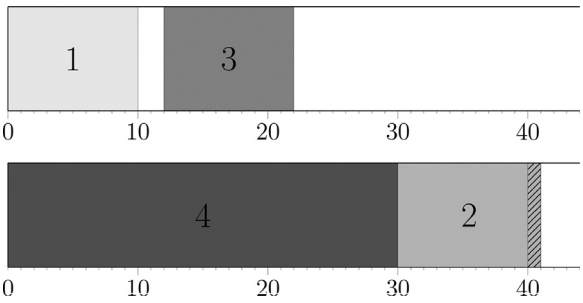

Fig. 1. Optimal job scheduling for Example 1 (machine 1 on top, machine 2 at the bottom).

4. Mathematical models

In this section, we present the MINLP model by Ruiz-Torres et al. (2017), introduce its linearized version, and then propose three novel MILP models. To ease notation, if the meaning of a variable is the same in several models, then we maintain the same name.

4.1. Non-linear position-based formulation

The model of Ruiz-Torres et al. (2017) decomposes each machine into slots $H = \{1, 2, \ldots, |H|\}$. A slot defines the position of an activity (a job or a maintenance) on a machine schedule and has no pre-determined duration. Since we do not know a priori the number of slots required by each machine, we use a valid upper bound $|H| = 2n - 1$ to estimate it. Indeed, in the worst case a single machine processes all the jobs and performs a maintenance activity between each pair of jobs. Because the number of slots per machine i is generally larger than the number of activities assigned to i, some of the slots on i may remain free (i.e., with no assigned activity). Such slots are herein called *empty slots.* Better upper bounds on H can be derived, using the form $|H| = 2(n - m') + 1$, if we can prove that there is always an optimal solution in which m' machines perform at least one job, or, in other words, in which no single machine performs strictly more than $n - m' + 1$ jobs, so resulting in at most $n - m'$ maintenance operations. In particular, if the ideal processing time of each job is the same on every machine, then we have $m' = m$.

Let x_{ijh} be a binary variable taking the value 1 if job j is executed in slot h of machine i, and 0 otherwise. Similarly, let s_{ih} be a binary variable taking the value 1 if a maintenance is executed in slot h of machine i, and 0 otherwise, and let q_{ih} be a continuous variable that reports the current *performance ratio* of machine i in slot h. The performance ratio is simply the inverse of the accumulated delay factor δ_{ij}: it satisfies $0 < q_{ih} \leq 1$ and the actual processing time of job j on machine i in slot h is equal to

$\frac{p_{ij}}{q_{ih}}$, while the performance ratio of machine i after processing job j in slot h is equal to $q_{ih}(1-d'_{ij})$.

Parameter d'_{ij} is called *deterioration effect* and satisfies $0 \leq d'_{ij} < 1$. It is adopted by Ruiz-Torres et al. (2017) to replace delay factor d_{ij} by imposing the relation $d_{ij} = \frac{1}{1-d'_{ij}}$. For example, a job j whose deterioration effect on machine i is $d'_{ij} = 0.01$ has a delay factor on machine i of $d_{ij} = \frac{1}{0.99} \approx 1.0101$.

By using these variables and an additional continuous variable C_{max} indicating the value of the makespan, the R|Sdd,mnt|C_{max} can be modeled as:

$$\min \quad C_{max} \tag{1}$$

$$\text{s.t.} \quad \sum_{j\in J} x_{ijh} + s_{ih} \leq 1 \quad \forall i \in M, h \in H, \tag{2}$$

$$\sum_{i\in M}\sum_{h\in H} x_{ijh} = 1 \quad \forall j \in J, \tag{3}$$

$$\sum_{j\in J}\sum_{h\in H} \frac{p_{ij}}{q_{ih}} x_{ijh} + \sum_{h\in H} t_i \; s_{ih} \leq C_{max} \quad \forall i \in M, \tag{4}$$

$$x_{ijh} \leq \sum_{j'\in J} x_{ij',h-1} + s_{ih} \quad \forall i \in M, j \in J, h \in H\backslash\{1\}, \tag{5}$$

$$q_{i,h-1} \sum_{j\in J} (1-d'_{ij}) \; x_{ij,h-1} + s_{i,h-1} = q_{ih} \quad \forall i \in M, h \in H\backslash\{1\}, \tag{6}$$

$$q_{i1} = 1 \quad \forall i \in M, \tag{7}$$

$$x_{ijh} \in \{0,1\} \quad \forall i \in M, j \in J, h \in H, \tag{8}$$

$$s_{ih} \in \{0,1\} \quad \forall i \in M, h \in H, \tag{9}$$

$$q_{ih} \geq 0 \quad \forall i \in M, h \in H, \tag{10}$$

$$C_{max} \geq 0. \tag{11}$$

Objective function (1) imposes the minimization of the makespan. Constraints (2) state that each slot can either accommodate a job, a maintenance activity, or remain empty. Constraints (3) impose each job to be processed by a machine. Constraints (4) make sure that the maskespan is equal to or greater than the finishing time of each machine. Constraints (5) force the empty slots to be positioned at the end of the planning horizon. Constraints (6) compute the performance ratio of each machine in each slot through an inductive process based on the previous slot of the machine. Constraints (7) impose the machines to be fully operative at the beginning of the planning horizon. It can be noticed that constraints (4) and (6) are non-linear.

4.2. Linearized position-based formulation

The first MILP formulation that we propose is derived from the work by Araújo et al. (2017), who modeled R|Sdd|C_{max} the version of our problem without maintenance activities. It uses an accumulated delay factor δ_{ih} instead of the performance ratio q_{ih}, and the delay factor d_{ij} instead of the deterioration effect d'_{ij}. It also uses a continuous variable a_{ijh} that indicates the actual processing time of job j on machine i at time slot h, and "big-M" constraints that force a_{ijh} to take (at least) value $p_{ij}\delta_{ih}$ when job j is assigned to slot h on machine i, and 0 otherwise. The linearized position-based formulation is as follows:

$$\min \quad C_{max} \tag{12}$$

$$\text{s.t.} \quad \sum_{j\in J} x_{ijh} + s_{ih} \leq 1 \quad \forall i \in M, h \in H, \tag{13}$$

$$\sum_{i\in M}\sum_{h\in H} x_{ijh} = 1 \quad \forall j \in J, \tag{14}$$

$$a_{ijh} \geq p_{ij}\delta_{ih} - K^a_{ij}(1-x_{ijh}) \quad \forall i \in M, j \in J, h \in H, \tag{15}$$

$$\sum_{j\in J}\sum_{h\in H} a_{ijh} + \sum_{h\in H} t_i s_{ih} \leq C_{max} \quad \forall i \in M, \tag{16}$$

$$\sum_{j\in J} x_{ijh} + s_{ih} \leq \sum_{j\in J} x_{ij,h-1} + s_{i,h-1} \quad \forall i \in M, h \in H\backslash\{1\}, \tag{17}$$

$$d_{ij}\delta_{i,h-1} - K^b_{ij}(s_{i,h-1} + 1 - x_{ij,h-1}) \leq \delta_{ih} \quad \forall i \in M, j \in J, h \in H\backslash\{1\}, \tag{18}$$

$$\delta_{i1} = 1 \quad \forall i \in M, \tag{19}$$

$$x_{ijh} \in \{0,1\} \quad \forall i \in M, j \in J, h \in H, \tag{20}$$

$$s_{ih} \in \{0,1\} \quad \forall i \in M, h \in H, \tag{21}$$

$$a_{ijh} \geq 0 \quad \forall i \in M, j \in J, h \in H, \tag{22}$$

$$\delta_{ih} \geq 1 \quad \forall i \in M, h \in H, \tag{23}$$

$$C_{max} \geq 0. \tag{24}$$

Objective function (12) and constraints (13) and (14) are identical to those reported in the MINLP model of Section 4.1. Constraints (15) are used to define the actual processing time of job j in slot h of machine m. Coefficient K^a_{ij} is a large constant defined for any machine i and job j in such a way that a_{ijh} is allowed to take value 0 when $x_{ijh} = 0$ and forced to take at least value $p_{ij}\delta_{ih}$ when $x_{ijh} = 1$. Constraints (16) are the linearized version of constraints (4), and are used to calculate the makespan. Constraints (17) are the counterparts of constraints (5) and prevent a job or a maintenance activity to be scheduled after an empty slot (slot h on machine i is empty if $\sum_{j\in J} x_{ijh} + s_{ih} = 0$). In other words, all the empty slots of a given machine are forced to be positioned at the end of the planning horizon. Worth is mentioning that, while constraints (5) have the purpose of removing symmetrical solutions, constraints (17) are necessary for the correctness of the model, because inserting an empty slot in the middle of the planning horizon would be equivalent to performing an instantaneous maintenance activity. Constraints (18) are the counterpart of constraints (6) and compute the accumulated delay factor of each machine in each time slot through an inductive process. The coefficient K^b_{ij} is a sufficiently large constant defined for any machine i and job j, so that δ_{ih} is allowed to take the value 0 when $x_{ij,h-1} = 0$. Constraints (19) impose the machines to be fully operative at the beginning of the planning horizon, but are not necessary for the correctness of the model.

In the following, we provide a property about the optimal scheduling structure derived from Lemma 1 in Ruiz-Torres et al. (2017), we explain how to compute good values for the big-M coefficients, and we outline the weaknesses of the model.

