Permutation Flowshop Scheduling with periodic maintenance and makespan objective [*]

Paz Perez-Gonzalez[1†], Victor Fernandez-Viagas[1], Jose M. Framinan[1]

[1] Industrial Management, School of Engineering, University of Seville,

Ave. Descubrimientos s/n, E41092 Seville, Spain, {pazperez, vfernandezviagas, framinan}@us.es

March 26, 2020

Abstract

In this paper, we address the permutation flowshop scheduling problem with cyclical unavailability periods where no operation can be processed. Under this constraint, all machines must stop at the same time due to the shift calendar (shift changes, nights, weekends, etc.), or due to preventive deterministic and fixed maintenance activities. For this reason, this constraint is known in the literature as periodic maintenance. Although different decision problems dealing with the simultaneous scheduling of jobs and maintenance activities have been studied in the literature, scheduling with periodic maintenance has been only addressed for the single machine and parallel machines layouts, and we are not aware of references tackling the flowshop. In this layout, different scheduling problems arise depending on the assumptions about the preemption of the operations. Here we focus on scheduling jobs in a flowshop with the objective of minimising the makespan assuming that the preemption of operations is not allowed, and therefore, if an operation cannot be finished within the current availability period, then it should be scheduled in the next one. The structure and hardness of the problem depending on the size of the availability periods is studied using Mixed Integer Linear Programming and complete enumeration, in order to determine the range of values for the availability period that makes the problem under consideration to be substantially different than its classical (unconstrained) counterpart. For these cases, specific heuristics with different computational complexity are developed, and an extensive computational experience is carried out to establish the efficiency of the proposed heuristics.

[*]This is the Submmited Manuscript of an article published by Elsevier in Computers and Industrial Engineering 143, 106369 available online: https://doi.org/10.1016/j.cie.2020.106369

[†]Corresponding author. Tel.: +34-954487214.

Keywords: Permutation flowshop scheduling, makespan, Heuristics, periodic maintenance, shift calendar, cyclical availability constraint

1 Introduction

In most manufacturing scheduling literature it is usually considered that machines are available during all the scheduling horizon. However, a number of causes (such as e.g. scheduled stops, machine breakdowns, etc.) may generate unavailability periods in the machines, so the jobs cannot be processed in these periods. In this paper we are interested in a special case of unavailability constraint, quite common in manufacturing companies, where all machines stop periodically after a given time interval. Some examples of these unavailability periods can be produced by periodic maintenance activities, breaks, end of shifts, etc. More specifically, our work is motivated by the natural interruption of work in the factory between one shift and the next one. In this case, a working shift has a constant duration and, since workers may change from a shift to the next one, unfinished operations are not allowed. This is a usual practice in many manufacturing companies with relatively complex manual operations, such as the assembly of wiring harness in the aerospace industry, where shifts are formed by different teams of workers and having one worker to complete the tasks of a previous worker is not desirable in terms of efficiency and quality of the operation. In our research, this periodic unavailability constraint, denoted in the scheduling literature as *periodic maintenance*, takes place in a flowshop layout where jobs are processed in the machines following all of them the same route. Furthermore, the order of jobs in all machines is the same (permutation assumption). Note that different scheduling problems can be defined depending on the assumption regarding the preemption of the operations. In our paper, operation-related preemption is not allowed, so if an operation cannot be finished in a working shift, then it should be scheduled in the next one. This problem is denoted as Permutation Flowshop Scheduling Problem with Periodic Maintenance (PFSP-PM), and the objective function to be minimised is the maximum completion time of the jobs (makespan), which is known to be related to the maximisation of the production rate (Framinan et al., 2014), being the most-studied objective in the permutation flowshop scheduling literature (Fernandez-Viagas et al., 2017).

Although scheduling problems considering unavailability constraints have been widely studied for different layouts (see e.g. Ma et al., 2009 for a review on the topic), this is not the case for the flowshop layout, where only very specific cases have been addressed: Allaoui et al. (2006) and

Xu et al. (2018) consider an unavailability period on the first machine in a two-machine flowshop, while Labidi et al. (2018) study the two-machine flowshop scheduling problem with unavailability and additional new constraints such as no-wait and non-zero release dates. Finally, Perez-Gonzalez and Framinan (2009) and Fernandez-Viagas and Framinan (2017) consider the m-machine flowshop with an unavailability period on each machine only at the beginning of the scheduling horizon. On the other hand, focusing on scheduling papers with periodic maintenance, the problem has been studied for a single machine (see e.g. the recent papers by Angel-Bello et al., 2011; Yu et al., 2014; Yazdani et al., 2018; Perez-Gonzalez and Framinan, 2018), and for parallel machines (see e.g. Kaabi and Harrath, 2019). To the best of our knowledge, this constraint has not been considered in the flowshop layout. Note that, since the single machine case with periodic maintenance is NP-hard in the strong sense (Hsu et al., 2010; Low et al., 2010), PFSP-PM is NP-hard even for two machines, in contrast to the classical counterpart (Permutation Flowshop Scheduling Problem or PFSP) where Johnson's rule is optimal for a two-machine flowshop. However, it is clear that, if the availability periods are sufficiently long, it may be possible to schedule all jobs within the shift, therefore the unavailability constraint would not apply, and efficient methods for PFSP could be used. Therefore, it is of interest to establish the range of values of the availability periods that make the PFSP-PM to be different than the PFSP. For these cases, specific approximate algorithms would have to be developed, as it is foreseeable that borrowing methods from the PFSP may not yield good solutions for PFSP-PM.

The remainder of the paper is structured as follows: The PSFS-PM problem is described and formalised in Section 2. Some properties of the problem and a mixed integer linear programming model (MILP) are presented in Section 3 in order to carry out a computational analysis to determine the hardness of the problem and its relation with the classical PFSP. To do so, an extensive testbed with different instance sizes and values of the availability periods is presented in Section 4. The results of the analysis are discussed in Section 5. For the cases where the PFSP-PM is found to be different than the PSFP, specific approximate procedures are proposed in Section 6. These procedures are subsequently tested in an extensive computational experience in Section 7. Finally, the conclusions obtained are presented in Section 8.

