LI Simpósio Brasileiro de Pesquisa Operacional
Limeira–SP, 02 a 06 de setembro de 2019

A mathematical model and a metaheuristic for a job and maintenance machine scheduling problem with sequence dependent deterioration

Nilson Felipe Matos Mendes
DISMI, Università degli Studi di Modena e Reggio Emilia
Via Amendola 2, Pad. Morselli, 42122 Reggio Emilia, Italia
nilson.mendes@unimore.it

Manuel Iori
DISMI, Università degli Studi di Modena e Reggio Emilia
Via Amendola 2, Pad. Morselli, 42122 Reggio Emilia, Italia
manuel.iori@unimore.it

RESUMO

Máquinas que possuem alto nível de ocupação comumente apresentam problemas de desgaste que podem impactar severamente o seu desempenho. Nestes casos, atividades de manutenção podem ser planejadas para restaurar a produtividade máxima. Neste trabalho tratamos um problema de escalonamento de tarefas e manutenções em um conjunto de máquinas paralelas não correlatas em que se considera que o tempo de processamento cresce de acordo com um fator de deterioração que depende da máquina e da própria tarefa. O objetivo é definir um conjunto de escalonamentos que minimizem o makespan. É apresentada uma versão linear de um modelo disponível na literatura, bem como uma heurística ILS. Extensivos testes computacionais são usados para verificar a eficiência dos métodos propostos, quando comparados com os resultados recentes da literatura.

PALAVRAS CHAVE. Job Scheduling. Modelagem matemática. Meta-heuristica.

Tópicos (Escalonamento de tarefas, Modelagem matemática, Linearização, Meta-heurística, ILS)

ABSTRACT

Machines that have a high occupation level commonly present deterioration issues that can severely impact on their performance. In such cases, maintenance activities can be scheduled to restore full productivity. In this work, we deal with the problem of scheduling jobs and maintenance activities on a set of unrelated parallel machines, by considering that the processing time of each job increases according to a deterioration factor that depends both on the machine and on the job itself. The aim is to define the set of schedules that minimize the makespan. We present a linear version of a mathematical model available in the literature, as well as an iterated local search metaheuristic. Extensive computational tests are used to asses the efficiency of the methods and compare them with the recent literature.

KEYWORDS. Job Scheduling. Mathematical modeling. Meta-heuristic.

Paper topics (Job scheduling, Deterioration, Mathematical model, Linearization, Iterated Local Search)

1. Introduction

Fatigue and deterioration might severely affect the performance of persons and machines that execute activities over time. This can cause increases in the processing time, in the errors incidence and in the amount of waste. As a consequence, in many applications it is convenient to include work stops or maintenance events on the scheduling of activities, in order to recover the full performance of the executors and provide an overall best functioning of the system.

In this paper, we approach with a machine job scheduling problem that models environments where deterioration is a relevant factor in a short time horizon, like medical services, student examinations and computer processing tasks. In those scenarios, we might experience a significant increase in the processing time required for a job after the execution of one or more previous jobs due to deterioration factors, and short maintenance procedures (such as a repair, stop, cooling, warming, etc...) might be invoked to optimize a given system performance metric. More in detail, we deal with the problem of scheduling jobs and maintenance activities on a set of unrelated parallel machines, by considering that the processing time of each job increases according to a deterioration factor that depends both on the machine and on the job itself. The aim is to define the set of schedules that minimize the makespan, that is, the last completion time of a job.

We have two main contributions. The first one is a linearization of the mixed integer nonlinear programming (MINLP) model originally proposed by Ruiz-Torres et al. [2017] to formally describe the problem that we tackle in our paper. Our resulting mixed integer linear programming (MILP) model can be solved on a standard MILP solver, and obtains interesting results on small-size instances. Our second contribution is to provide a metaheuristic procedure based on the concept of Iterated Local Search (ILS), that allows to quickly solve instances from medium and large size near to optimality considering different deterioration rates and different processing and maintenance times.

The remainder of the paper is organized as follows. In the next section, we present a short review of previous literature that deals with scheduling problems with deterioration and maintenance. In Section 3, we formally describe the problem, and then in Section 4, the model by Ruiz-Torres et al. [2017] and our linearized version are introduced. In Section 5, we detail the proposed ILS meatheuristic. In Section 6, we describe the outcome of extensive computational experiments that we executed to assess the quality of the proposed approaches, also comparing with the most performing heuristic in the literature. Finally, in Section 7, we draw some conclusions and discuss promising future research directions.

2. Literature review

Scheduling problems with deterioration issues have been investigated since the late 80's. The first study, to the best of our knowledge, was formulated by Gupta and Gupta [1988] and focused on the case in which job processing times depend on their starting times. This seminal study was followed in the next years by Browne and Yechiali [1990], Mosheiov [1991] and Mosheiov [1994] that used similar ideas to evaluate deterioration in a number of scheduling problems.

The published studies can be divided into two main groups according to how deterioration is estimated. The first group estimates the deterioration on the basis of the job starting time, supposing that a machine is deteriorated with the same rate by any processed job and thus independently of the nature of the jobs. The second group estimates the deterioration on the basis of the set of jobs previously processed by a machine, thus considering the eventual different deterioration rates due to the hardness of the different jobs and obtaining a sequence-dependent function.