Property 1. *There is always an optimal solution in which a job j is scheduled in a position h of a machine i in such a way that $(\delta_{ih}-1)p_{ij} \leq t_i$ holds.*

Proof. Consider a solution S in which a job j is scheduled in slot h on machine i such that $(\delta_{ih}-1)p_{ij} > t_i$, and let Q be the time required by machine i to process all the activities until it has finished job j. The alternative solution S' in which machine i schedules a maintenance in slot h and processes job j in slot $h+1$ takes $Q' = Q - \delta_{ih}p_{ij} + t_i + p_{ij}$. It follows that $Q - Q' = (\delta_{ih}-1)p_{ij} - t_i$, which is strictly positive according to the initial condition. Finally, because all delay factors d_{ij} are greater than or equal to 1, we can simply show that the accumulated delay factor in the original solution S after processing job j in slot h, which is $\delta_{i,h+1} = \delta_{ih}d_{ij}$, cannot be smaller than the accumulated delay factor in solution S' after performing a maintenance in slot h and processing job j in slot $h+1$, which is $\delta_{i,h+2} = d_{ij}$. As a result, because the two solutions S and S' have the same job ordering but S' completes j sooner and with a lower accumulated delay factor, then S' is at least as good as S. □

An interesting implication of Property 1 is that there is always an optimal solution that satisfies $\delta_{ih} \leq 1 + \frac{t_i}{p_{ij}}$, $\forall i \in M, h \in H : s_{ih} = 0$. This allows us to define the big-M values as:

$$K^a_{ij} = p_{ij} \times \min\left\{\prod_{j' \in J} d_{ij'}, \left(1 + \frac{t_i}{\min_{j'}\{p_{ij'}\}}\right) \times \max_{j'}\{d_{ij'}\}\right\} \quad i \in M, j \in J, \tag{25}$$

$$K^b_{ij} = d_{ij} \times \min\left\{\prod_{j' \in J} d_{ij'}, \left(1 + \frac{t_i}{\min_{j'}\{p_{ij'}\}}\right) \times \max_{j'}\{d_{ij'}\}\right\} \quad i \in M, j \in J. \tag{26}$$

Indeed, we know that, in order to deactivate constraints (15) when variables x_{ijh} take value 0, we need coefficients K^a_{ij} to be greater than or equal to $p_{ij}\delta_{ih}$. An obvious upper bound on δ_{ih} is the product of the delay factors of all jobs. Thanks to Property 1, a refined upper bound of δ_{ih} when a job is scheduled in position h (i.e., when $s_{ih} = 0$) is $(1 + \frac{t_i}{\min_{j'}\{p_{ij'}\}})$. By multiplying this value by $\max_{j'}\{d_{ij'}\}$, we obtain an upper bound which is also valid when $s_{ih} = 1$. A similar reasoning can be applied to determine coefficients K^b_{ij}.

The linearized position-based model involves a polynomial number $O(|M||J|^2)$ of variables and a polynomial number $O(|M||J|^2)$ of constraints. However, it also has a large amount of symmetry: indeed, a subset of jobs processed between two maintenance activities can be exchanged with another subset of jobs processed between two other maintenance activities on the same machine without affecting the makespan. In the following, we introduce an event-based formulation aimed at palliating this issue.

4.3. Event-based formulation

The event-based formulation uses the notion of a *block*. A block is a sequence of jobs that occurs either before the first maintenance activity, or between two consecutive maintenance activities, or after the last maintenance activity. The model decides on which machine a job is scheduled, and whether a job initiates a block or is positioned after another job. The decisions regarding the ordering of the blocks within the same machine do not have any impact on the makespan and are thus removed from the model.

We define j_0 as a dummy job of processing time 0 and delay factor 1 that initiates each block. We use a binary variable $x_{ijj'}$ that takes the value 1 if job j' is scheduled right after job j on machine i, and 0 otherwise ($i \in M, j \in J \cup j_0, j' \in J$). For each machine, we identify sequences of jobs that can be disregarded by the model (as they are not necessary to reach an optimal solution) and store them in set $\mathcal{S}$. Continuous variables δ_j and a_{ij} now define the accumulated delay factor before starting job j and the actual processing time of job j on machine i, respectively. The event-based formulation is as follows:

$$\min \quad C_{\max} \tag{27}$$

$$\sum_{i \in M}\sum_{j \in J \cup j_0} x_{ijj'} = 1 \quad \forall j' \in J, \tag{28}$$

$$\sum_{j \in J \cup j_0} x_{ijj'} \geq \sum_{j'' \in J} x_{ij'j''} \quad \forall j' \in J, i \in M, \tag{29}$$

$$a_{ij'} \geq p_{ij'}\delta_{j'} - K^a_{ij'}(1 - x_{ijj'}) \quad \forall i \in M, j \in J, j' \in J, \tag{30}$$

$$a_{ij'} \geq p_{ij'}x_{ij_0j'} \quad \forall i \in M, j' \in J, \tag{31}$$

$$\sum_{j \in J} a_{ij} - t_i + \sum_{j' \in J} t_i x_{ij_0j'} \leq C_{\max} \quad \forall i \in M, \tag{32}$$

$$\delta_{j'} \geq \delta_j d_{ij} - K^b_{ij}(1 - x_{ijj'}) \quad \forall i \in M, j \in J, j' \in J, \tag{33}$$

$$x_{ijj'} = 0 \quad \forall (i, j, j') \in \mathcal{S} \tag{34}$$

$$\delta_{j'} \geq 1 \quad \forall j' \in J, \tag{35}$$

$$x_{ijj'} \in \{0, 1\} \quad \forall i \in M, j \in J \cup j_0, j' \in J, \tag{36}$$

$$a_{ij'} \geq 0 \quad \forall i \in M, j' \in J, \tag{37}$$

$$C_{\max} \geq 0. \tag{38}$$

Constraints (28) force each job to be scheduled on a machine, either after another job or after a dummy job j_0. Constraints (29) allow a job j'' to be scheduled after job j' on machine i only if job j' was itself scheduled after another job j or after the dummy job j_0 on machine i. Constraints (30) and (31) are used to define the actual processing time of job j' on machine i. While the former constraints are activated if j' is scheduled after another job, the latter ones are activated if j' is scheduled after the dummy job j_0. Constraints (32) are used to calculate the makespan. Note that the number of maintenance activities scheduled on a given machine is equal to the number of blocks scheduled on that machine (i.e., the number of times a job is scheduled after the dummy job j_0) minus one. Constraints (33) compute the accumulated delay factor before starting job j' and constraints (34) forbid certain jobs to be scheduled after some others on specific machines. Initially, we forbid each job to be scheduled right after itself on any machine (i.e., $\forall i \in M, j \in J, (i, j, j) \in \mathcal{S}$).

Example 1 (*resumed*)**.** The values of the variables of the event-based formulation for Example 1 in the solution depicted in Fig. 1 are as follows:

- $x_{101} = 1, \delta_1 = 1, a_{11} = 10$
- $x_{103} = 1, \delta_3 = 1, a_{13} = 10$
- $x_{204} = 1, \delta_4 = 1, a_{24} = 30$
- $x_{242} = 1, \delta_2 = 1.1, a_{22} = 11$

And constraints (32) are as follows:

- $C_{\max} \geq 10 + 10 + 2(2-1) = 22$ (load on machine 1)
- $C_{\max} \geq 30 + 11 + 5(1-1) = 41$ (load on machine 2)

In the following, we provide a property on the optimal scheduling structure that allows further triplets (i, j, j') to be added to set $\mathcal{S}$, so reducing the number of binary variables in the model. This property is derived from Lemma 1 in Ding et al. (2019) and Lemma 1 from Ruiz-Torres et al. (2013), both devoted to the case without maintenance, and was used by Ruiz-Torres et al. (2017) to derive heuristics for the case with maintenance.

Property 2. *There is always an optimal solution in which two successive jobs j and j' scheduled in the same block on a machine i satisfy the condition*

$$\frac{p_{ij}}{d_{ij}-1} \geq \frac{p_{ij'}}{d_{ij'}-1} \quad \forall i \in M, j, j' \in J : x_{ijj'} = 1. \tag{39}$$

Proof. Consider a solution S in which job j is scheduled right after job j' on machine i, in the same block, in such a way that $\frac{p_{ij}}{d_{ij}-1} > \frac{p_{ij'}}{d_{ij'}-1}$ holds. Let Q be the time required by machine i to process all activities until job j is finished. Let us also consider an alternative solution S' in which job j is scheduled on machine i just before job j', and let Q' be the time required to process all activities until job j' is finished. By denoting k the time required to process all jobs before the beginning of job j' (resp., job j) in solution S (resp., solution S'), we can define Q (resp., Q') as follows:

$$Q = k + p_{ij'} + (\delta_{j'} - 1)p_{ij'} + p_{ij} + (\delta_{j'} d_{ij'} - 1)p_{ij},$$
$$Q' = k + p_{ij} + (\delta_j - 1)p_{ij} + p_{ij'} + (\delta_j d_{ij} - 1)p_{ij'}.$$

As $\delta_{j'}$ only depends on the set of jobs performed before job j' by machine i since the last maintenance, then $\delta_{j'}$ in Q is equal to δ_j in Q'. To ease the notation, we set $\delta_j = \delta_{j'} = \delta$ in the rest of the proof. By computing the difference between the completion times of the two partial schedules, we obtain:

$$\begin{aligned} Q - Q' &= (\delta - 1)p_{ij'} + (\delta d_{ij'} - 1)p_{ij} - (\delta - 1)p_{ij} - (\delta d_{ij} - 1)p_{ij'}, \\ &= (\delta - \delta d_{ij})p_{ij'} + (\delta d_{ij'} - \delta)p_{ij}, \\ &= (1 - d_{ij})\delta p_{ij'} + (d_{ij'} - 1)\delta p_{ij}, \\ &= -(d_{ij} - 1)\delta p_{ij'} + (d_{ij'} - 1)\delta p_{ij}. \end{aligned}$$

Because of (39), we know that $\frac{p_{ij}}{d_{ij}-1} > \frac{p_{ij'}}{d_{ij'}-1}$, or, in other words, as $d_{ij} - 1$ and $d_{ij'} - 1$ are strictly positive, that $p_{ij}(d_{ij'} - 1) > p_{ij'}(d_{ij} - 1)$. Thus, as also δ is strictly positive, then $Q - Q'$ is strictly positive as well. In addition, the accumulated delay factor after processing job j in solution Q is the same as the accumulated delay factor after processing job j' in solution Q', which is $\delta d_{ij} d_{ij'}$. As a result, since the two solutions have processed the same set of jobs, but solution S' completed it sooner and with the same accumulated delay factor, solution S' is at least as good as solution S. □

Note that, for a given machine, Property 2 is a necessary but not sufficient condition for the optimality of a schedule. That is, there could be some job schedules satisfying condition (39) that do not minimize the makespan of the machine (an intuitive example is to schedule all the jobs in the same block). This differs from the version of the problem without maintenance in which, for a given machine, the job schedules minimizing the makespan of the machine always satisfy condition (39). We also outline that there may exist several optimal solutions that do not satisfy condition (39) on every machine: indeed, as the makespan only depends on the schedule of the machine(s) with largest workload, it is possible that one or more blocks in machines with smaller workload do not fulfill (39). However, Property 2 states that there is always at least one solution in which the job ordering of each block satisfies this condition. As a corollary of Property 2, we obtain:

$$x_{ijj'} = 0 \quad \forall i \in M, \forall j, j' \in J : \frac{p_{ij}}{d_{ij}-1} < \frac{p_{ij'}}{d_{ij'}-1}, \tag{40}$$

which allows us to set a large number of variables to 0 through constraints (34).