2 Notation and identification of related decision problems

The permutation flowshop scheduling problem consists of scheduling n jobs in m machines, such as each job has an operation on each machine, and the route of the jobs is the same for all of them. Operation of job j in machine i has a processing time p_{ij}. The permutation constraint is considered, i.e. the sequence of jobs is the same for all machines, and the objective is the makespan, C_{max}, defined as the completion time of the last job in the last machine. Additionally, jobs are available at the beginning of the scheduling horizon. Without further considerations, the so-defined problem is the Permutation Flowshop Scheduling Problem (PFSP). In our problem, there is an additional constraint related to the availability of the machines: i.e. each T units of time, all machines are not available during τ units of time.

The most critical assumption with respect to the unavailability constraint considered in the literature (Ma et al., 2009) is related to different hypotheses about the preemption of the jobs if they cannot be completed within an availability period:

- Resumable case (rs). If an operation cannot be finished before the unavailability period, it is preempted and it can continue when the machine is available again.
- Non-resumable case (nr). Preemption is not allowed and the disrupted operation has to restart completely rather than continue.
- Semi-resumable case (sr). The disrupted operation will have to partially restart, usually multiplying the remaining processing time by a factor $\alpha > 1$.

Furthermore, in the permutation flowshop layout, each one of the above preemption-related assumptions can be considered at operation level (see Figure 1), or at job level (see Figure 2). By combining the assumptions and the levels, the following problems can be identified and noted extending the notation by Graham et al. (1979) and Low et al. (2010):

- Operation level:

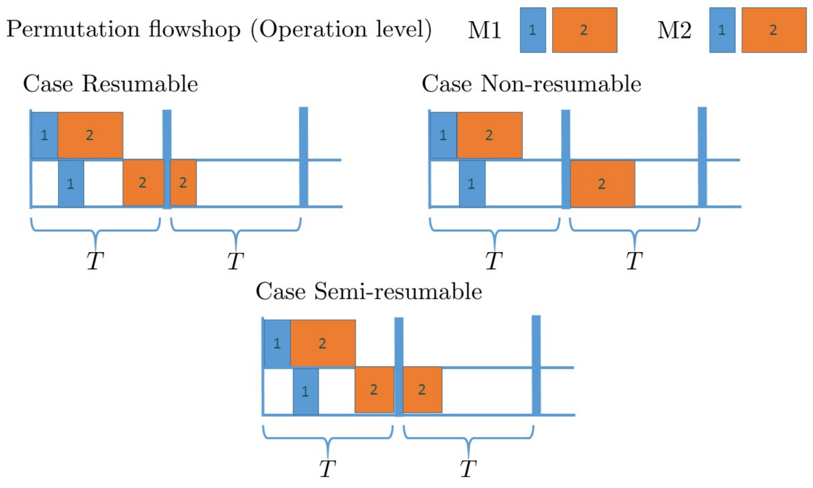

Figure 1: Permutation flowshop scheduling with periodic maintenance: Operation level

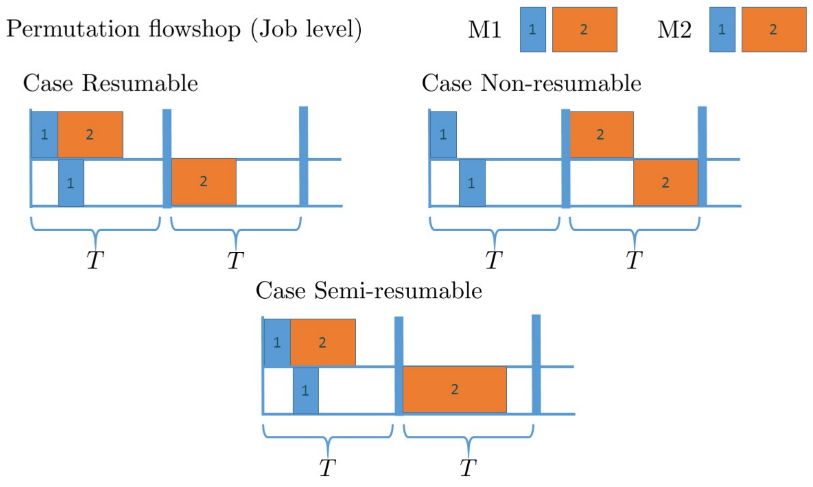

Figure 2: Permutation flowshop scheduling with periodic maintenance: Job level

– Resumable periodic maintenance ($rs - pm$): It is easy to see that the problem $Fm|prmu, rs - pm|C_{max}$ is equivalent to the classical problem $Fm|prmu|C_{max}$, which has been widely studied in the literature.

– Non-resumable periodic maintenance ($nr - pm$): In this case, operations cannot be preempted, and if an operation cannot be finished in a shift, then it is scheduled in the following shift. This problem, denoted $Fm|prmu, nr - pm|C_{max}$, is NP-hard in the strong sense as explained previously since $1|nr-pm|C_{max}$ is NP-hard in the strong sense (Hsu et al., 2010; Low et al., 2010). To the best of our knowledge, this problem has not been studied before, and it is the motivation of our research.

  - Semi-resumable periodic maintenance ($sr - pm$): The operation may be preempted if it cannot be finished in a shift, and the remaining processing time is modified when it is processed in the following shift. The resulting $Fm|prmu, sr - pm|C_{max}$ problem has not been considered previously in the literature either.