Among the first group of works, in which the deterioration is dependent on starting time, we can highlight Mosheiov [1998], who proved that makespan minimization on parallel machines

is a strongly NP-hard problem even for a two machine case, and Ji and Cheng [2008], who created a polynomial-time approximation scheme for the case with a fixed number of machines. In Cheng and Ding [2001], Leung et al. [2008] and Lalla-Ruiz and Voß [2016] the concept of step-deterioration is used to describe situations where the processing time changes if and only if the job is processed after a certain threshold time.

The first paper dealing with a sequence-dependent deterioration is, to the best of our knowledge, that of Ruiz-Torres et al. [2013]. In this paper, a non-linear mixed integer programming model is presented with the aim of formally describing the problem. The same problem was the subject of the study in Santos and Arroyo [2015], where an ILS and an ILS combined with a Random Variable Neighborhood Descent algorithm were proposed and de Araújo et al. [2017], that created a linear model to the problem and also obtained important theoretical results.

Maintenance activities were introduced by Yang [2011] and Yang et al. [2012] in a parallel machine scheduling with position-dependent deterioration, where the time is not important to define the deterioration level but still the jobs are considered equivalent one another in terms of deterioration potential. We also would like to highlight Huang and Wang [2015], who still use the position-dependent deterioration concept, but without maintenance activities.

The first work in which sequence-dependent deterioration and maintenance activities have been studied in conjunction has been published by Ruiz-Torres et al. [2017]. In their paper, the processing time of a job depends on the sequence of activities performed before the execution of the job itself. After this work, still to the best of our knowledge, only Ding et al. [2019] focused on the same subject, by slightly changing the problem definition so as to make the job deterioration influence also its own processing time.

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 set $M = \{1, 2, \ldots, m\}$ of unrelated parallel machines. Each machine suffers a deterioration of its processing speed computed as follows. Let p_{ij} be the processing time of job $j \in J$ on machine $i \in M$ when the machine is fully operative, that is, at the beginning of activities or right after a maintenance event has occurred. Let t_i be the duration of a maintenance event that returns machine i to its fully operative state. Maintenance activities can only be started when the processing of a job has been completed, so no preemption is allowed. There is no limit on the number of maintenance activities to be performed on a machine, nor on the number of maintenance activities to be performed in parallel on all machines. Each machine either processes a single job or undergoes a maintenance activity in any moment of the working horizon.

Each job must be assigned to a machine so that the makespan, that is, the completion time of the latest job, is minimized. To compute the completion times, we need to evaluate the time spent by each machine on processing jobs and on maintenance activities. Suppose job j is assigned to machine i, and let $\delta_{ij} \geq 1$ be the accumulated delay factor caused by deterioration on machine $i \in M$ just before the beginning of the activity on job $j \in J$, which depends on the previous jobs processed on i because of the sequence dependent deterioration. Let also $d_{ij} \geq 1$ be the additional delay factor caused by deterioration on the machine $i \in M$ after the processing of job $j \in J$. Then we can state that:

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

By summing the actual jobs and maintenances processing times we can easily compute all completion times and thus the makespan. Note indeed that idle times on machines do not occur in any optimal schedule, because maintenance activities are independent one another and unlimited in number and because jobs are independent one another and all available at time 0.

According to the three-field notation by Graham et al. [1979], the 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 NP-hard because generalizes the well-known R||C_{max}, already proven to be NP-hard in Pinedo [2016].

4. Mathematical models

In this section, we present the mathematical model originally proposed by [Ruiz-Torres et al., 2017] and then the modifications that we implemented to create its linearized version. We also include some constraints to avoid some infeasible solutions and make the model stronger.

The model is based on machine slots that can be filled with a job or a maintenance. These slots are defined by a set $H = \{1, 2, \ldots, |H|\}$ and are used to define the position of an activity on a machine schedule and have no fixed duration. Let x_{ijh} be a binary variable taking the value 1 if $j \in J$ is executed in slot $h \in H$ of machine $i \in M$, 0 otherwise. Similarly, let s_{ih} be a binary variable taking the value 1 if a maintenance is executed in slot $h \in H$ of machine $i \in M$, 0 otherwise.

According to the activity performed in a machine slot, we can define the resulting deterioration in the next machine slot. This information is stored in the continuous variable q_{ih} that reports the current performance ratio of machine i in slot h. The variable satisfies $0 \leq q_{ih} \leq 1$, so when $q_{ih} = 1$ the machine is fully operative and there is no increase in the processing time of the job assigned to slot h, and when $q_{ih} = 0$ the machine is totally deteriorated and cannot process any additional job. By using these variables and an additional continuous variable C_{max} simply indicating the value of the makespan, the R|Sdd,mnt|C_{max} can be represented as the following MINLP model:

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

subject to

$$\sum_{j \in J} x_{ijh} + s_{ih} \leq 1 \qquad \forall i \in M, h \in H \tag{2}$$

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

$$\sum_{j \in J} \sum_{h \in H} \frac{p_{ij}}{q_{ih}} x_{ijh} + \sum_{h \in H} t_i s_{ih} - C_{max} \leq 0 \qquad \forall i \in M \tag{4}$$

$$q_{ih} - \sum_{j \in J} (1 - d_{ij}) q_{i,h-1} x_{ij,h-1} - s_{i,h-1} = 0 \qquad \forall i \in M, h \in H \backslash \{1\} \tag{5}$$

$$q_{i1} = 1 \qquad \forall i \in M \tag{6}$$

$$x_{ijh} \in \{0, 1\} \qquad \forall i \in M, j \in J, h \in H \tag{7}$$

$$s_{ih} \in \{0, 1\} \qquad \forall i \in M, h \in H \tag{8}$$

$$q_{ih} \geq 0 \qquad \forall i \in M, h \in H \tag{9}$$

The objective function (1) imposes the minimization of the makespan. Constraints (2) state that each slot can accommodate either a single job, either a maintenance activity, or remain empty.