The event-based model also involves a polynomial number $O(|M||J|^2)$ of variables and a polynomial number $O(|M||J|^2)$ of constraints. It has less symmetry since the block ordering decision is removed from the model. However, its linear relaxation is weak due to the big-M constraints. In the following, we introduce a block-based formulation free of big-M constraints.

4.4. Block-based formulation

While the event-based formulation decides on which machine a job is scheduled, and whether a job initiates a block or is positioned after another job, the block-based formulation selects from a large set of feasible blocks the ones that minimize the makespan. The decisions regarding the job ordering within each block are determined before running the model in accordance with Property 2. This formulation is a natural extension of the set covering formulation of Gilmore and Gomory (1961, 1963) originally proposed for the cutting stock problem. Some formulations using the notion of blocks were already proposed for other scheduling problems (see, e.g., Pacheco, Àngel Bello, & Álvarez, 2013 for a single-machine scheduling problem with set-up times), but not all of these formulations associate a variable to each of the feasible blocks as the model presented in this section does.

We borrow the set covering notation and use $\mathcal{B}_i$ to define both the set of blocks and the set of block indices that can be scheduled on machine i. We also use b to define both a block and its index. Note that sets $\mathcal{B}_i$ and $\mathcal{B}_{i'}$ $(i, i' \in M, i \neq i')$ are not necessarily identical as shown in the example at the end of this section. The b^{th} block on machine i is described by its duration a^i_b and by an integer array $(\alpha^i_{1b}, \alpha^i_{2b}, \ldots, \alpha^i_{|J|b})$, where α^i_{jb} takes the value 1 if job j is included in the b^{th} feasible block of machine i, and 0 otherwise. We now use binary variables x^i_b that take the value 1 if the b^{th} feasible block of machine i is selected, and 0 otherwise. The block-based formulation is as follows:

$$\min \quad C_{\max} \tag{41}$$

$$\sum_{i \in M} \sum_{b \in \mathcal{B}_i} \alpha^i_{jb} x^i_b = 1 \quad \forall j \in J, \tag{42}$$

$$-t_i + \sum_{b \in \mathcal{B}_i} (a^i_b + t_i) x^i_b \leq C_{\max} \qquad \forall i \in M, \tag{43}$$

$$x^i_b \in \{0, 1\} \quad \forall i \in M, b \in \mathcal{B}_i, \tag{44}$$

$$C_{\max} \geq 0. \tag{45}$$

Constraints (42) impose that each job is contained in exactly one of the selected blocks, and constraints (43) calculate the makespan, which is minimized in (41).

In Algorithm 1, we provide a pseudo-code to exhaustively enumerate every feasible *non-dominated* block. A block b is dominated if there exists a set of blocks S containing the same jobs as b that can be processed in a shorter time (taking into account the

Algorithm 1 Create blocks.

1: **for each** $i \in M$ **do** ▷ For each machine i
2: $\mathcal{B}_i \leftarrow \emptyset$, $b = 0$ ▷ There are no blocks in $\mathcal{B}_i$
3: **for each** $j \in J$ **do**
4: $\alpha^i_{jb} \leftarrow 0$ ▷ Create the empty block
5: **end for**
6: $a^i_b \leftarrow 0, \delta^i_b \leftarrow 1, \mathcal{B}_i \leftarrow \mathcal{B}_i \cup \{b\}, b \leftarrow b+1$ ▷ Add the empty block to $\mathcal{B}_i$
7: **for each** $j \in J$ ordered by non-increasing $\frac{p_{ij}}{d_{ij}-1}$ **do** ▷ For each job
8: $\bar{\mathcal{B}}_i \leftarrow \emptyset$ ▷ There are no new blocks
9: **for each** $b' \in \mathcal{B}_i$ **do** ▷ For each existing block in $\mathcal{B}_i$
10: **for each** $j' \in J$ **do**
11: $\alpha^i_{j'b} \leftarrow \alpha^i_{j'b'}$ ▷ Block b is a copy of the existing block b'
12: **end for**
13: $\alpha^i_{jb} \leftarrow 1, a^i_b \leftarrow a^i_{b'} + \delta^i_{b'} p_{ij}, \delta^i_b \leftarrow \delta^i_{b'} d_{ij}$ ▷ add job j to block b and perform a^i_b and δ^i_b
14: **if** `isNotDominated(`b`)` **then** ▷ if the block is not removed by our reduction criterion
15: $\bar{\mathcal{B}}_i \leftarrow \bar{\mathcal{B}}_i \cup \{b\}, b \leftarrow b+1$ ▷ add block b to the new blocks, and increment b
16: **end if**
17: **end for**
18: **for each** $\bar{b} \in \bar{\mathcal{B}}_i$ **do**
19: $\mathcal{B}_i \leftarrow \cup\{\bar{b}\}$, ▷ add the new blocks to $\mathcal{B}_i$
20: **end for**
21: **end for**
22: **end for**

duration of the $|S|$ blocks and the $|S|-1$ maintenance activities in-between blocks). Dominated blocks are not useful to the model because there is always an optimal solution that does not contain any dominated block. Before describing the algorithm steps, we give its general idea: starting from $\mathcal{B}_i$ containing only the empty block, the algorithm temporarily duplicates every block in $\mathcal{B}_i$ and adds the job currently processed in each duplicate. It then discards the newly generated blocks that are dominated, inserts the remaining ones into $\mathcal{B}_i$ and moves on to the next job. As a result, $\mathcal{B}_i$ contains at most 2^j blocks at step j.

For each machine i, the algorithm sets the block counter to 0 (step 2), creates the dummy empty block of duration 0 and accumulated delay factor 1, and adds it to the set of blocks of machine i (steps 3–6). Then, for each job j (preliminary re-ordered by non-increasing $\frac{p_{ij}}{d_{ij}-1}$ values), it creates a temporary new set $\bar{\mathcal{B}}_i$ in which it stores the new blocks generated when adding job j (step 8). After this, each original block in $\mathcal{B}_i$ is copied, one at a time, into block b (steps 9–12). The algorithm adds job j to block b and computes its duration and its accumulated delay factor (step 13). Then, the algorithm determines if block b is dominated or not in the way described below. If block b is not dominated, then it is added to the new blocks (step 15) and the block counter is incremented. If, instead, block b is dominated, then it will simply be overwritten during the next loop. Finally, the algorithm adds every newly generated non-dominated block from $\bar{\mathcal{B}}_i$ into $\mathcal{B}_i$ (steps 18–20) and moves on to the next item j.

Assessing whether a block is dominated or not is not trivial. An easy criterion can be derived through Property 1, by checking that the duration of the new block a^i_b is strictly shorter than the duration of the old block $a^i_{b'}$ plus the ideal processing time of job j plus the duration of a maintenance activity t_i. If it is not the case, then it is shorter to perform a maintenance activity before processing j and block b is dominated (and so, it does not need to be generated). The more powerful test that we implemented consists in trying, in turns, to insert a maintenance activity between each pair of consecutive jobs and comparing the sum of the duration of these two blocks plus the duration of a maintenance activity with the duration of the new block a^i_b. If a^i_b is longer, then it is more advantageous to use two blocks separated by a maintenance activity instead of block b, and thus it is not necessary to generate b as it is dominated. Note that this test was also used in the heuristic of Ruiz-Torres et al. (2017) in order to improve incumbent solutions.

Example 1 (*resumed*)**.** The blocks generated by Algorithm 1 for Example 1 are presented in Table 2. An optimal solution provided by an ILP solver selects block 6 in machine 1 and block 9 in machine 2 for a total makespan equal to 41. Note that there exists other optimal solutions with a makespan equal to 41, such as the one obtained by selecting blocks 1 and 2 in machine 1 and block 10 in machine 2.

The block-based model involves a polynomial number of constraints $O(|M|+|J|)$ and an exponential number of variables, which empirically grows very fast as the number of jobs increases (some instances with 100 jobs led to models with more than 100 million variables in our tests). A possible option could be to only generate a subset of variables through column generation and embed the procedure in a branch-and-price algorithm. In the following, we introduce an arc flow formulation whose number of variables still grows exponentially with the number of jobs, but at a lesser extent as shown in the computational experiments section.

4.5. Arc flow-based formulation

The arc flow formulation was popularized by Valério de Carvalho (1999) for the CSP, and extensively studied afterwards in several application fields. Recently, it has been applied to cutting and packing (see, e.g., Clautiaux, Sadykov, Vanderbeck, & Viaud, 2018, Dell'Amico, Delorme, Iori, & Martello, 2019, and Delorme & Iori, 2020), production (see, e.g., Nesello, Delorme, Iori, & Subramanian, 2018a) and Ramos, Alves, & Valério de Carvalho, 2020 (forthcoming), and vehicle routing (see, e.g., Clautiaux, Hanafi, Macedo, Voge, & Alves, 2017). In the scheduling field, arc flow models were developed to solve problems on parallel machines, either to minimize weighted tardiness (Pessoa, Uchoa, Poggi, & Rodrigues, 2010) or weighted completion times (Kramer, Dell'Amico, & Iori, 2019).

In the arc flow-based formulation we propose for the R|Sdd,mnt|C_{max}, the composition of a block is defined as a path in a graph where nodes are intermediary accumulated delay factors and arcs are jobs. Formally, for each machine i, let $\mathcal{G}_i = (\mathcal{V}_i, \mathcal{A}_i)$ be a multigraph where the node set $\mathcal{N}_i$ includes every relevant intermediary accumulated delay factors. A trivial way to define $\mathcal{N}_i$ is to include every z satisfying:

$$z = \prod_{j \in J}((d_{ij}-1)y_j + 1),$$

$$y_j \in \{0,1\} \qquad \forall j \in J.$$

Example 1 (*resumed*)**.** All the possible intermediary accumulated delay factors for Example 1 are $\mathcal{N}_1 = \{1, 1.1, 1.2, 1.21, 1.32, 1.44, 1.452, 1.584, 1.7424\}$ and $\mathcal{N}_2 = \{1, 1.1, 1.2, 1.21, 1.32, 1.331, 1.452, 1.5972\}$.