- Job level:

  - Resumable periodic maintenance at job level ($rs - pm(job)$): The job can be preempted, although the operations of the jobs cannot be interrupted. I.e. if one operation cannot be finished in a shift, it is scheduled in the following shift, and the job has operations within different working shifts. In this case the problem is denoted $Fm|prmu, rs - pm(job)|C_{max}$. Note that this problem is equivalent to the $Fm|prmu, nr - pm|C_{max}$ problem, i.e it is equivalent to the non-resumable case in the operation level, so the optimal solution is the same for both problems.
  - Non-resumable periodic maintenance at job level ($nr - pm(job)$): All operations of a job must be scheduled in the same shift, i.e if all operations cannot be finished in a shift, the complete job must be scheduled in the next shift. In this case the problem is denoted as $Fm|prmu, nr - pm(job)|C_{max}$. This problem is different to the resumable case and to the operation level problems defined previously and it has not been considered in the literature.
  - Semi-resumable periodic maintenance at job level ($sr - pm(job)$): When all operations of a job cannot be scheduled in the same shift, those operations scheduled in the following shift have a penalty in the processing times. Note that this problem is different to the semi-resumable case in the operation level. This problem, denoted as $Fm|prmu, sr - pm(job)|C_{max}$, has not been studied before.

The problems identified are summarized in Table 1. It can be observed that, out of the six possibilities, four new problems are identified: $Fm|prmu, nr - pm|C_{max}$, $Fm|prmu, sr - pm|C_{max}$, $Fm|prmu, nr - pm(job)|C_{max}$ and $Fm|prmu, sr - pm(job)|C_{max}$. Among these problems, in this paper we address the problem $Fm|prmu, nr - pm|C_{max}$, i.e. the problem with machine unavailability non-resumable at the operation level, which is equivalent to resumable at the job level. As explained

Level	Resumable	Non-resumable	Semi-resumable
Operation	$Fm\|prmu\|C_{max}$	$Fm\|prmu, nr-pm\|C_{max}$	$Fm\|prmu, sr-pm\|C_{max}$
Job	$Fm\|prmu, nr-pm\|C_{max}$	$Fm\|prmu, nr-pm(job)\|C_{max}$	$Fm\|prmu, sr-pm(job)\|C_{max}$

Table 1: Approaches for permutation flowshop scheduling problems with periodic maintenance constraints

in Section 1, in the manufacturing process inspiring this problem, unfinished operations are not allowed since working teams change in each shift, and operations started by a working team should be finished by the same working team. In the next section, we analyse this problem in detail.

3 Problem properties and Mathematical programming model

The considered problem, $Fm|prmu, nr-pm|C_{max}$, is denoted in this paper as PFSP-PM. As explained previously, this problem consists on scheduling a set of n jobs within shifts of length T. Between two shifts there is an unavailability period of length τ. Without loss of generality, we consider $\tau = 0$ since τ does not influence the scheduling decision, although naturally the value of the makespan would be different.

The following observations can be done for the PFSP-PM:

Observ. 1. $T \geq \max p_{ij} = \max_{\{1\leq i\leq m; 1\leq j\leq n\}} p_{ij}$, since each operation should fit in a shift, otherwise, the problem is unfeasible.

Observ. 2. $T < C_{max}(S)$, $\forall$ S schedule. Otherwise, if T is too loose, the problem is tantamount to the PFSP.

Observ. 3. The number of shifts used for each schedule is *a priori* unknown. However, the maximal number of shifts is bounded by $\max\{1, \lceil \sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij}/T \rceil\}$.

Observ. 4. The PFSP-PM is NP-hard even for $m = 2$ (see previous section).

Observ. 5. The concept of critical path of PFSP (see e.g. Pinedo, 2008) does not apply for PFSP-PM.

Observ. 6. The reversibility property is not satisfied, even in the two machines problem for the resumable case (see Lee, 1997).

Observ. 7. As a consequence of the previous observation, Taillard's accelerations (Taillard, 1990) cannot be implemented to compute the makespan more efficiently.

Taking into account observations 1 and 2, it is clear that the structure of PFSP-PM depends on the parameter T. Hence, PFSP-PM can be unfeasible for small values of T (i.e. when $T < \max p_{ij}$), or the constraint may have no effect for big values of T, converting the PFSP-PM into its classical counterpart PFSP. In order to determine the differences between these two problems, a Mixed Integer Linear Programming (MILP) model can be developed to optimally solve this problem and to compare it with the solutions of the PFSP. According to Stafford et al. (2005), the most efficient MILP for the PFSP is the TS2 of the Wilson family models. The adaptation of this model to the PFSP-PM is described below, using the same notation than in Stafford et al. (2005).

Parameters

n	number of jobs
m	number of machines
p_{ij}	processing time of job j in machine i
T	length of the shifts
M	big number. It can be computed as $M = \sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij}$
S	maximal number of shifts. According to the Observ. 3, it can be computed as $\max\{1, \lceil M/T \rceil\}$

Indexes

i	Machines $1 \leq i \leq m$
j	Jobs $1 \leq j \leq n$
l	Positions $1 \leq l \leq n$
s	Shifts $1 \leq s \leq S$

Variables

E_{ij} Completion time of job in position j in machine i

$$Z_{jl} \begin{cases} 1 & \text{job } j \text{ is scheduled in position} l \\ 0 & \text{in other case} \end{cases}$$

$$\gamma_{ils} \begin{cases} 1 & \text{job in position } l \text{ is scheduled in the shift } s \text{ in machine } i \\ 0 & \text{in other case} \end{cases}$$

Model:

$$\min E_{mn} \tag{1}$$

s.t.

$$\sum_{l=1}^{n} Z_{jl} = 1 \qquad 1 \le j \le n \tag{2}$$

$$\sum_{j=1}^{n} Z_{jl} = 1 \qquad 1 \le l \le n \tag{3}$$

$$E_{il} + \sum_{j=1}^{n} p_{ij} Z_{jl+1} \le E_{il+1} \qquad 1 \le i \le m,\ 1 \le l \le n-1 \tag{4}$$

$$E_{il} + \sum_{j=1}^{n} p_{i+1j} Z_{jl} \le E_{i+1l} \qquad 1 \le i \le m-1,\ 1 \le l \le n \tag{5}$$

$$E_{11} \ge \sum_{j=1}^{n} p_{1j} Z_{j1} \tag{6}$$

$$E_{il} - sT \le M(1 - \gamma_{ils}) \qquad 1 \le i \le m,\ 1 \le l \le n,\ 1 \le s \le S \tag{7}$$