Constraints (3) impose each job to be assigned to a slot of a machine. Constraints (4) compute the maskespan based on the finishing time of operations on each machine. Constraints (5) compute the deterioration on each machine slot through an inductive process that starts from the previous slot on the same machine. Constraints (6) initialize the q variables to 1 for the first slot of each machine, thus imposing the machines to be fully operative at the beginning of activities (these constraints could be removed without losing optimality, as there would be no gain in reducing the values of q_{i1}, but are kept for the sake of clarity of the model). Constraints (7), (8) and (9) define the domains of the variables. It can be noticed that constraints (4) and (5) are non-linear. We remove these non-linearities and obtain a MILP model as follows.

The new formulation proposed uses a strategy similar to the presented in de Araújo et al. [2017], creating a new continuous variable q'_{ih} to represent the processing time delay factor of machine i on slot h after a sequence of activities composed by jobs and maintenances. This variable is used to substitute the information on deterioration previously stored by using the performance ratio variable q_{ih}. The lower bound of variable q'_{ih} is set to 1, representing a no delay due to deterioration on the machine, and the upper bound is set to the product of all possible deteriorations on the machine. By using this variable, we can compute the actual processing time of job j on machine i in slot h as $p_{ij}q'_{ih}$, thus replacing the fraction in (4) by a multiplication.

Formally, we make use of a continuous variable a_{ijh}, that is forced to take value (at least) equal to $p_{ij}q'_{ih}$ when job j is assigned to slot h on machine i, but can take value 0 otherwise. Another modification that we use is to replace the original parameter d_{ij} by a new parameter $d'_{ij} = 1/d_{ij}$ that represents the delay caused by deterioration when processing j on i. The resulting MILP model is the following:

$$\min \mathrm{C}_{\max} \tag{10}$$

subject to

$$\sum_{j\in J} x_{ijh} + s_{ih} \leq 1 \qquad \forall i \in M, h \in H \tag{11}$$

$$\sum_{i\in M}\sum_{h\in H} x_{ijh} = 1 \qquad \forall j \in J \tag{12}$$

$$a_{ijh} \leq M^a_i x_{ijh} \qquad \forall i \in M, j \in J, h \in H \tag{13}$$

$$a_{ijh} \geq p_{ij}q'_{ih} - M^a_i(1 - x_{ijh}) \qquad \forall i \in M, j \in J, h \in H \tag{14}$$

$$\sum_{j\in J}\sum_{h\in H} a_{ijh} + \sum_{h\in H} t_i s_{ih} - \mathrm{C}_{\max} \leq 0 \qquad \forall i \in M \tag{15}$$

$$q'_{ih} \geq d'_{ij}q'_{i,h-1} - M^b_i(s_{i,h-1} + 1 - x_{ij,h-1}) \qquad i \in M, j \in J, h \in H\backslash\{1\} \tag{16}$$

$$q'_{i1} = 1 \qquad \forall i \in M \tag{17}$$

$$x_{ijh} \in \{0, 1\} \qquad \forall i \in M, j \in J, h \in H \tag{18}$$

$$s_{ih} \in \{0, 1\} \qquad \forall i \in M, h \in H \tag{19}$$

$$a_{ijh} \geq 0 \qquad \forall i \in M, j \in J, h \in H \tag{20}$$

$$q'_{ih} \geq 1 \qquad \forall i \in M, h \in H \tag{21}$$

In this new model, objective function (10) and constraints (11) and (12) are identical to those reported in the original MINLP model. Constraints (13) and (14) are used to define the actual processing time of job j on slot h of machine m, with $M^a_i = \sum_{j\in J} p_{ij}$ be a large constant defined, for any machine $i \in M$, so as to fix the value of a_{ijh} whenever $x_{ijh} = 1$. Constraints (15) are

the linear version of (4), and are used to calculate the makespan. The linear representation of (5) is obtained by constraints (16), that adopt the large constant $M_i^b = \prod_{j \in J} d_{ij}$, for any machine $i \in M$, to evaluate the delay in the processing time. Constraints (17) are used to impose maximum speed for all machines in their first slots, similarly to what previously imposed with (6). Constraints (18)–(21) define the domains of the variables.