To correctly represent a block, each path must start at the source node $s = 1$. There is no restriction on the node in which a path must terminate. Arcs in $\mathcal{A}_i$ are in the form (e, f, j) and have a given duration τ necessary to calculate the makespan, where j ($j \in J$) is the job index, e ($e \in \mathcal{N}_i$) is the intermediary accumulated delay factor at the beginning of job j, f ($f \in \mathcal{N}_i, f = e\ d_{ij}$) is the intermediary accumulated delay factor at the end of job j, and τ is obtained by multiplying p_{ij} (the ideal processing time of job j on machine i) by e (the accumulated delay factor at the beginning of the arc). By using a binary variable x^i_{efj} that takes the value 1 if arc (e, f, j) is selected on machine i, and 0 otherwise, the arc flow-based formulation is as follows:

$$\min \quad C_{max} \tag{46}$$

Table 2
Blocks generated by Algorithm 1 for Example 1.

Jobs	Id	Machine 1				Machine 2			
		order	a_b^i	δ_b^i	isNotDominated	order	a_b^i	δ_b^i	isNotDominated
{1}	1	1	10	1.2	true	1	20	1.1	true
{2}	2	2	20	1.1	true	2	10	1.2	true
{3}	3	3	10	1.1	true	3	10	1.1	true
{4}	4	4	30	1.2	true	4	30	1.1	true
{1,2}	5	2,1	31	1.32	true	1,2	31	1.32	true
{1,3}	6	3,1	21	1.32	true	1,3	31	1.21	true
{1,4}	7	4,1	42	1.44	false	4,1	52	1.21	true
{2,3}	8	2,3	31	1.21	true	3,2	21	1.32	true
{2,4}	9	2,4	53	1.32	false	4,2	41	1.32	true
{3,4}	10	4,3	42	1.32	false	4,3	41	1.21	true
{1,2,3}	11	2,3,1	43.1	1.452	false	1,3,2	43.1	1.452	true
{1,2,4}	12	2,4,1	66.2	1.584	false	4,1,2	64.1	1.452	true
{1,3,4}	13	4,3,1	55.2	1.584	false	4,1,3	64.1	1.331	true
{2,3,4}	14	2,4,3	66.2	1.452	false	4,3,2	53.1	1.452	true
{1,2,3,4}	15	2,4,3,1	80.72	1.7424	false	4,1,3,2	77.41	1.5972	true

$$\sum_{i\in M}\sum_{\substack{(e,f,j)\in\mathcal{A}_i\\ j=j'}} x^i_{efj} = 1 \quad \forall j' \in J, \tag{47}$$

$$\sum_{\substack{(e,f,j)\in\mathcal{A}_i\\ e=k}} x^i_{efj} \le \sum_{\substack{(e,f,j)\in\mathcal{A}_i\\ f=k}} x^i_{efj} \quad \forall i \in M, \forall k \in \mathcal{N}_i \setminus \{s\}, \tag{48}$$

$$-t_i + t_i \sum_{\substack{(e,f,j)\in\mathcal{A}_i\\ e=s}} x^i_{efj} + \sum_{(e,f,j)\in\mathcal{A}_i} e\, x^i_{efj}\, p_{ij} \le \mathrm{C}_{\max} \qquad \forall i \in M, \tag{49}$$

$$x^i_{efj} \in \{0,1\} \quad \forall i \in M, (e,f,j) \in \mathcal{A}_i, \tag{50}$$

$$\mathrm{C}_{\max} \ge 0. \tag{51}$$

Constraints (47) force exactly one arc associated with each job to be selected. Constraints (48) ensure flow conservation, or, in other words, that the number of jobs starting with accumulated delay factor k ($k \in \mathcal{N}_i \setminus \{s\}$) is smaller than or equal to the number of jobs ending with accumulated delay factor k. Constraints (49) calculate the makespan by summing, for each machine, the duration of the selected arcs with the time required for the maintenance activities (if we count one maintenance for each selected arc originating from the source node, we should remove one extra maintenance). In Algorithm 2, we provide a procedure to generate all the feasible arcs.

The arc generation algorithm is similar to the block generation algorithm. However, assessing whether an arc is dominated or not is more challenging. There is still an easy criterion which can be derived from Property 1, by checking that the duration of each new arc $(k, d_{ij}k, j)$ is strictly smaller than the ideal processing time of job j plus the duration of a maintenance activity (i.e., $p_{ij}k < p_{ij} + t_i$). If it is not the case, then it is shorter to perform a maintenance activity before processing j at intermediary accumulated delay factor k and arc $(k, d_{ij}k, j)$ does not need to be generated as it is dominated.

Example 1 (*resumed*)**.** The arcs generated by Algorithm 2 for Example 1 are displayed in Figs. 2 and 3. The first set of nodes, $s, B, \ldots, G$, corresponds to each intermediary accumulated factor in $\mathcal{N}_1$ for machine 1, which are $1, 1.1, \ldots, 1.7424$. The second set of nodes, $s', B, \ldots, H'$, corresponds to each intermediary accumulated factor in $\mathcal{N}_2$ for machine 2, which are $1, 1.1, \ldots, 1.5972$.

Algorithm 2 Create arcs.

```
for each i ∈ M do                                              ▷ For each machine i
    A_i ← ∅, N_i ← s                                           ▷ There are no arcs in A_i and only the source node in N_i
    for each j ∈ J ordered by non-increasing p_ij/(d_ij − 1) do ▷ For each job
        N̄_i ← ∅                                                ▷ There are no new nodes
        for each k ∈ N_i do                                    ▷ For each existing node in N_i
            if isNotDominated(k, d_ij k, j) then               ▷ if the arc is not removed by our reduction criterion
                A_i ← (k, d_ij k, j)                           ▷ Add the new arc to A_i
                N̄_i ← d_ij k                                   ▷ Add the new node to N̄_i
            end if
        end for
        for each k̄ ∈ N̄_i do
            N_i ← ∪{k̄},                                        ▷ add the new nodes to N_i
        end for
    end for
end for
```

We now briefly explain how the graph in Fig. 2 was obtained. Before processing the first item, the set of nodes $\mathcal{N}_1$ only contains s, which corresponds to intermediary accumulated factor 1. After processing job 2, Algorithm 2 creates arc $(s, B, 2)$, where B corresponds to intermediary accumulated factor 1.1, and adds B to $\mathcal{N}_1$. After processing job 4, Algorithm 2 creates arcs $(s, C, 4)$ and $(B, E, 4)$, where C corresponds to intermediary accumulated factor 1.2 and E to 1.32 (1.1×1.2), and adds C and E to $\mathcal{N}_1$. After processing job 3, Algorithm 2 creates arcs $(s, B, 3)$, $(B, D, 3)$, $(C, E, 3)$, and $(E, G, 3)$, where D corresponds to intermediary accumulated factor 1.21 (1.1×1.1) and G to 1.452 (1.32×1.1), and adds D and G to $\mathcal{N}_1$. Note that nodes E and B were already contained in $\mathcal{N}_1$ and do not need to be added again. This gives an intuition of the reason why the number of variables grows at a lesser extent in the arc flow-based formulation than it does in the block-based formulation. Finally, after processing job 1, Algorithm 2 creates arcs $(s, C, 1)$, $(B, E, 1)$, $(C, F, 1)$, $(D, G, 1)$, $(E, H, 1)$, and $(G, I, 1)$, where F corresponds to intermediary accumulated factor 1.44 (1.2×1.2), H to 1.584 (1.32×1.2), and I to 1.7424 (1.452×1.2), and adds F, H, and I to $\mathcal{N}_1$.

The block containing jobs 1 and 2 can be obtained on machine 1 by selecting the path containing arcs $(s, B, 2)$ and $(B, E, 1)$, for a duration of $20 \times 1 + 10 \times 1.1 = 31$. The block containing jobs 2, 3, and 4 can be obtained on machine 1 by selecting the path containing arcs $(s, B, 2)$, $(B, E, 4)$, and $(E, G, 3)$, for a duration of $20 \times 1 + 30 \times 1.1 + 10 \times 1.32 = 66.2$. Note that the application of

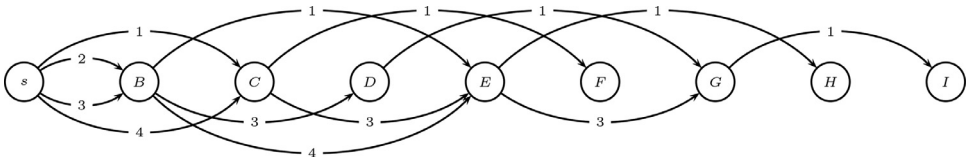

Fig. 2. Arcs generated for Machine 1 in Example 1 without reduction procedure.

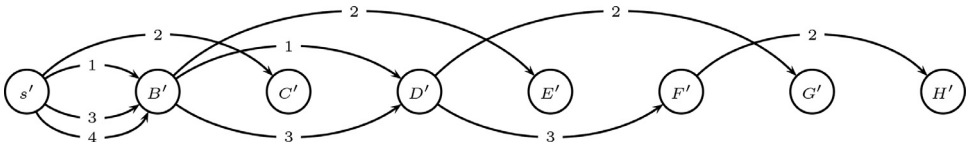

Fig. 3. Arcs generated for Machine 2 in Example 1 without reduction procedure.

the easy reduction procedure would prevent arcs initiating at node C or at subsequent nodes to be generated.

The block containing jobs 1 and 2 can be obtained on machine 2 by selecting the path containing arcs $(s', B', 1)$ and $(B', E', 2)$, for a duration of $20 \times 1 + 10 \times 1.1 = 31$. The block containing jobs 2, 3, and 4 can be obtained on machine 2 by selecting the path containing arcs $(s', B', 4)$, $(B', D', 3)$, and $(D', G', 2)$, for a duration of $30 \times 1 + 10 \times 1.1 + 10 \times 1.21 = 53.1$. Note that applying the easy reduction procedure would not prevent any arc to be generated.

An optimal solution provided by an ILP solver selects arcs $(s, B, 3)$ and $(B, E, 1)$ in machine 1 and arcs $(s', B', 4)$ and $(B', E', 2)$ in machine 2 for a total makespan of 41. There exist other optimal solutions with a makespan equal to 41, such as the one obtained by selecting arcs $(s, B, 3)$ and $(s, C, 1)$ in machine 1 and arcs $(s', B', 4)$ and $(B', E', 2)$ in machine 2.