$$E_{il} - \sum_{j=1}^{n} p_{ij} Z_{jl} + M(1 - \gamma_{ils}) \ge T(s-1) \qquad 1 \le i \le m,\ 1 \le l \le n,\ 1 \le s \le S \tag{8}$$

$$\sum_{s=1}^{S} \gamma_{ils} = 1 \qquad 1 \le i \le m,\ 1 \le l \le n \tag{9}$$

Equation (1) states the objective to be minimized (makespan), defined as the completion time in the last machine of the job in the last position. Equations (2) force that each job is assigned to only one position in the sequence, while equations (3) ensure that each position is occupied by only one job. Equations (4) force that, for each machine, two consecutive jobs do not overlap their processing, while equations (5) impose that operations of a job in two consecutive machines do not overlap. Equation (6) computes the completion time of the job scheduled in the first position in the first machine. Sets of equations (7) and (8) control that each operation starting in a shift also finishes within this shift. Finally, the set of equations (9) ensures that each operation is scheduled in one shift.

The model will be used in Section 5 to analyse the problem and its similarities with the PFSP by means of a computational evaluation of the optimal solutions. To do so, an extensive set of instances to conduct the computational evaluation is first developed in Section 4.

4 Sets of instances

Several sets of instances are presented in this section. The first two sets, described in detail in Section 4.1, are composed of small-size instances so the problem can be optimally solved in reasonable computation times using the MILP model presented in the previous section, or by complete enumeration of the solutions. The results of this analysis are presented in Section 5. On the other hand, in subsection 4.2, the well-known Taillard's testbed (Taillard, 1993) is adapted for the problem under consideration, so different methods to generate T are presented in order to develop a set of instances for our problem. This set will be used in Section 6 to test the efficiency of the heuristic methods proposed for the problem.

4.1 Small-size instances

The following two sets of small-size instances have been generated, in both cases with the processing times $p_{ij} \sim U[1,99]$ as usual in the related literature (see e.g. Taillard, 1993 and Vallada et al., 2015):

- Testbed β_1: This set is composed of small-size instances, so the can be optimally solved using the MILP in reasonable computation time. More specifically, 30 instances per combination of values of $n \in \{5,10\}$, $m \in \{2,5,10\}$ have been generated, being a total of 180 instances.

- Testbed β_2: This set is composed of small-size instances, so the problem can be optimally solved by complete enumeration (CE) in reasonable computation time. More specifically, 200 instances have been generated for each combination of $n \times m \in \{5 \times 10, 10 \times 10, 10 \times 5\}$, to provide three scenarios ($n < m$, $n = m$ and $n > m$). Due to the high computational times needed, 10 jobs is the biggest case that can be solved within reasonable computational effort. This set contains 600 instances.

For each one of the instances in the testbeds, different values of T have been generated. The objective is to generate a set of values of T ranging from tight to loose values with respect to an upper bound of the worst makespan for the unconstrained instance (i.e. the PFSP), thus providing cases where the problem is either more constrained (tight T values) or more similar to the PFSP (loose T values). Therefore, $T \in \{100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, M\}$ with $M = \sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij}$. All instances in β_1 and β_2 are combined with each value of T. Note that, for $T = M$, the PFSP-PM is equivalent to PFSP according to the Observ. 2 in Section 3.

4.2 Big-size instances

Taillard's testbed (Taillard, 1993) has been widely used in the permutation flowshop scheduling literature. It is formed by 10 instances per case of the following combinations of $n \times m \in \{20 \times 5, 20 \times 10, 20 \times 20, 50 \times 5, 50 \times 10, 50 \times 50, 100 \times 5, 100 \times 10, 100 \times 20, 200 \times 10, 200 \times 20, 500 \times 20\}$. For each instance in this testbed, two methods to generate T have been employed:

- Constant T. $T \in \{100, 200, 300, 400\}$. This set of values has been adopted in view of the results of the analysis carried out in Section 5.1.

- Variable T. In this case T is not the same value for all instances of the testbed. The value of T is dependent on the size (being the same for all instances for a given size), in order to simulate the scheduling process on a rolling horizon basis, in which jobs with different characteristics arrive to a system where the value of the availability period is the same. In this case, inspired by the due date generation by Potts and Van Wassenhove (1982) and Armentano and Ronconi (1999), we generate $T \sim U[P(1 + t + \frac{r}{2}), P(1 + t + r)]$, where t and r are control parameters described below, and $P = \max p_{ij}$.

This method guarantees to obtain feasible values of T (see Section 3, Observ. 1). In this approach, parameter t is seen as a tightness factor (since the smaller t, the tighter T) and r is the dispersion range factor. Several scenarios with different values of tightness factor and dispersion range are tested to provide cases where T is loose or tight (to study the effect of the length of the availability period as compared to the processing times) as well as cases where the values of T are very different or similar between the sizes of the instances (to study the effect of the dispersion of the availability period around its average). More specifically, the values used for each parameter are $t \in \{0.2, 0.4, 0.6\}$, representing low, medium and high tightness, and $r \in \{0.6, 1.2, 1.8\}$ representing low, medium and high dispersion range.

The values obtained for Taillard's testbed with this method and the main statistics provided for the combination of $r \in \{0.6, 1.2, 1.8\}$ and $t \in \{0.2, 0.4, 0.6\}$ are presented in Table 2.

t		0.2			0.4			0.6	
r	0.6	1.2	1.8	0.6	1.2	1.8	0.6	1.2	1.8
Mean	164.14	199.62	255.56	184.20	232.62	262.76	200.69	228.50	281.20
Maximum	176	234	287	197	252	310	214	255	332
Minimum	149	180	206	166	201	233	189	217	248
St. Dev.	10.18	19.61	13.83	6.82	15.17	24.91	9.12	9.81	20.47

Table 2: Main statistics for the values of T, depending on r and t: Taillard Test bed

According to the values of T obtained, and taking into account Table 2, the following representative scenarios have been selected:

Scenario 1. Homogeneously Tight (denoted HoT): Low tightness factor $t = 0.2$ and small dispersion range $r = 0.6$. This scenario is the one providing the lowest mean value among the values in Table 2.