4.1. Further considerations on the slot-based formulation

Slot-based models typically involve a large number of variables and constraints, so it is convenient to try to limit their size. To this aim, we reduce the space of solutions of model (10)–(21) by using three additional constraints:

$$s_{i,h-1} + s_{ih} \leq 1 \qquad \forall i \in M, h \in H \backslash \{1\} \tag{22}$$

$$s_{i,h-1} + \sum_{j \in J} x_{ij,h-1} \geq s_{ih} + \sum_{j \in J} x_{ijh} \qquad \forall i \in M, h \in H \backslash \{1\} \tag{23}$$

$$x_{ijh} \leq \sum_{l \in N} x_{li,h-1} - s_{i,h-1} \qquad \forall i \in M, j \in J, h \in H \backslash \{1\} \tag{24}$$

Constraints (22) state that is not permitted to perform two maintenances in two consecutive slots. Constraints (23) and (24) are complementary and state that empty slots in the middle of a scheduling are not allowed. The resulting model (10)–(24) has been used in our computational experiments in Section 6.

One of the main concerns in slot-based formulations is to define the number of slots available for each machine in order to allow the model to reach optimality but at the same time be as small as possible. As any job except the last in a machine could be followed by a maintenance activity, a pessimistic value for $|H|$ could be easily fixed $2n$. This value is reasonable only for instances where deterioration is huge with respect to processing times. For standard instances, it can be lowered as follows. We check if, on all machines, the sum of the shortest job processing times in a machine, without considering deterioration, is larger than the processing time of a single job in any of the other machines. If this is true, then we are sure that an optimal solution has at most $n - m$ jobs on a machines and at most $2(n - m)$ slots used. As the number of jobs is often much larger than the number of machines, this reduction is not so large, but in any case it allows us to get a significant reduction on the number of variables and an observable improvement in the solving performance.

5. Metaheuristic algorithm

In this section, we present the metaheuristic that we implemented to quickly obtain good-quality solutions for R|Sdd,mnt|C_{max} instances. It is an ILS-based algorithm with four local searches and two perturbations, that are used in an alternate way. The general ILS idea is to iteratively destroy the current solution and then apply one or more local search algorithms to look for new local optimal solutions. We refer to Lourenço et al. [2010] for further details on the ILS paradigm. The algorithm that we implemented is shown in Algorithm 1. It iterates the main loop for at most MAX_IT iterations without improvement in the incumbent solution and for at most MAX_TIME computing time.

The ILS procedure in Algorithm 1 receives in input the set J of jobs, the set M of machines, the maintenance time for each machine, the processing time of each job, and the deterioration delays of each job on each machine.

```
   input : J, M, p, d, t
   output: s*
1  nonImproveIters ← 0 ;
2  startTimer (time)
3  s* ← constructiveHeuristic (J, M, p, d, t);
4  s ← s*
5  while nonImproveIters ≤ MAX_IT  and  time < MAX_TIME do
6      nonImproveIters ← nonImproveIters + 1 ;
7      s ← intra2SwapLS (s, J, M, p, d, t);
8      s* ← updateBestGlobal (s, s*, nonImproveIters);
9      s ← intra3SwapLS (s, J, M, p, d, t);
10     s* ← updateBestGlobal (s, s*, nonImproveIters);
11     s ← inter2SwapLS (s, J, M, p, d, t);
12     s* ← updateBestGlobal (s, s*, nonImproveIters);
13     s ← inter3SwapLS (s, J, M, p, d, t);
14     s* ← updateBestGlobal (s, s*, nonImproveIters);
15     if nonImproveIters  mod 5 = 0 then
16         s ← doPertubation (s)
17     end
18 end
```

Algorithm 1: ILS heuristic

At line 3, the incumbent solution, represented by s*, is initialized by means of a constructive heuristic. In our work, we tried two constructive algorithms. The first, that we will reference as *longest processing time* (LPT), orders all jobs by non-increasing processing time on each machine and, after that, starting from the first machine, assigns to each machine the still unassigned job that has the longest processing time on the machine. The second, referenced as *lowest delay first* (LDF) orders job by increasing deterioration delay order. After that, as in the previous method, it starts from the first machine and inserts one job at a time on each machine, choosing the still unassigned job that has lowest delay delay.

In each constructive algorithm, once the jobs are inserted in the machines, we take into account the maintenance operations. The strategy that we use is to evaluate the accumulated delay, starting from the last job and proceeding backwards to the first. Every time the deterioration delay becomes larger than the maintenance time, a maintenance activity is inserted.

We developed two variants of the ILS, one called *LPT-ILS* that invokes the LPT algorithm, and the other, called *LDF-ILS*, that invokes the LDF.

At lines from 7 to 14 we apply a sequence of local search procedures. After any local search, the function *updateBestGlobal* is called to evaluate if the new solution we obtained is better than the incumbent solution. In such a case, we update s* and set the number of iterations without improvements, *nonImproveIters*, to 0.

The first local search performed, *intra2SwapLS*, swaps the position of two jobs on a given machine, attempting all machines, one at a time, and all pairs of jobs in the machine. The second local search in the sequence, called *intra3SwapLS*, changes the position of three jobs assigned to the same machine in two different ways. Starting from a tuple of three indexes (i_1, i_2, i_3), taken from three nested *for* loops, it tests two moves: (i_3, i_1, i_2) and (i_2, i_3, i_1). The other three possible moves are equivalent to simple two-position swaps, and are not executed.

The two following local searches are inter-machines movements. The *inter2SwapLS* procedure swaps two jobs assigned to two different machines. The machines are scanned according to their index, using two nested *for* loops, and then the jobs are scanned in the same way. Finally the *inter3SwapLS* procedure is analogous to the *inter2SwapLS*, but changes three jobs assigned to three different machines by adopting the same move strategy of the *intra3SwapLS* procedure.