In the following, we propose a significantly more powerful reduction criterion by assigning an "expiration date" to each node. Before explaining in details the concept of expiration date, we first need to observe that, as $k \geq 1$,

$$(d_{ij} - 1) + k \leq (d_{ij} - 1)k + k = d_{ij}k,$$

or, in other words, that by approximating $d_{ij}k$ with $(d_{ij} - 1) + k$, we obtain a lower bound on the accumulated delay factor. This approximation has a nicer mathematical structure, because it removes the multiplicative aspect of the delay factors. We can now predict that, by processing a set S of jobs whose sum of ideal processing times is l (i.e., $\sum_{j \in S} p_{ij} = l$) after a set of jobs S' whose accumulated delay factor is k (i.e., $\prod_{j \in S'} d_{ij} = k$), then at least $l(k-1)$ additional time can be solely imputed to the machine deterioration caused by the jobs in S'. As a result, if $l(k-1)$ is greater than or equal to the time necessary to perform a maintenance activity t_i (i.e., if $l \geq \frac{t_i}{(k-1)}$), then it is better to schedule a maintenance activity between S' and S than to schedule the two sets of jobs successively one after the other.

The "natural" expiration date of a node on machine i is equal to $\frac{t_i}{(k-1)}$. When we create a new node $d_{ij}k$ through arc $(k, d_{ij}k, j)$, we compare its natural expiration date $\frac{t_i}{(d_{ij}k-1)}$ with the "real" expiration date of its predecessor (i.e., node k) minus the ideal processing time of job j and we keep the minimum among the two values. If node $d_{ij}k$ was already defined, then its expiration date becomes the maximum between its current expiration date and the expiration date it would have had if the node was just created.

Example 2. Let us consider an instance with a single machine having maintenance time $t_1 = 6$, and seven identical jobs having ideal processing time $p_{1j} = 100$ and delay factor $d_{1j} = 1.01$, for $(j = 1, \ldots, 7)$. After creating the initial node s, we process each job in turn and obtain:

- node $n_1 = 1.01$, expiration date $= \frac{6}{1.01-1} = 600$
- node $n_2 = 1.01^2$, expiration date $= \min\{600 - 100, \frac{6}{1.01^2-1}\} \approx 298.507463$
- node $n_3 = 1.01^3$, expiration date $= \min\{298.507463 - 100, \frac{6}{1.01^3-1}\} \approx 198,0132669$
- node $n_4 = 1.01^4$, expiration date $= \min\{198.0132669 - 100, \frac{6}{1.01^4-1}\} \approx 98.0132669$

No further nodes need to be created because the remaining jobs are longer than the expiration date. Note that if we only applied the reduction criterion from Property 1 we would have also created node 1.01^5. An optimal solution schedules the first four jobs on the machine followed by a maintenance activity and then the last three jobs, for a total duration time of 715.0501. The selected arcs are $(s, n_1, 1)$, $(n_1, n_2, 2)$, $(n_2, n_3, 3)$, $(n_3, n_4, 4)$, $(s, n_1, 5)$, $(n_1, n_2, 6)$, $(n_2, n_3, 7)$.

The arc flow-based model involves an exponential number of constraints $O(\sum_{i \in M} |\mathcal{N}_i|)$ and an exponential number of variables $O(\sum_{i \in M} |\mathcal{A}_i|)$. Empirically, these numbers grow at a reasonable pace as the number of jobs increases, because (i) the reduction criterion removes a significant number of arcs and nodes, and (ii) only a few supplementary nodes need to be created when generating the arcs for job j if the arcs of another job j' with similar delay factor (i.e., $d_{ij} = d_{ij'}$) were generated previously. Despite its scalability, the model becomes too large for instances with 10 machines and 200 jobs. In the next section, we introduce a metaheuristic able to find good quality solutions for large size instances.

5. Metaheuristic algorithm

In this section, we describe the ILS algorithm that we developed to look for high-quality solutions in quick time as the ILP formulations can be long to provide a feasible solution, especially for large size instances. The ILS uses a greedy algorithm to build an initial solution, and five local search procedures to improve it.

In the following, we call S a set of jobs, and (S, i) the block obtained on machine i if it had to process the jobs in S in the order determined by Property 2. The greedy procedure uses the notion of "temporary blocks", which are blocks that may be changed later in the algorithm, and the notion of "final blocks", which are blocks that are parts of the initial solution. The greedy procedure works as follows. First, we assign an empty set S_i of jobs to each machine $i \in M$. Then, for each job j, we determine the machine i whose temporary block is the best fitted to perform j, (i.e., the one for which $(S_i \cup \{j\}, i)$ has minimum duration), and add j to S_i. After

that, we try to form a temporary block (S_i, i). If the block satisfies Property 1, we simply continue the job assignment. If, instead, the block does not satisfy the property, then (S_i, i) is split into two smaller blocks, (S'_i, i) and (S''_i, i), separated by a maintenance activity. We consider the first block (S'_i, i) as a final block and update job set S_i so that it now contains only the jobs in S''_i. Once all the jobs have been processed, we transform all the remaining temporary blocks S_i into final blocks.

The initial solution is improved by means of the following five local search procedures:

1. `Swap blocks:` Swap two blocks (S_1, i) and (S_2, i') to obtain (S_1, i') and (S_2, i). Machine i is the machine with the largest workload, and block (S_1, i) is the block of machine i with the largest duration. Block (S_2, i') is chosen randomly from another machine. The procedure performs the first swap that improves the makespan, if any, so working with a *first improvement* policy. It tries to swap block (S_1, i) with at most $\bar{\Gamma}_1$ other blocks (S_2, i').
2. `Move job intra:` Select two blocks (S_1, i) and (S_2, i) on the same machine, and move one job j from the first block to the second one, so as to obtain blocks $(S_1 \setminus \{j\}, i)$ and $(S_2 \cup \{j\}, i)$. The procedure works with first improvement policy and tries to move at most $\bar{\Gamma}_2$ jobs j. At each iteration, machine i, job j, and the two blocks S_1 and S_2 are chosen randomly.
3. `Swap jobs intra:` Select two blocks (S_1, i) and (S_2, i) on the same machine, and swap two jobs $j \in S_1$ and $j' \in S_2$, so as to obtain blocks $(S_1 \cup \{j'\} \setminus \{j\}, i)$ and $(S_2 \cup \{j\} \setminus \{j'\}, i)$. The procedure works with first improvement policy and tries to swap at most $\bar{\Gamma}_3$ jobs j and j'. At each iteration, machine i, jobs j and j', and the two blocks S_1 and S_2 are chosen randomly.
4. `Move block:` Select the block with the largest duration from the machine with the largest workload (S_1, i), and move it to another machine i' so as to create block (S_1, i'). The procedure works in first improvement, and tries to move block S_1 to at most $\bar{\Gamma}_4$ machines. At each iteration, the machine i' with smallest workload (and that was not chosen in a previous iteration) is selected.
5. `Move job inter:` Select two blocks (S_1, i) and (S_2, i') on two distinct machines, and move one job j from the first set to the second one, so as to obtain blocks $(S_1 \setminus \{j\}, i)$ and $(S_2 \cup \{j\}, i')$. The procedure finishes after a given number of iterations $\bar{\Lambda}_5$. The procedure stops as soon as the makespan is improved and tries to move at most $\bar{\Gamma}_5$ jobs j. At each iteration, machines i and i', job j, and the two blocks S_1 and S_2 are chosen randomly.

The local search procedures are called one after the other. Each procedure i $(i = 1, \ldots, 5)$ is called $\bar{\Lambda}_i$ times. Once the local search phase is terminated, we call the following perturbation procedure in order to diversify the solution:

- `Perturb:` Randomly remove 20% of the blocks and assign, one at a time, each of the removed jobs to the block with smallest accumulated delay factor in the machine with smallest workload.

In each of the aforementioned procedures, it is possible that a block has to be split into two smaller blocks if, after the job reordering from Property 2, the algorithm determines that a maintenance activity should be performed within the block because of Property 1. The metaheuristic stops once 20 local search/perturbation cycles were performed without any improvement in the incumbent solution. Preliminary experiments were used to determine the following parameters values: $\bar{\Gamma}_1 = 5$, $\bar{\Gamma}_2 = 10$, $\bar{\Gamma}_3 = 5$, $\bar{\Gamma}_4 = \frac{|M|}{2}$, $\bar{\Gamma}_5 = 10$, $\bar{\Lambda}_1 = 10$, $\bar{\Lambda}_2 = 30$, $\bar{\Lambda}_3 = 30$, $\bar{\Lambda}_4 = 10$, and $\bar{\Lambda}_5 = 30$. This approach is different from the heuristics proposed by Ruiz-Torres et al. (2017), as (1) we use a perturbation component that prevents the objective function to get stuck in a local optimum, (2) our procedure includes a random component to avoid repeating a specific local search move in case the same solution is reached twice during the search procedure, and (3) our moves are suitable for the general case of non-identical machines, while some of the constructive heuristics of Ruiz-Torres et al. (2017) use the fact that the machines are identical.