Scenario 2. Homogeneously Loose (denoted HoL): High tightness factor $t = 0.6$ and small dispersion range $r = 0.6$. This scenario is the one providing the combination of lowest standard deviation with highest mean values in Table 2.

Scenario 3. Heterogeneously Tight (denoted HeT): Low tightness factor $t = 0.2$ and medium dispersion range $r = 1.2$. This scenario is the one providing the combination of highest standard deviation with lowest mean values in Table 2.

Scenario 4. Heterogeneously Loose (denoted HeL): Medium tightness factor $t = 0.4$ and high dispersion range $r = 1.8$. This scenario is the one providing the highest standard deviation among the values in Table 2.

5 Analysis of the problem

The objective of this analysis is to determine the values of the parameter T for which the differences between the PFSP-PM and PFSP are not significant, i.e., a schedule given for the PFSP is a good solution for the PFSP-PM. The problem is analysed on the one hand by exact solutions using the MILP (Subsection 5.1) and, on the other hand, using complete enumeration (Subsection 5.2).

5.1 Analysis of the optimal solutions using MILP

In order to compare PFSP with PFSP-PM we have solved the instances of β_1, presented in Section 4, using the MILP model presented in Section 3. Each instance $I \in \beta_1$ has been solved using the solver Gurobi 7.0 (Gurobi Optimization, 2018) with a time limit of 900 seconds, for $T \in \{100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, M\}$. More specifically, for each instance, the following values are tallied:

- For each $T \neq M$, the makespan of the optimal/best schedule obtained is denoted as C^*_{max}.
- For $T = M$, PFSP-PM is equivalent to the PFSP, and the makespan value obtained is denoted as LB, since this value is a lower bound of the makespan for the PFSP-PM.
- Additionally, in order to know if the optimal schedule for the PFSP is a good solution for the PFSP-PM, the optimal schedule obtained for $T = M$ is evaluated for the rest of the values of T. The so-obtained makespan value is denoted as UB, since, in general[1], this value is an upper bound for the PFSP-PM.

The comparison has been carried using the relative percentage deviation from LB measured as

[1]Note that this value can be lower than C^*_{max} in the case that Gurobi does not reach the optimal for PFSP-PM, but it is always greater than LB.

the following formula for each instance $I \in \beta_1$ and objective value $VAL \in \{C^*_{max}, UB\}$:

$$RPD = \frac{VAL - LB}{LB} \cdot 100 \tag{10}$$

Table 3 shows the Average RPD (ARPD) obtained for each combination of n and m, for C^*_{max} (in order to observe the performance of the optimal/best makespan value of PFSP-PM), and for UB (in order to show the performance of the optimal solution provided for PFSP evaluated for PFSP-PM). Note that the results for $T = M$ (case PFSP) are not included, since its ARPD is equal to zero. Gurobi has solved optimally almost all instances within the 900 seconds, except for the case $T = 100$ with $n \times m = 10 \times 10$, where only 5 out of 30 instances were optimally solved (this fact is indicated by the symbol *). The average computational time, CPU, needed by Gurobi to solve the instances of the PFSP-PM for each value of T is shown in the last row. On average, Gurobi has needed 0.3118 seconds to solve an instance for PFSP.

From Table 3 in the case $T \geq 400$ it can be seen that, for many instance sizes, C^*_{max} as well as UB coincide with LB, with ARPD values equal to zero, showing that PFSP and PFSP-PM have the same optima.

Regarding C^*_{max}, it can be seen that, as T increases, it gets closer to the optimal makespan value of the PFSP as expected, being only 0.23 % for $T = 1000$. UB has the same behaviour, being around 0.71, 0.65 and 0.52 % worse than C^*_{max} for $T = 800, 900$ and 1000 respectively. For the smallest values of T, 100 and 200, the solution of the PFSP-PM is far from the solution of PFSP for all sizes, being the differences between C^*_{max} and UB more than 11% for $T = 100$ and more than 10% for $T = 200$. For $T = 300$ and $T = 400$ these differences are more than 6% and more than 4%, respectively. For $T \geq 500$, the differences are below 3%.

Therefore, from these results we can conclude that:

- Small sizes instances of the problem PFSP-PM when $T \in \{800, 900, 1000\}$ may be solved approximately using the MILP of PFSP, as there are small differences between the so-obtained solution and the optimal solution of the PSFP-PM.

- For the smallest values of T (100 and 200), using the solution of the PFSP to solve the PFSP-PM does not seem a good option. Additionally, Gurobi needs higher computation times to

	n	m	100	200	300	400	500	600	700	800	900	1000
C^*_{max}	5	2	16.978	5.148	3.467	0.000	0.000	0.000	0.000	0.000	0.000	0.000
		5	25.470	13.713	5.101	4.566	2.266	0.000	0.000	0.000	0.000	0.000
		10	29.731	16.058	8.389	5.534	3.518	3.748	2.589	0.706	0.168	0.000
	10	2	16.231	1.891	0.648	0.708	1.076	0.322	0.000	0.000	0.000	0.000
		5	20.183	9.368	4.373	2.686	2.635	1.978	1.816	0.215	0.000	0.000
		10	29.466*	13.989	8.080	5.158	4.853	2.622	2.141	2.415	2.353	1.383
		Total	23.010	10.028	5.010	3.109	2.391	1.445	1.091	0.556	0.420	0.230
CPU PFSP-PM			142.24	13.65	2.15	1.27	0.93	0.86	0.50	0.58	0.51	0.39
UB	5	2	26.551	14.433	5.711	0.000	0.000	0.000	0.000	0.000	0.000	0.000
		5	35.805	19.255	8.673	8.255	3.523	0.000	0.000	0.000	0.000	0.000
		10	35.159	23.337	12.243	8.518	5.595	6.290	4.232	1.443	0.379	0.000
	10	2	33.123	18.792	11.665	6.763	4.409	1.370	0.000	0.000	0.000	0.000
		5	38.560	21.261	12.782	9.381	6.534	5.401	3.136	0.286	0.000	0.000
		10	39.839	24.653	16.196	12.165	10.798	7.160	6.425	5.890	6.058	4.522
		Total	34.840	20.288	11.212	7.514	5.143	3.370	2.299	1.270	1.073	0.754

Table 3: ARPD and CPU time of the optimal solution for PFSP-PM, and ARPD for the evaluation of the optimal solution for PFSP.
* Only five instances have been solved optimally

solve PFSP-PM for the biggest sizes ($n \times m = 10 \times 10$).