If the condition shown at line 17 is satisfied then we proceed with a perturbation of the current solution. The perturbation procedure adopted iteratively invokes two functions. The first one reverses the order of all jobs in a machine and reevaluates the position of the maintenance activities. The second one shifts by one the job assignment on the machines, assigning to machine i all jobs currently assigned to machine $i + 1$, for all $i = 1, 2, \ldots, m - 1$. The jobs previously assigned to

machine 1 are assigned to machine m. The order of the jobs in this perturbation does not change; In both pertubations the position of maintenance activities are recalculated for each machine.

Due to performance issues, in the local search described above there are no maintenance position swaps, except on the *intra2SwapLS* function, where after each swap the maintenances are reallocated as described.

6. Computational experiments

To test the performance of the methods we proposed for the R|Sdd,mnt|C_{max}, we created a set of instances based on the parameters described in Ruiz-Torres et al. [2017]. We created three instances for each combination of number m of machines, ratio n/m of jobs per machine, deterioration interval and maintenance time interval, summing up to a total of 144 instances. We used this first set to test the MILP model. Then, we obtained a larger set by producing 10 random instances for each parameter combination, and we use it to test the heuristic methods. The instances that we created have identical processing times of each job on all machines, but different deterioration rates and maintenance times, in a semi-unrelated configuration. All algorithms we implemented, however, work with instances having different processing times on unrelated machines.

All the experiments were executed on a Ubuntu 18.04 LTS machine with 32 GB of ram and a Intel Xeon E3-1245 v5 processor with eight core of 3.5GHz . The algorithms were implemented using Julia language (version 1.1.0) and the MILP model was solved by Cplex 12.6 in default configuration, called through the Julia optimization library JuMP. We impose the MILP model to stop after one hour of computing time. As terminating conditions for the ILS, we imposed MAX_IT equal to 30 iterations and MAX_TIME equal to 30 computing minutes.

To compare the results of this ILS heuristic with those of the literature, we reimplemented the best heuristic proposed by Ruiz-Torres et al. [2017] for the case of identical machines, and called *Bh2c*. Algorithm Bh2c is a multistart constructive heuristic that, starting from a list of jobs ordered by processing time, tries to insert each job at the end of the group with lower completion time on the current solution or in a new group, being the total number of groups limited according to the current algorithm iteration. Each group represents a sequence of jobs between two maintenances. After that, as the machines are identical, the groups are assigned to the machines starting from the shortest to the longest, in order to minimize the makespan. There is no local search in Bh2c, which makes the algorithm very quick.

In our adaptation, called Bh2c-R, the jobs are also sorted by processing time, but the assignment of a job to a machine is done before the assignment of a job to a group. We chose the machine with lowest total processing time to receive the new job and, after that, the group is chosen like in the original method. This small change allows us to have an algorithm that is close to the one in the literature but that also works with unrelated machines.

We start our computational analysis in Table 1 with the description of the results that have been obtained by the heuristic procedures. The table reports the number m of machines, the ratio of jobs per machine n/m, the deterioration range d and the maintenance times range mtr. The table also reports, for each method, the percentage gap from the bext know solution value, the number of times the algorithm reached the best solution value, and the computing time in seconds. The rightmost column reports the best known solution value. Due to space limitations, each line reports average values on 10 instances. The last line reports overall average values.

The LPTF-ILS heuristic, that is the ILS based heuristic using the LPT constructive algorithm, obtained the best solution for 351 instances out of 480. The LDF-ILS heuristic obtained 74 best solutions, the heuristic adapted from the literature (Bh2c-R) 55. As stated before, Bh2c-R is a simple iterative-constructive procedure, so as expected it is very fast. It has an average processing

Table 1: Heuristic results comparison (10 instances per line)