6. Computational experiments

To test the performance of the methods we proposed for the R|Sdd,mnt|C_{max}, we generated a set of instances by adopting the parameters already used by Ruiz-Torres et al. (2017), namely: number of machines m taking value 2, 5, 10, or 20; ratio of jobs per machine n/m taking value 10, 15, or 20; delay factor d_{ij} uniformly distributed in the range $\{1.01, 1.02, \ldots, 1.06\}$ or in the range $\{1.05, 1.06, \ldots, 1.10\}$; integer duration t_i of a maintenance activity i uniformly distributed in either [1,3] or in [1,9]. For each of the $4 \times 3 \times 2 \times 2 = 48$ configurations, we generated 10 instances resulting in 480 instances in total. In each of the 480 instances, the (integer) ideal processing time of a job p_{ij} was uniformly distributed in range [1,100]. In our experiments, the ideal processing time of a job was the same on every machine (i.e., $p_{ij} = p_{i'j}$, $\forall i, i' \in M, j \in J$), but, unlike Ruiz-Torres et al. (2017), the duration of the maintenance activities and of the delay factors were machine-dependent. This decision was taken in order to make the machines unrelated, while preventing trivial decision making (e.g., if $p_{11} = 1$ and $p_{21} = 100$, then it is unlikely that job 1 gets assigned to machine 2). All our instances can be downloaded at https://github.com/mdelorme2/Scheduling_Sequence_Dependent_Deterioration_Maintenance_Events_Data. All our algorithms were coded in C++. The experiments were run on an Intel Xeon E5-2680W v3, 2.50 GigaHertz with 192 GigaByte of memory, running under Scientific Linux 7.5, and Gurobi 7.5.2 was used to solve the MILP models. Each instance was run using a single core with a time limit of 3600 seconds. We do not compare our models and metaheuristic with other approaches from the literature, because previous works did not consider maintenance as in Santos and Arroyo (2017) or Ding et al. (2019)) or were designed for identical machines as in Ruiz-Torres et al. (2017). We mention, however, that an adaptation of the approach in Ruiz-Torres et al. (2017) developed to handle non-identical machines was investigated in a preliminary version of this work (see Mendes & Iori, 2019), but turned out to be computationally outperformed by a preliminary version of our metaheuristic. We also mention that preliminary computational tests were performed with FICO Xpress Solver to evaluate the MINLP model of Ruiz-Torres et al. (2017), but the results were not satisfying.

6.1. Computational results on small size instances

We first tested each of the four MILP models and the metaheuristic on the 120 randomly generated instances with 2 machines, and we provide detailed results in Tables 3–6. In each table, column "Method" identifies the approach used, column "# opt." gives the number of proven optimal solutions found by the model, column "# UB is opt." gives the number of instances in which the upper bound found by the approach was equal to the optimal solution value (including the cases in which the lower bound did not match the upper bound), column "TT" gives the average execution time (in seconds) required to solve an instance (including those terminated by the time limit), column "TP" indicates the average time (in seconds) required to build the model, columns "LB" and "UB" indicate, respectively, the average lower and upper bounds produced, columns "nb. var.", "nb. cons.", and "nb. nzs" give, respectively, the average number of variables, constraints, and non-zero elements in the MILP models, column "LP" gives the average LP-relaxation of the model, and column

Table 3
Evaluation of the proposed approaches on instances with 2 machines.

method	# opt.	# UB is opt.	TT	TP	LB	UB	nb. var.	nb. cons.	nb. nzs	LP	gap
Slot	0	0	3597.9	0	756.37	795.84	7602	7685	44 448	0	4.4
Event	32	47	2791	0	786.57	791.49	2084	4079	17 582	0	0.6
Block	90	103	1084.6	1.3	791.05	791.09	1 393 975	32	12 154 388	790.9196	0.0
Arc flow w/o exp. date	119	120	109.3	0	791.08	791.09	24 889	11 094	99 493	790.9193	0.0
Arc flow	120	120	60.3	0	791.09	791.09	6044	2865	24 113	790.9193	0.0
ILS	-	0	0.5	-	-	797.75	-	-	-	-	-

Table 4
Evaluation on instances with 2 machines, results grouped by n/m ratio.

n/m	method	# opt.	TT	TP	LB	UB	nb. var.	nb. cons.	nb. nzs
10	Position	0	3597.6	0	499.1	516.4	3109	3162	17 888
	Event	20	2135.3	0	513.2	514.8	901	1742	7322
	Block	40	138.0	0	514.8	514.8	16 415	22	108 220
	Arc flow	40	3.2	0	514.8	514.8	1475	801	5857
	ILS	-	0.4	-	-	520.4	-	-	-
15	Position	0	3596.7	0	791.0	819.4	7069	7152	41 248
	Event	9	2853.9	0	811.5	816.1	1951	3812	16 382
	Block	29	1189.4	0.4	815.7	815.8	461 919	32	3 846 625
	Arc flow	40	48.0	0	815.7	815.7	4958	2556	19 768
	ILS	-	0.5	-	-	822.1	-	-	-
20	Position	0	3599.3	0	979.1	1051.7	12 629	12 742	74 208
	Event	3	3383.9	0	1035.0	1043.6	3401	6682	29 042
	Block	21	1926.4	3.5	1042.7	1042.8	3 703 590	42	32 508 319
	Arc flow	40	129.7	0	1042.8	1042.8	11 699	5236	46 714
	ILS	-	0.7	-	-	1050.8	-	-	-

Table 5
Evaluation on instances with 2 machines, results grouped by delay factor range.

d_{ij} range	method	# opt.	TT	TP	LB	UB	nb. var.	nb. cons.	nb. nzs
[1.01,1.06]	Position	0	3597.4	0	740.49	784.51	7602	7685	44 448
	Event	6	3369.1	0	774.05	780.42	2084	4079	17 582
	Block	35	1797.7	2.6	779.90	779.98	2 744 176	32	24 036 514
	Arc flow	60	110.4	0	779.96	779.96	11 023	5196	44 032
	ILS	-	0.5	-	-	788.71	-	-	-
[1.05,1.10]	Position	0	3598.4	0	772.26	807.16	7602	7685	44 448
	Event	26	2213.0	0	799.09	802.57	2084	4079	17 582
	Block	55	371.5	0	802.20	802.21	43 773	32	272 262
	Arc flow	60	10.2	0	802.21	802.21	1064	533	4194
	ILS	-	0.6	-	-	806.79	-	-	-

Table 6
Evaluation on instances with 2 machines, results grouped by maintenance duration t_i.

t_i range	method	# opt.	TT	TP	LB	UB	nb. var.	nb. cons.	nb. nzs
[1,3]	Position	0	3598.7	0	765.25	786.91	7602	7685	44 448
	Event	21	2416.9	0	782.16	783.60	2084	4079	17 582
	Block	49	915.3	0.1	783.45	783.46	92 834	32	641 860
	Arc flow	60	14.1	0	783.46	783.46	1637	895	6486
	ILS	-	0.6	-	-	787.76	-	-	-
[1,9]	Position	0	3597.1	0	747.50	804.77	7602	7685	44 448
	Event	11	3165.2	0	790.98	799.38	2084	4079	17 582
	Block	41	1253.9	2.6	798.65	798.73	2 695 115	32	23 666 916
	Arc flow	60	106.5	0	798.71	798.71	10 451	4834	41 740
	ILS	-	0.5	-	-	807.74	-	-	-

"gap" gives the average relative gap computed as $100 \times \frac{UB-LB}{UB}$, where LB is the lower bound provided by the model and UB is the best solution found by the model.

We observe that the MILP models that use big-M constraints displayed poor performance. We opted to keep the two models in our computational experiments because (1) they are natural extensions of other models proposed in the literature for the R|Sdd,mnt|C_{max} without maintenance, and (2) they outline how computationally difficult the problem is in practice. The position-based model could not solve a single instance to optimality within an hour (relative gap at 4.4% on average), and the event-based model only solved 32 instances in total – 47

if we count those without proof of optimality (relative gap at 0.6% on average). This can be explained by the very poor quality of their continuous relaxation (having value 0), which is due to the disjunctive constraints. The block-based formulation obtained relatively good results, as it could solve 90 instances – 103 if we count those without proof of optimality (relative gap at 0.005% on average), but the large number of variables it involved (more than a million on average), indicates that it is not a viable option for solving larger instances. The arc flow-based formulation solved all the instances in one minute on average, requiring only 6000 variables on average, and thus appearing as a good candidate for solving larger instances. The continuous relaxation of both the arc flow-based and the block-based formulations are very good, but are almost never equal to an optimal solution. The tiny difference between the continuous relaxation of the two models can be explained by the fact that a block cannot contain twice the same job in the block-based formulation, while this can happen in the arc flow-based formulation. A similar behavior was already noticed for the bin packing problem (see Delorme & Iori, 2020).

For comparison purposes, we also ran the arc flow-based formulation without expiration dates. Even though the results we obtained were competitive (119 instances solved to proven optimality with an average running time of 2 minutes), using expiration dates is strictly better as it reduces by approximately 75% the model size. We thus adopted the expiration dates in all subsequent tests.

The metaheuristic was extremely fast (less than a second on average), and produced good quality solutions as the absolute gap with respect to the optimal solution was less than 7 on average. In the following, we provide more detailed results in which the instances are grouped by ratio of jobs per machine (Table 4), delay factor range (Table 5), and maintenance activity duration range (Table 6).

The results from Table 4 indicate that instances become more difficult to solve as the ratio of jobs per machine n/m increases. This is particularly evident for the block-based formulation, which solved all the instances with $n/m = 10$ but only half of the instances with $n/m = 20$, and for the arc flow-based formulation, which solved all instances with $n/m = 10$ in 3 seconds on average but took more than 2 minutes on average to solve instances with $n/m = 20$. This is not surprising because the number of feasible blocks and the number of relevant intermediary accumulated factors increases with the number of jobs, resulting in an increase in the number of variables for both formulations. The big-M formulations too obtained better results for $n/m = 10$ (absolute gaps – computed as the difference between the upper bound and the lower bound found by the model – around 17 for the position-based and 1.5 for the event-based) than for $n/m = 20$ (absolute gaps around 70 for the position-based and 8.5 for the event-based). The metaheuristic was not particularly impacted by the ratio of jobs per machine, as it obtained an absolute gap with respect to the optimal solution around 6 for $n/m = 10$ and equal to 8 for $n/m = 20$.

The results displayed in Table 5 show that instances are more difficult to solve when the jobs have smaller delay factors d_{ij}. This is true for all models (e.g., the block-based formulation could solve 35 instances with small delay factors, while it could solve 55 instances with large delay factor) and also for the metaheuristic (as the absolute gap with respect to the optimal solution is around 9 for low delay factors while it is around 4.5 for large delay factors). This behaviour is expected for the arc flow-based and the block-based formulations, because the number of jobs per block decreases as the delay factors of the jobs increase, thus resulting in models in fewer variables.

We observe in Table 6 that instances are more difficult to solve when the maintenance activities t_i have longer duration. This is evident for all models (e.g., the arc flow-based formulation could solve all instances with shorter maintenance duration in 14.1 seconds on average, while it took 40 seconds on average to solve instances with longer maintenance duration) and the metaheuristic. This can be explained by the fact that the number of jobs per block increases as the maintenance activity duration increases, resulting in additional variables for the arc flow-based and the block-based formulations.