- The similarity between both problems in specific cases (biggest sizes of T) seems to be clear. However, the decision about to apply or not approximate methods developed for PFSP to PFSP-PM cannot be verified only with the results obtained, since it depends on the distribution of the solutions for both problems. This issue has to be analysed in the following section.

5.2 Empirical distribution of the solutions

It is possible to determine the relative frequency (i.e. the estimated distribution) of the values of the objective function for all solutions of the problem by using complete enumeration. By analysing such estimated distribution, it is possible to infer the empirical hardness of the problem, i.e. the likelihood that one solution is close to the optimal solution. This analysis helps to determine which approximate methods should be appropriate to solve the PFSP-PM, since if there are many solutions close to the optimum, then simple methods can be applied to find good solutions, However, if the probability of finding a solution close to the optimum is very low, then sophisticated methods should be developed to efficiently solve the problem. Similar approaches have been conducted by Taillard (1990); Perez-Gonzalez and Framinan (2009); Fernandez-Viagas and Framinan (2015), or Dios et al. (2018).

More specifically, we try to determine the values of T providing similar empirical distributions for the PFSP and PFPS-PM problems. In this way, if such empirical distributions are similar, we may think about using approximate methods from the extensive literature of PFPS for the PFSP-PM. Otherwise, specific methods for the proposed problem should be developed.

To do so, the empirical distribution is plotted by evaluating the $n!$ possible schedules for a given set of instances. For each instance I, the worst makespan value C^w_{max} and the best makespan value C^*_{max} are computed, and the Relative Deviation Index for each makespan value obtained by evaluating each schedule S is computed using the following formula:

$$RDI(S) = \frac{C_{max}(S) - C^*_{max}}{C^w_{max} - C^*_{max}} \cdot 100 \tag{11}$$

Once the RDI is obtained for each schedule in an instance, the frequencies of the number of solutions with RDI in each interval $[k-1, k)$ with $k = 1, \ldots, 100$ are represented graphically, providing the empirical probability density function (pdf). Additionally, the cumulative frequencies are represented graphically as well, providing the empirical cumulative distribution function (cdf). As the RDI is normalized by $d = C^w_{max} - C^*_{max}$, a solution in the interval $[k-1, k)$ indicates that its makespan is less than $d \cdot k\%$ worse than the optimum. Therefore, the higher the frequencies in the right hand of the empirical pdf, the harder the problem is (as more solutions are far from the optimum).

The above procedure has been applied to all the instances of the set β_2 described in Section 4.1. Each instance has been solved using the different values of $T \in \{100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, M\}$. The empirical pdfs are plotted in Figure 3, and empirical cdfs in Figure 4, for all the instances in β_2 with sizes 5×10, 10×5 and 10×10 respectively. For clarity of presentation, only the results corresponding to $T = \{100, 200, 400, 600, 1000, M\}$ are plotted.

In Figure 3 and Figure 4, it can be seen that the empirical hardness for $T = 100$ (blue line)