Instance parameters				LPTF-ILS			LDF-ILS			Bh2C-R			Average best known solution value
m	n/m	d	mt	%gap	Best solutions count	Avg. time (sec)	%gap	Best solutions count	Avg. time (sec)	%gap	Best solutions count	Avg. time (sec)	
2	10	(1.01 - 1.05)	(1 - 3)	0.38	2	3.05	**0.00**	7	3.07	1.69	1	0.07	514.61
			(1 - 9)	1.39	3	2.86	**0.00**	7	3.30	5.50	0	0.00	502.10
		(1.05 - 1.10)	(1 - 3)	0.36	2	2.87	**0.00**	8	3.50	2.25	0	0.00	535.17
			(1 - 9)	**0.00**	5	3.20	2.06	4	2.83	9.68	1	0.00	567.39
	15	(1.01 - 1.05)	(1 - 3)	**0.00**	6	3.42	1.82	4	3.78	3.04	0	0.01	803.71
			(1 - 9)	**0.00**	6	3.47	0.94	4	3.93	5.77	0	0.01	864.16
		(1.05 - 1.10)	(1 - 3)	0.12	3	3.84	**0.00**	7	3.92	1.81	0	0.11	848.17
			(1 - 9)	**0.00**	4	3.80	1.00	6	3.57	8.31	0	0.01	831.74
	20	(1.01 - 1.05)	(1 - 3)	0.94	2	9.22	**0.00**	8	4.84	3.72	0	0.02	1026.15
			(1 - 9)	**0.00**	6	4.86	0.34	4	5.26	7.19	0	0.02	1092.55
		(1.05 - 1.10)	(1 - 3)	**0.00**	5	4.52	1.34	5	4.13	2.58	0	0.02	1072.73
			(1 - 9)	**0.00**	4	5.25	0.75	6	5.03	9.72	0	0.02	1065.71
5	10	(1.01 - 1.05)	(1 - 3)	**0.00**	10	14.63	6.09	0	15.79	2.63	0	0.05	534.55
			(1 - 9)	**0.00**	10	14.44	8.82	0	17.08	5.80	0	0.10	516.26
		(1.05 - 1.10)	(1 - 3)	**0.00**	9	16.60	7.64	1	16.64	2.04	0	0.11	535.88
			(1 - 9)	**0.00**	10	17.78	8.49	0	18.08	7.91	0	0.05	552.36
	15	(1.01 - 1.05)	(1 - 3)	**0.00**	10	20.90	6.50	0	21.30	2.99	0	0.17	786.20
			(1 - 9)	**0.00**	10	21.39	4.63	0	21.72	7.55	0	0.16	801.53
		(1.05 - 1.10)	(1 - 3)	**0.00**	9	20.07	5.83	1	21.63	2.60	0	0.18	818.34
			(1 - 9)	**0.00**	9	24.04	7.49	1	19.68	8.58	0	0.17	811.75
	20	(1.01 - 1.05)	(1 - 3)	**0.00**	10	32.55	8.63	0	30.83	3.33	0	0.39	1011.58
			(1 - 9)	**0.00**	9	27.64	7.40	1	25.85	7.39	0	0.38	1106.85
		(1.05 - 1.10)	(1 - 3)	**0.00**	10	27.68	4.70	0	32.07	2.94	0	0.40	1075.01
			(1 - 9)	**0.00**	10	27.54	6.12	0	30.65	10.70	0	0.39	1085.82
10	10	(1.01 - 1.05)	(1 - 3)	0.39	2	25.13	15.83	0	34.20	**0.00**	8	0.56	532.41
			(1 - 9)	**0.00**	10	23.08	14.69	0	28.55	2.50	0	0.56	554.58
		(1.05 - 1.10)	(1 - 3)	0.53	3	24.90	18.37	0	30.40	**0.00**	7	0.58	551.93
			(1 - 9)	**0.00**	10	24.49	13.53	0	29.11	5.22	0	0.56	573.56
	15	(1.01 - 1.05)	(1 - 3)	**0.00**	9	44.60	10.75	0	47.84	0.94	1	1.87	808.58
			(1 - 9)	**0.00**	10	44.12	10.70	0	59.40	5.03	0	1.86	800.08
		(1.05 - 1.10)	(1 - 3)	**0.00**	7	43.94	12.86	0	50.77	0.64	3	1.94	805.99
			(1 - 9)	**0.00**	10	40.67	10.74	0	49.48	7.61	0	1.90	832.13
	20	(1.01 - 1.05)	(1 - 3)	**0.00**	10	68.91	12.44	0	72.74	2.46	0	4.56	1082.39
			(1 - 9)	**0.00**	10	65.95	9.28	0	93.57	5.89	0	4.57	1083.80
		(1.05 - 1.10)	(1 - 3)	**0.00**	10	72.20	11.67	0	88.23	1.67	0	4.72	1057.23
			(1 - 9)	**0.00**	10	84.89	11.41	0	67.28	9.39	0	4.53	1123.47
20	10	(1.01 - 1.05)	(1 - 3)	2.00	0	68.75	20.52	0	103.05	**0.00**	10	7.03	549.68
			(1 - 9)	**0.00**	9	75.35	14.89	0	81.42	0.89	1	7.00	580.88
		(1.05 - 1.10)	(1 - 3)	3.37	0	78.88	24.89	0	81.06	**0.00**	10	7.23	542.77
			(1 - 9)	**0.00**	10	78.74	19.64	0	89.50	4.27	0	7.11	567.28
	15	(1.01 - 1.05)	(1 - 3)	**0.00**	7	193.55	15.82	0	206.72	0.14	3	24.43	826.65
			(1 - 9)	**0.00**	10	227.27	14.58	0	190.75	3.12	0	24.60	831.55
		(1.05 - 1.10)	(1 - 3)	0.68	2	182.42	18.13	0	207.64	**0.00**	8	24.96	828.06
			(1 - 9)	**0.00**	10	192.62	15.19	0	173.17	6.05	0	24.47	843.60
	20	(1.01 - 1.05)	(1 - 3)	**0.00**	10	394.63	15.75	0	356.60	1.28	0	59.33	1068.44
			(1 - 9)	**0.00**	10	406.43	13.06	0	489.29	4.78	0	58.96	1113.89
		(1.05 - 1.10)	(1 - 3)	**0.00**	8	463.36	14.42	0	456.11	0.42	2	60.53	1110.49
			(1 - 9)	**0.00**	10	488.31	15.99	0	346.62	7.34	0	60.38	1123.02
Average values				0.21	7.31	77.69	1.54	9.28	78.19	8.27	1.15	4.01	815.68

time of only 4.01 seconds. The two ILS algorithms are not too time consuming anyway, with LPT-ILS requiring 26.11 seconds on average, and LDF-ILS 77.69. Overall, the LPTF-ILS heuristic is the most performing algorithm, achieving the best solution on almost 80% of the cases.