6.2. Computational results on medium and large size instances

We tested the arc flow-based formulation and the ILS on the remaining instances having 5, 10, and 20 machines. We display the results we obtained in Tables 7–9. In each table, column “group” indicates the criterion used to aggregate the instances into groups, and column “# inst.” indicates the number of aggregated instances per group. The other columns are identical to those displayed in the previous tables. We opted not to report the relative gaps in this section as the quality of the lower bound obtained by the arc flow-based model was inconsistent and could even reach 0 sometimes when the model was not able to solve the linear relaxation in an hour. We focus instead on the quality of the upper bounds provided by the two tested approaches.

We observe in Table 7 that the arc flow-based formulation could only solve one instance to optimality among the 120 instances with 5 machines. However, the average absolute gap between the upper and lower bounds obtained is very small (around 0.2 on average). The metaheuristic was very fast but the solutions it obtained were around 10 units longer on average than those found by the arc flow-based formulation. Interestingly, we only observe a minor impact of the tested parameters on the absolute gaps obtained by the arc flow-based formulation:

- The absolute gap is around 0.2 for each ratio n/m;
- The absolute gap for short delay factors is around 0.3, while it is around 0.1 for long delay factors;
- The absolute gap for short maintenance duration is around 0.1, while it is around 0.3 for long maintenance duration.

Similarly to what was noticed for instances with 2 machines, we observe that the metaheuristic tends to find better quality solutions for instances with long delay factors and short maintenance duration, while it is not particularly impacted by the average ratio n/m.

In Table 8, we focus on the 120 instances with 10 machines. We notice that the arc flow-based formulation could not solve any instance to proven optimality. The average gap between the upper and lower bounds vary significantly depending on the tested parameters: for instances with short maintenance duration, long delay factors, and small n/m ratio, the absolute gap was less than 0.8 on average, while it was significantly higher for the other instances, even reaching 140 on average for instances with high maintenance duration. The behavior of the metaheuristic is relatively similar to what was observed for instances with 2 and 5 machines. We notice here for the first time that the metaheuristic found better solutions (upper bound equal to 790.22 on average) than the arc flow-based formulation (upper bound equal to 804.63 on average). By studying the distribution of parameter Δ, which we define as the difference between the upper bound obtained by the ILS and the upper bound obtained by the arc flow-based model (i.e., $\Delta = UB_{ILS} - UB_{AF}$), we report that Δ is equal to -14.41 on average, but its median is equal to 8.87. This indicates that $UB_{AF} < UB_{ILS}$ for a majority of instances, but in the rare cases in which $UB_{AF} > UB_{ILS}$, the difference is significant (up to approximatively 930). Further analysis outlined that the upper bound obtained by the arc flow-based model is particularly bad for instances in which most of the computation time is spent on computing the lower bound.

Table 7
Evaluation on instances with 5 machines.

group	# inst.	method	# opt.	TT	TP	LB	UB	nb. var.	nb. cons.	nb. nzs
$n/m = 10$	40	Arc flow	1	3600	0.1	515.99	516.24	71 613	29 763	286 201
		ILS	-	2.1	-	-	524.16	-	-	-
$n/m = 15$	40	Arc flow	0	3600	0.3	777.55	777.75	224 008	84 430	895 660
		ILS	-	3.0	-	-	787.20	-	-	-
$n/m = 20$	40	Arc flow	0	3600	0.6	1038.26	1038.46	478 514	164 100	1 913 558
		ILS	-	3.8	-	-	1049.72	-	-	-
$d_{ij} \in [1, 6]$	60	Arc flow	0	3600	0.6	766.14	766.43	491 081	176 263	1 963 948
		ILS	-	2.6	-	-	778.50	-	-	-
$d_{ij} \in [5, 10]$	60	Arc flow	1	3600	0	788.40	788.53	25 010	9266	99 664
		ILS	-	3.3	-	-	795.55	-	-	-
$t_i \in [1, 3]$	60	Arc flow	0	3600	0	772.77	772.91	37 908	16 377	151 257
		ILS	-	3.3	-	-	779.42	-	-	-
$t_i \in [1, 9]$	60	Arc flow	1	3600	0.6	781.77	782.06	478 182	16 9152	1 912 356
		ILS	-	2.7	-	-	794.63	-	-	-
Overall	120	Arc flow	1	3600	0.3	777.27	777.48	258 045	92 764	1 031 806
		ILS	-	3.0	-	-	787.03	-	-	-

Table 8
Evaluation on instances with 10 machines.

group	# inst.	method	# opt.	TT	TP	LB	UB	nb. var.	nb. cons.	nb. nzs
$n/m = 10$	40	Arc flow	0	3600	1.2	516.75	517.51	904 542	314 498	3 617 176
		ILS	-	8.2	-	-	541.67	-	-	-
$n/m = 15$	40	Arc flow	0	3600	2.4	748.31	774.14	2 033 046	657 109	8 130 689
		ILS	-	12.1	-	-	778.64	-	-	-
$n/m = 20$	40	Arc flow	0	3600	6.5	938.12	1122.26	5 040 395	1 450 953	20 159 585
		ILS	-	17.0	-	-	1050.35	-	-	-
$d_{ij} \in [1, 6]$	60	Arc flow	0	3600	6.6	686.32	826.40	5 151 331	1 566 014	20 603 831
		ILS	-	11.1	-	-	778.61	-	-	-
$d_{ij} \in [5, 10]$	60	Arc flow	0	3600	0.1	782.47	782.87	167 324	49 026	667 801
		ILS	-	13.8	-	-	801.83	-	-	-
$t_i \in [1, 3]$	60	Arc flow	0	3600	0.4	769.61	770.04	340 493	127 799	1 360 478
		ILS	-	13.4	-	-	777.77	-	-	-
$t_i \in [1, 9]$	60	Arc flow	0	3600	6.4	699.17	839.23	4 978 162	1 487 241	19 911 155
		ILS	-	11.5	-	-	802.67	-	-	-
Overall	120	Arc flow	0	3600	3.4	734.39	804.63	2 659 328	807 520	10 635 816
		ILS	-	12.5	-	-	790.22	-	-	-

Table 9
Evaluation on instances with 20 machines.

group	# inst.	method	# opt.	TT	TP	LB	UB	nb. var.	nb. cons.	nb. nzs
$n/m = 10$	40	Arc flow	0	3600	12.9	428.16	630.39	8 771 690	2 584 382	35 082 775
		ILS	-	35.2	-	-	535.29	-	-	-
$n/m = 15$	40	Arc flow	0	3600	43.9	587.08	991.11	28 817 601	7 058 348	115 264 422
		ILS	-	53.2	-	-	789.44	-	-	-
$n/m = 20$	40	Arc flow	0	3600	94.5	655.55	1283.72	38 341 270	8 760 478	153 577 723
		ILS	-	83.1	-	-	1062.14	-	-	-
$d_{ij} \in [1, 6]$	60	Arc flow	0	3600	99.9	328.39	1150.58	49 490 269	12 000 939	198 102 178
		ILS	-	50.5	-	-	787.31	-	-	-
$d_{ij} \in [5, 10]$	60	Arc flow	0	3600	1	785.47	786.23	1 130 105	267 866	4 514 435
		ILS	-	63.9	-	-	803.93	-	-	-
$t_i \in [1, 3]$	60	Arc flow	0	3600	4.2	693.53	851.68	3 555 787	1 134 663	14 217 164
		ILS	-	59.9	-	-	795.08	-	-	-
$t_i \in [1, 9]$	60	Arc flow	0	3600	96.7	420.33	1085.14	47 064 587	11 134 141	188 399 449
		ILS	-	54.4	-	-	796.16	-	-	-
Overall	120	Arc flow	0	3600	50.5	556.93	968.41	25 310 187	6 134 402	101 308 307
		ILS	-	57.2	-	-	795.62	-	-	-

Unsurprisingly, we observe in Table 9 that for instances with 20 machines the arc flow-based formulation also obtains better results for instances with short maintenance duration, long delay factors, and small ratio n/m. The behavior of the metaheuristic is very satisfactory, as it now outperforms the average solution provided by the arc flow-based formulation on all types of instances, with a single exception on those with large delay factors. The average computational effort of the ILS is around one minute, and ranges between 35 seconds for $n/m = 10$, to 83 seconds for $n/m = 20$, on average.

7. Conclusion

We studied the problem of processing a set of jobs on a set of unrelated parallel machines by considering sequence-dependent deterioration and the option of restoring a machine to its full operational speed by performing a maintenance activity. We reviewed an integer non-linear programming formulation from the literature, and introduced four novel mixed integer linear programming formulations. We derived two properties from the literature that allowed us to improve the performance of the models. In addition, we developed a new metaheuristic approach, based on the concept of iterated local search, to provide good quality solutions for large size instances.

We tested all our approaches with an extensive set of computational experiments. Among the mathematical models, we observed that the arc flow-based formulation was the one providing the best results on average: it could solve to proven optimality all instances with 2 machines, and obtained good quality solutions for most instances with 5, 10, and 20 machines. We also noticed that, due to the large number of variables it requires, the arc flow-based formulation could have a large optimality gap, and thus be outperformed by the metaheuristic in terms of running time and solution value. This happened mostly on instances with 20 machines. We also outlined specific parameters that made the instances easier to solve by our approaches (namely, small n/m ratio, short maintenance time, and long delay factors).

Interesting future research directions concern the development of a branch-and-price algorithm for the block-based formulation, the search for a tighter arcflow LP-relaxation (e.g., by avoiding the duplication of a job in the same block) and the investigation of maintenance scheduling problems with sequence-dependent deterioration with alternative objective functions, such as weighted completion time, weighted earliness or weighted tardiness functions.

Acknowledgments

We thank three anonymous referees for their helpful comments. We acknowledge financial support from University of Modena and Reggio Emilia under grant FAR 2018 and from the Engineering and Physical Science Research Council through grant EP/P029825/1.

References

Alidaee, B., & Womer, N. (1999). Scheduling with time dependent processing times: Review and extensions. *Journal of the Operational Research Society, 50*, 711–720. https://doi.org/10.1057/palgrave.jors.2600740.

Araújo, O., Dhein, G., & Fampa, M. (2017). Minimizing the makespan on parallel machines with sequence dependent deteriorating effects. In *Xlix sbpo, smposio brasileiro de pesquisa operacional*.