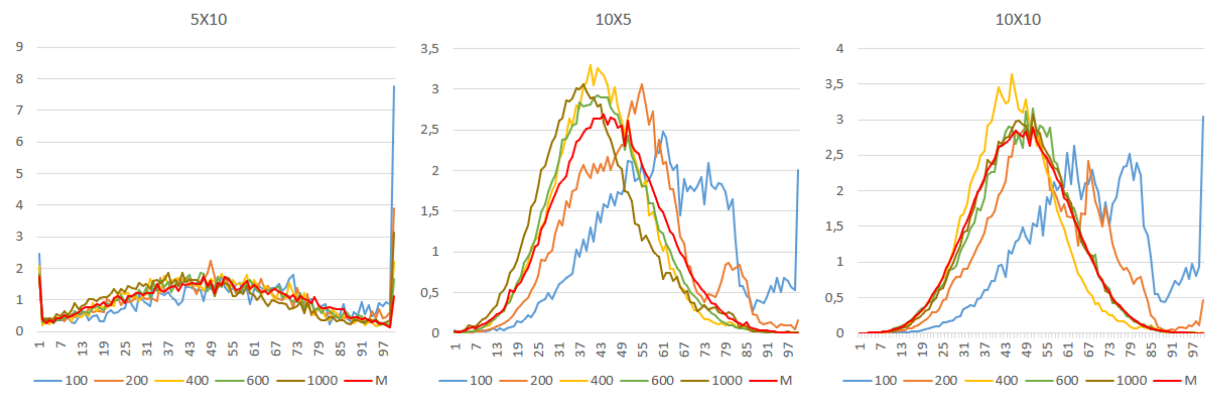

Figure 3: Empirical Density Function

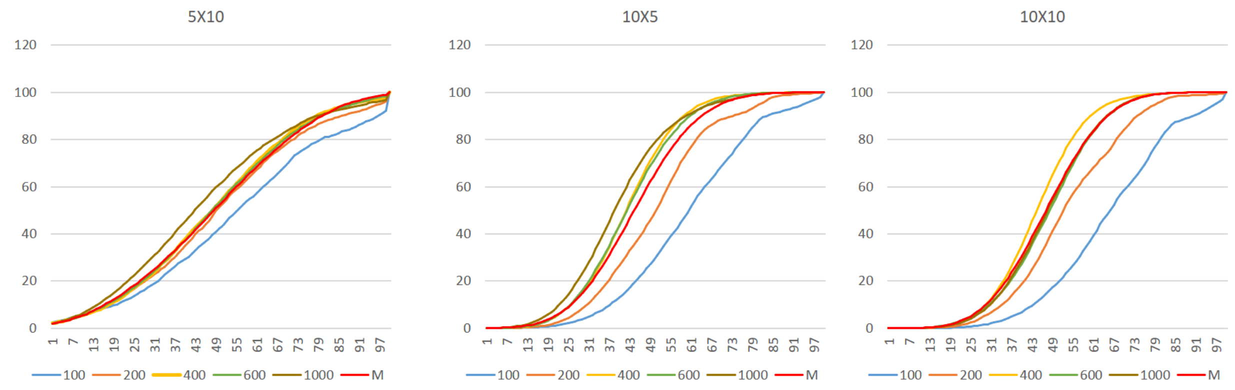

Figure 4: Empirical Cumulative Distribution Function

and 200 (orange line) are greater than for the PFSP ($T = M$, red line) for all sizes, presenting a peak in the frequencies of the worst schedules. The behaviour is more clear when the problem size increases. Therefore, $T = 100$ and $T = 200$ constitute the hardest cases. This hardness does not seem the same for the three sizes for each value of T. For example, when $n \neq m$, the problem with $T = 1000$ (brown line) is easier than PFSP, however, for the size 10×10, both have a similar pattern. Additionally, for 10×10, it can be observed that problems $T = 600$ (green line) and 1000 (brown line) have the same empirical hardness than $T = M$ (red line), being the case $T = 400$ (yellow line) empirically easier than $T = M$.

Taking into account the results in Subsection 5.1, and in this section, we can conclude that:

- From the empirical distribution, we see that the empirical hardness for $T = 100$ and $T = 200$ is greater than for the PFSP. In addition to the difficulty to find good solutions, the previous results show that the evaluation of the optimal solution for PFSP does not provide good solutions for the PFSP-PM in the same cases. Therefore, approximate algorithms developed for PSFP are not expected to provide good makespan values for PFSP-PM.

- For $T = 300$ and $T = 400$, the empirical distribution for PFSP-PM shows that the problem is easier than PFSP. However, the analysis carried out in Section 5.1 shows that the optimal solutions for PFSP are not good solutions for PFSP-PM.

- For $T > 400$, the empirical distribution for PSFP and PFSP-PM are very similar, and in some cases empirically easier. The evaluation of the optimal solutions provided for PSFP are less than 3% worse than the optimal makespan for PSFP-PM, so in these cases we expect that approximate solutions for PSFP should be good solutions for the PSFP-PM, and therefore, we can expect a good performance of the approximate methods developed for PSFP to be applied to PFSP-PM.

As a summary, according to the scope of our analysis, for bigger sizes instances, it is necessary to develop approximate methods for the problem PFSP-PM when T is tight as compared to the processing times (in the range from one to four times the maximum value of p_{ij}, i.e. $T \in \{100, 200, 300, 400\}$). Some efficient heuristics for these cases are presented in the next section.

6 Approximate procedures

In line with the conclusions presented in the previous section, it could be interesting to test the performance of approximate methods from the PFSP literature to PFSP-PM. In general, two different approaches can be followed:

1. Adapting the most efficient methods for PFSP to the PFSP-PM.

2. Developing specific methods for the PFSP-PM.

Regarding the adaptation of the most efficient methods for the PFSP, different constructive heuristics have been employed, being the NEH by Nawaz et al. (1982) the most used (Framinan et al., 2003; Kalczynski and Kamburowski, 2007, 2008; Fernandez-Viagas and Framinan, 2014). The NEH heuristic, starting from an initial sequence S_{ini}, constructs the best partial sequence S

formed by the two first jobs of S_{ini} according to the objective function, and iteratively insert the remaining jobs of S_{ini} in the best position in S. Depending on the procedure to construct S_{ini} and the tie-breaking mechanism (Tie method) in the selection of the best position during the insertion process, different versions of the NEH can be developed. Therefore, in order to adapt this heuristic to the PFSP-PM, we test different initial sequences based on different indicators and criteria as in Framinan et al. (2003). More specifically, 16 indicators have been combined with 8 criteria, providing 54 initial sequences. The best results are obtained by three of them without statistical differences among them. Since one of them is the one used in the original NEH (i.e jobs are arranged in non-ascending order of $p_j = \sum_{i=1}^{m} p_{ij}\ \forall j = 1, \ldots, n$), this criterion –denoted as $S_{ini} = \mathit{Sumdecr}$– has been selected for the subsequent experiments. With respect to the tie-breaking methods, the three most efficient are the original method used by the NEH (i.e. in case of ties, it selects the last evaluated position), the tie breaking method by Dong et al. (2008), and the tie breaking method proposed by Fernandez-Viagas and Framinan (2014). Note that the details of these tie breaking methods are omitted due to the lack of space. Combining these three tie-breaking methods with the best criterion for the initial solution, the following three algorithms are selected as adaptations:

- NEH_tie0: $S_{ini} = \mathit{Sumdecr}$ and selecting the last evaluated position in case of ties.
- NEH_tie1: $S_{ini} = \mathit{Sumdecr}$ and the tie breaking method by Dong et al. (2008).
- NEH_tie2: $S_{ini} = \mathit{Sumdecr}$ and the tie breaking method by Fernandez-Viagas and Framinan (2014).