Another interesting aspect is the fast increase of Bh2c-R heuristic runtime when the number of machines is not above 10. When compared with the ILS heuristic runtimes, we can notice a 300 times difference on small instances but just a 10 times difference on larger ones, indicating that the ILS heuristics scale better when m increases. Algorithm Bh2c-R has its bests results, in comparison with the other methods, for the large instances for which the maintenance times are in the interval between 1 and 3. Probably, this can be imputed to a lack of ability of the ILS algorithm in dealing with instances where maintenances are very useful to reduce the makespan.

In Table 2 we report the results obtained by the MILP model and compare them with the best known solution values obtained by the heuristics we implemented. Each line presents average values on three instances. For the model, we report the average solution value, the average best (lower) bound, the average gap between bound and solution value, and the number of nodes

Table 2: Model and best heuristic results comparison (3 instances per line)

Instance				Model				Best known solution value
m	n/m	d	mt	Avg. sol. value	Avg. best bound	Avg. %gap	Avg. nodes	
2	10	(1.00 - 1.05)	(1 - 3)	**502.98**	499.06	0.78	51859	514.66
			(1 - 9)	**496.22**	491.04	1.05	44012	509.67
		(1.05 - 1.10)	(1 - 3)	**427.46**	421.58	3.53	49022	441.89
			(1 - 9)	**514.84**	502.62	2.43	56723	542.7
	15	(1.00 - 1.05)	(1 - 3)	**847.00**	836.12	1.30	45716	870.18
			(1 - 9)	**815.21**	804.93	1.27	216.13	852.46
		(1.05 - 1.10)	(1 - 3)	**873.57**	837.55	4.30	46216	896.11
			(1 - 9)	**774.91**	700.39	10.64	64965	805.64
	20	(1.00 - 1.05)	(1 - 3)	**1021.08**	687.24	48.57	13248	1041.18
			(1 - 9)	**723.35**	470.23	53.82	47919	760.27
		(1.05 - 1.10)	(1 - 3)	**680.42**	170.05	300.12	66931	707.68
			(1 - 9)	**592.05**	243.35	143.29	140114	613.83
5	10	(1.00 - 1.05)	(1 - 3)	**364.89**	***	***	3203	362.68
			(1 - 9)	**328.45**	***	***	3786	339.67
		(1.05 - 1.10)	(1 - 3)	378.12	***	***	4629	**368.17**
			(1 - 9)	1370.61	***	****	3536	**358.03**
	15	(1.00 - 1.05)	(1 - 3)	1378.46	***	***	17	**497.89**
			(1 - 9)	723.50	***	***	376	**524.73**
		(1.05 - 1.10)	(1 - 3)	1491.68	***	***	595	**552.68**
			(1 - 9)	683.27	***	***	481	**508.71**
	20	(1.00 - 1.05)	(1 - 3)	15897.62	***	***	13	**652.20**
			(1 - 9)	1031.06	***	***	5	**743.93**
		(1.05 - 1.10)	(1 - 3)	1890.07	***	***	63	**690.91**
			(1 - 9)	1913.33	***	***	19	**796.70**
10	10	(1.00 - 1.05)	(1 - 3)	179820.66	***	***	0	**364.74**
			(1 - 9)	15567.00	***	***	1	**363.80**
		(1.05 - 1.10)	(1 - 3)	7549.66	***	***	3	**353.01**
			(1 - 9)	273212.22	***	***	0	**372.06**
	15	(1.00 - 1.05)	(1 - 3)	166787.6	***	***	0	**536.14**
			(1 - 9)	176257.33	***	***	0	**540.64**
		(1.05 - 1.10)	(1 - 3)	***	***	***	***	541.17
			(1 - 9)	***	***	***	***	559.34
	20	(1.00 - 1.05)	(1 - 3)	***	***	***	***	769.93
			(1 - 9)	***	***	***	***	694.61
		(1.05 - 1.10)	(1 - 3)	***	***	***	***	684.96
			(1 - 9)	***	***	***	***	779.14
20	10	(1.00 - 1.05)	(1 - 3)	***	***	***	***	369.58
			(1 - 9)	***	***	***	***	370.44
		(1.05 - 1.10)	(1 - 3)	***	***	***	***	348.38
			(1 - 9)	***	***	***	***	370.26
	15	(1.00 - 1.05)	(1 - 3)	***	***	***	***	571.31
			(1 - 9)	***	***	***	***	555.56
		(1.05 - 1.10)	(1 - 3)	***	***	***	***	557.09
			(1 - 9)	***	***	***	***	551.29
	20	(1.00 - 1.05)	(1 - 3)	***	***	***	***	703.07
			(1 - 9)	***	***	***	***	752.58
		(1.05 - 1.10)	(1 - 3)	***	***	***	***	762.87
			(1 - 9)	***	***	***	***	744.11

explored. None of the model solutions is proven optimal. The symbol "***", reported in the columns for many instances, means that the model execution was interrupted without providing the value in the column.