Browne, S., & Yechiali, U. (1990). Scheduling deteriorating jobs on a single processor. *Operations Research, 38*(3), 495–498. https://doi.org/10.1287/opre.38.3.495.

Valério de Carvalho, J. (1999). Exact solution of bin-packing problems using column generation and branch-and-bound. *Annals of Operations Research, 86*, 629–659. https://doi.org/10.1023/A:1018952112615.

Cheng, T., & Ding, Q. (2001). Single machine scheduling with step-deteriorating processing times. *European Journal of Operational Research, 134*(3), 623–630. https://doi.org/10.1016/S0377-2217(00)00284-8.

Cheng, T., Ding, Q., & Lin, B. (2004). A concise survey of scheduling with time-dependent processing times. *European Journal of Operational Research, 152*(1), 1–13. https://doi.org/10.1016/S0377-2217(02)00909-8.

Clautiaux, F., Hanafi, S., Macedo, R., Voge, M.-E., & Alves, C. (2017). Iterative aggregation and disaggregation algorithm for pseudo-polynomial network flow models with side constraints. *European Journal of Operational Research, 258*(2), 467–477. https://doi.org/10.1016/j.ejor.2016.09.051.

Clautiaux, F., Sadykov, R., Vanderbeck, F., & Viaud, Q. (2018). Combining dynamic programming with filtering to solve a four-stage two-dimensional guillotine-cut bounded knapsack problem. *Discrete Optimization, 29*, 18–44. https://doi.org/10.1016/j.disopt.2018.02.003.

Dell'Amico, M., Delorme, M., Iori, M., & Martello, S. (2019). Mathematical models and decomposition methods for the multiple knapsack problem. *European Journal of Operational Research, 274*(3), 886–899. https://doi.org/10.1016/j.ejor.2018.10.043.

Delorme, M., & Iori, M. (2020). Enhanced pseudo-polynomial formulations for bin packing and cutting stock problems. *INFORMS Journal on Computing, 32*(1), 101–119. https://doi.org/10.1287/ijoc.2018.0880.

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. https://doi.org/10.1016/j.cor.2018.10.016.

Gawiejnowicz, S. (2020). A review of four decades of time-dependent scheduling: Main results, new topics, and open problems. *Journal of Scheduling, 23*, 3–47. https://doi.org/10.1007/s10951-019-00630-w.

Gilmore, P., & Gomory, R. (1961). A linear programming approach to the cutting-stock problem. *Operations Research, 9*(6), 849–859. https://doi.org/10.1287/opre.9.6.849.

Gilmore, P., & Gomory, R. (1963). A linear programming approach to the cutting-stock problem. *Operations Research, 11*(6), 863–888.

Graham, R., Lawler, E., Lenstra, J., & Kan, A. R. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. In P. Hammer, E. Johnson, & B. Korte (Eds.), *Discrete optimization ii*. In *Annals of Discrete Mathematics: 5* (pp. 287–326). Elsevier. https://doi.org/10.1016/S0167-5060(08)70356-X.

Gupta, J., & Gupta, S. (1988). Single facility scheduling with nonlinear processing times. *Computers & Industrial Engineering, 14*(4), 387–393. https://doi.org/10.1016/0360-8352(88)90041-1.

Ji, M., & Cheng, T. (2008). Parallel-machine scheduling with simple linear deterioration to minimize total completion time. *European Journal of Operational Research, 188*(2), 342–347. https://doi.org/10.1016/j.ejor.2007.04.050.

Kramer, A., Dell'Amico, M., & Iori, M. (2019). Enhanced arc-flow formulations to minimize weighted completion time on identical parallel machines. *European Journal of Operational Research, 275*(1), 67–79. https://doi.org/10.1016/j.ejor.2018.11.039.

Kubiak, W., & van de Velde, S. (1998). Scheduling deteriorating jobs to minimize makespan. *Naval Research Logistics, 45*(5), 511–523. https://doi.org/10.1002/(SICI)1520-6750(199808)45:5<511::AID-NAV5)3.0.CO;2-6.

Kunnathur, A., & Gupta, S. (1990). Minimizing the makespan with late start penalties added to processing times in a single facility scheduling problem. *European Journal of Operational Research, 47*(1), 56–64. https://doi.org/10.1016/0377-2217(90)90089-T.

Kuo, W.-H., & Yang, D.-L. (2008). Minimizing the makespan in a single-machine scheduling problem with the cyclic process of an aging effect. *Journal of the Operational Research Society, 59*(3), 416–420. https://doi.org/10.1057/palgrave.jors.2602363.

Lalla-Ruiz, E., & Voß, S. (2016). Modeling the parallel machine scheduling problem with step deteriorating jobs. *European Journal of Operational Research, 255*(1), 21–33. https://doi.org/10.1016/j.ejor.2016.04.010.

Lee, C.-Y., & Leon, V. (2001). Machine scheduling with a rate-modifying activity. *European Journal of Operational Research, 128*(1), 119–128. https://doi.org/10.1016/S0377-2217(99)00066-1.

Leung, J.-T., Ng, C., & Cheng, T. (2008). Minimizing sum of completion times for batch scheduling of jobs with deteriorating processing times. *European Journal of Operational Research, 187*(3), 1090–1099. https://doi.org/10.1016/j.ejor.2006.03.067.

Lu, S., Liu, X., Pei, J., Thai, M., & Pardalas, P. (2018). A hybrid ABC-TS algorithm for the unrelated parallel-batching machines scheduling problem with deteriorating jobs and maintenance activity. *Applied Soft Computing, 66*, 168–182. https://doi.org/10.1016/j.asoc.2018.02.018.

Mendes, N., & Iori, M. (2019). A mathematical model and metaheuristic for a job and maintenance machine scheduling problem with sequence dependent deterioration. In *Xlix sbpo, s-mposio brasileiro de pesquisa operacional: 2*.

Mosheiov, G. (1991). V-shaped policies for scheduling deteriorating jobs. *Operations Research, 39*(6), 979–991. https://doi.org/10.1287/opre.39.6.979.

Mosheiov, G. (1994). Scheduling jobs under simple linear deterioration. *Computers & Operations Research, 21*(6), 653–659. https://doi.org/10.1016/0305-0548(94)90080-9.

Mosheiov, G. (1995). Scheduling jobs with step-deterioration; minimizing makespan on a single- and multi-machine. *Computers & Industrial Engineering, 28*(4), 869–879. https://doi.org/10.1016/0360-8352(95)00006-M.

Mosheiov, G. (1996). Λ-shaped policies to schedule deteriorating jobs. *Journal of the Operational Research Society, 47*(9), 1184–1191. https://doi.org/10.1057/jors.1996.146.

Mosheiov, G. (1998). Multi-machine scheduling with linear deterioration. *INFOR: Information Systems and Operational Research, 36*(4), 205–214. https://doi.org/10.1080/03155986.1998.11732359.

Mosheiov, G., & Sidney, J. (2003). New results on sequencing with rate modification. *INFOR: Information Systems and Operational Research, 41*(2), 155–163. https://doi.org/10.1080/03155986.2003.11732673.

Nesello, V., Delorme, M., Iori, M., & Subramanian, A. (2018a). Mathematical models and decomposition algorithms for makespan minimization in plastic rolls production. *Journal of the Operational Research Society, 69*, 326–339. https://doi.org/10.1057/s41274-017-0221-8.

Nesello, V., Subramanian, A., Battarra, M., & Laporte, G. (2018b). Exact solution of the single-machine scheduling problem with periodic maintenances and sequence-dependent setup times. *European Journal of Operational Research, 266*(2), 498–507. https://doi.org/10.1016/j.ejor.2017.10.020.

Pacheco, J., Àngel Bello, F., & Álvarez, A. (2013). A multi-start tabu search method for a single-machine scheduling problem with periodic maintenance and sequence-dependent set-up times. *Journal of Scheduling, 16*, 661–673.

Pessoa, A., Uchoa, E., Poggi, M., & Rodrigues, R. (2010). Exact algorithm over an arc–time-indexed formulation for parallel machine scheduling problems. *Mathematical Programming Computation, 2*, 259–290.

Pinedo, M. (2016). *Scheduling: Theory, algorithms and systems development*. New York: Springer. Fifth Edition

Ramos, B., Alves, C., & Valério de Carvalho, J. (2020 (forthcoming)). An arc flow formulation to the multitrip production, inventory, distribution, and routing problem with time windows. *International Transactions in Operational Research*. https://doi.org/10.1111/itor.12765.

Ruiz-Torres, A., Paletta, G., & M'Hallah, R. (2017). Makespan minimisation with sequence-dependent machine deterioration and maintenance events. *International Journal of Production Research, 55*(2), 462–479. https://doi.org/10.1080/00207543.2016.1187776.

Ruiz-Torres, A., Paletta, G., & Pérez, E. (2013). Parallel machine scheduling to minimize the makespan with sequence dependent deteriorating effects. *Computers & Operations Research, 40*(8), 2051–2061. https://doi.org/10.1016/j.cor.2013.02.018.

Santos, V., & Arroyo, J. (2017). Iterated greedy with random variable neighborhood descent for scheduling jobs on parallel machines with deterioration effect. *Electronic Notes in Discrete Mathematics, 58*, 55–62. https://doi.org/10.1016/j.endm.2017.03.008.

Strusevich, V., & Rustogi, K. (2017). *Scheduling with time-changing effects and rate–modifying activities*. Berlin: Springer.

Wang, J.-B., & Li, L. (2017). Machine scheduling with deteriorating jobs and modifying maintenance activities. *The Computer Journal, 61*(1), 47–53. https://doi.org/10.1093/comjnl/bxx032.

Yang, D.-L., Cheng, T., Yang, S.-J., & Hsu, C.-J. (2012). Unrelated parallel-machine scheduling with aging effects and multi-maintenance activities. *Computers & Operations Research, 39*(7), 1458–1464. https://doi.org/10.1016/j.cor.2011.08.017.

Yang, S.-J. (2011). Parallel machines scheduling with simultaneous considerations of position-dependent deterioration effects and maintenance activities. *Journal of the Chinese Institute of Industrial Engineers, 28*(4), 270–280. https://doi.org/10.1080/10170669.2011.573006.

Zhao, C., & Tang, H. (2010). Single machine scheduling with general job-dependent aging effect and maintenance activities to minimize makespan. *Applied Mathematical Modelling, 34*(3), 837–841. https://doi.org/10.1016/j.apm.2009.07.002.