We can observe from the table that the model is effective in finding good-quality solutions on the small instances, especially for those involving only two machines. This shows that there is still room for improvement for our metaheuristic methods. As the number of machines increases, the heuristic solutions become better and then, when the number of jobs is greater than 150 (and so the number of slots is greater than 350), the model is not able to find solutions anymore and the heuristic remain the only possible solution method. In most of cases where a solution is not found, the process has been aborted for exceeding the machine memory capacity.

We also notice that very large objective function values have been found by the model on instances with 100 jobs and 5 machines, 100 jobs and 10 machines, and 150 jobs and 10 machines. These have been found at the root node, on in very early nodes of the enumeration tree, by the heuristics in Cplex. The difficulty in creating new branches, because of the large size of the model, forbid the solver in exploring more in depth the space of feasible solutions. In these solutions, what commonly happen is a scheduling of several jobs in a few machines without any maintenance,

making the deterioration consistently grow consistently, and so the makespan.

Overall, the model provided feasible solutions for more than 60% of the tested instances. Even if no proven optimal solution was found, the model could solve some hard cases with just two machines in a better better way than the heuristic. In our next investigations, we will try to create better models and prove optimality for some instances of small or moderate size.

7. Conclusions and future research directions

In this work, we studied the problem of assigning jobs on unrelated parallel machines by considering sequence dependent deterioration and the possibility to introduce maintenance activities to restore full operational speed on a machine. We presented a new mixed integer linear programming model that linearizes a previous model from the literature, and a metaheuristic algorithm based on the concept of iterated local search. The model we proposed was able to get feasible solutions for more than 60% of the tested instances, although just a few of them were solved to proven optimality. The metaheuristic obtained good results on all instances, comparing will with the state-of-art algorithm in the literature, especially in the case of high maintenance times.

As future research, we intend to enlarge the computational tests so as to have a better understanding of the difficulty of the problem. We then plan to improve the performance of the metaheuristic that we proposed, by exploring better ways to deal with the maintenance positions or trying alternative solution representations. We will also look for the development of alternative models that could enable to solve to proven optimality instances of moderate size. We are also interested in investigating the impact of maintenance activities and sequence dependent deteriorations in other scheduling problems, foe example considering weighted completion time, as done in Kramer et al. [2019], or weighted tardiness.

References

Browne, S. and Yechiali, U. (1990). Scheduling deteriorating jobs on a single processor. *Operations Research*, 38(3):495–498.

Cheng, T. and Ding, Q. (2001). Single machine scheduling with step-deteriorating processing times. *European Journal of Operational Research*, 134(3):623–630.

de Araújo, O. C. B., Dhein, G., and Fampa, M. (2017). Minimizing the makespan on parallel machines with sequence dependent deteriorating effects. In *XLIX SBPO, Simpósio Brasileiro de Pesquisa Operacional*.

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

Graham, R., Lawler, E., Lenstra, J., and Kan, A. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. In Hammer, P., Johnson, E., and Korte, B., editors, *Discrete Optimization II*, volume 5 of *Annals of Discrete Mathematics*, p. 287–326. Elsevier.

Gupta, J. N. and Gupta, S. K. (1988). Single facility scheduling with nonlinear processing times. *Computers & Industrial Engineering*, 14(4):387–393.

Huang, X. and Wang, J.-J. (2015). Machine scheduling problems with a position-dependent deterioration. *Applied Mathematical Modelling*, 39(10):2897–2908.

Ji, M. and Cheng, T. (2008). Parallel-machine scheduling with simple linear deterioration to minimize total completion time. *European Journal of Operational Research*, 188(2):342–347.

Kramer, A., Dell'Amico, M., and 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.

Lalla-Ruiz, E. and Voß, S. (2016). Modeling the parallel machine scheduling problem with step deteriorating jobs. *European Journal of Operational Research*, 255(1):21–33.

Leung, J. Y.-T., Ng, C., and Cheng, T. E. (2008). Minimizing sum of completion times for batch scheduling of jobs with deteriorating processing times. *European Journal of Operational Research*, 187(3):1090–1099.

Lourenço, H., Martin, O., and Stützle, T. (2010). Iterated local search: Framework and applications. In Gendreau, M. and Potvin, J.-Y., editors, *Handbook of Metaheuristics*, International Series in Operations Research and Management Science, chapter 12, p. 362–397. Springer, 2 edition.

Mosheiov, G. (1991). V-shaped policies for scheduling deteriorating jobs. *Operations Research*, 39 (6):979–991.

Mosheiov, G. (1994). Scheduling jobs under simple linear deterioration. *Computers & Operations Research*, 21(6):653–659.

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

Pinedo, M. (2016). *Scheduling: theory, algorithms and systems development*. Springer, New York, 5 edition.

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

Ruiz-Torres, A. J., Paletta, G., and Pérez, E. (2013). Parallel machine scheduling to minimize the makespan with sequence dependent deteriorating effects. *Computers & Operations Research*, 40 (8):2051–2061.

Santos, V. L. A. and Arroyo, J. E. C. (2015). Sequenciamento de tarefas em maquinas paralelas considerando desgastes dependentes da sequencia. In *XLVII SBPO, Simpósio Brasileiro de Pesquisa Operacional*, p. 2572–2583.

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

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.