Regarding the development of specific methods for the PFSP-PM, it seems sensible to retain a mechanism similar to that employed by the NEH to construct the solutions, giving its applicability to a wide range of scheduling problems and in order to maintain a similar complexity. However, differences far beyond a mere adaptation are required to incorporate specific knowledge form the PFSP-PM. More specifically, the following heuristics are proposed:

- Heuristic H1. This heuristic applies the NEH_tie2 starting from an initial solution specifically designed for the problem under consideration. More specifically, this initial solution is constructed as follows: The initial sequence is generated by sorting the jobs in non-descending order of the indicator χ_j:

$$\chi_j = \sum_{i=1}^{m} p_{ij} + \sum_{i=2}^{M} \frac{M \cdot \sum_{l=1}^{i-1} CT_{lj}}{i-1}$$

The indicator considers the features of job j in the system in the following way: The first term is the sum of the processing times of this job, giving priority to jobs with shorter sum of processing times. The second term is an estimation of the weighted idle times generated by job j. The weight $(\frac{M}{i-1})$ provides homogeneity to the values, since the estimation of the idle times are bigger when the index of the machine increases. The terms CT_{lj} are computed for each job j iteratively starting from $CT_{0j} = 0$ and $s_{0j} = 1$ as follows:

$$CT_{lj} = \begin{cases} CT_{l-1,j} + p_{lj} & \text{if } CT_{l-1,j} + p_{lj} \leq s_{l-1,j} \cdot T \\ s_{l,j} \cdot T + p_{lj} & \text{otherwise} \end{cases} \qquad l = 1, \ldots, M \tag{12}$$

and

$$s_{l,j} = \begin{cases} s_{l-1,j} & \text{if } CT_{l-1,j} + p_{lj} \leq s_{l-1,j} \cdot T \\ s_{l-1,j} + 1 & \text{otherwise} \end{cases} \qquad l = 1, \ldots, M \tag{13}$$

$s_{l,j}$ gives an approximation of the shift where job j would be processed on machine l, and CT_{lj} approximates the completion times of job j on machine l. In this manner, $\sum_{l=1}^{i-1} C_{lj}$ is an estimation of the idle time generated by job j in machine i if placed in the first position of the sequence.

- Heuristic H2. In the same way than H1, this heuristic applies the NEH_tie2 starting from an initial solution, in this case trying to schedule first the jobs with a higher number of operations fitting in a given shift. In this case, the initial solution is constructed as follows: the algorithm starts with all jobs unscheduled in the set U and an empty sequence S_{ini}. Iteratively, the job σ in U with more operations fitting into the last current shift (with respect to the first machine) of the partial schedule $S_{ini} \cup \{\sigma\}$ is removed from U and appended to S_{ini} (ties are broken according to the non ascending order of $p_j = \sum_{i=1}^{m} p_{ij}$). The pseudo-code is provided in Algorithm 1.

- Algorithm IT. This algorithm is based on the idea that, if a given number of jobs fits in a given shift (i.e. the sum of the processing times of all these jobs is lower than T), then the heuristic NEH can be applied to these jobs as in PFSP (i.e. T does not have influence on the partial sequence formed by these jobs). Therefore, a given initial sequence S_{ini} is divided in partial sequences, each one verifying that the sum of the processing times of all these jobs is lower than T. These partial sequences are merged one by one, and used as initial sequences of increasing sizes for the NEH, in order to fit the jobs in the best possible way (i.e. avoiding idle times). More specifically, the algorithm starts with all jobs unscheduled in the set U, given in the order of a initial sequence, S_{ini}, and an empty sequence S. Iteratively, each job σ in U is removed from U and appended to S while the sum of the processing times (sum) is lower than T. Then, NEH_tie2 is applied using S as initial sequence, and the resulting sequence replaces S (it contains the same jobs in different order). sum is initialized to zero, and the process is repeated, appending to S the next jobs from U that verify the condition that the sum of their processing times is smaller than T, applying again the NEH_tie2 to S. The pseudo-code is provided in Algorithm 2. Three versions of the IT algorithm are tested with the following initial sequences: S_{ini} given by the original NEH ($Sumdec$), denoted IT_NEH; the sequence provided by H1, denoted IT_H1; and finally, the sequence provided by H2, denoted IT_H2.

All methods have the same complexity than the original NEH, $O(n^3m)$ (taking into account that Taillard's accelerations cannot be implemented as explained in Observ. 7 in Section 3) except IT. This method executes iteratively the NEH procedure, a total of $K = \lceil \sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij}/T \rceil$ times, so its complexity is $O(Kn^3m)$.

7 Computational experience

In order to assess the performance of the heuristics presented in Section 6, they are compared using the testbed for big-size instances developed in Section 4.2. More specifically, the following heuristics have been compared:

- Heuristics adapted from the PFSP, i.e. NEH_tie0, NEH_tie1 and NEH_tie2.

Algorithm 1 Heuristic H2

```
1: procedure H2($n$, $m$, $T$, $p_{ij}$)
2:     $Sumdecr$ = Initial sequence by original NEH (non-increasing order of $p_j = \sum_{i=1}^{m} p_{ij}$);
3:     $U = \{Sumdecr[1], \ldots, Sumdecr[n]\}$ set of unscheduled jobs in the sequence order.
4:     $k = 1$; (Initialization of the position being occupied of $S_{ini}$)
5:     $S_{ini}[1] = U[1]$;
6:     Remove $U[1]$ from $U$ and $dimU = n - 1$;
7:     while $k < n$ do
8:         for $i = 1, \ldots, m$ do
9:             $CT_i$ completion time of machine $i$ by the subsequence $S_{ini}$
10:        end for
11:        $k + +$; (new position of $S_{ini}$ to be occupied)
12:        $s = CT_1/T + 1$; (number of occupied shifts in the first machine by the actual $S_{ini}$)
13:        $bestindexU = -1$; (Initialization of the best job from U to be inserted in $S_{ini}$)
14:        $numbmach = -1$; (Initialization of the count of number of machines occupied by
    a job within the shift $s$)
15:        for $j = 1, \ldots, dimU$ do
16:            if $s \cdot T - CT_1 > p_{1U[j]}$ then
17:                $i = 1$; (the first operation fits in the shift $s$ in the first machine)
18:                while $p_{iU[j]} < s \cdot T - \max\{CT_{i-1} + p_{i-1U[j]}, CT_i\}$ and $i < M + 1$ do
19:                    i++; (the operation fits in the shift in the next machine)
20:                end while
21:                if $i > numbmach$ then
22:                    $numbmach = i$ and $bestindexU = j$;
23:                end if
24:            end if
25:            if $bestindexU == -1$ then
26:                $S_{ini}[k] = U[1]$;
27:                Remove $U[1]$ from $U$;
28:            else
29:                $S_{ini}[k] = U[bestindexU]$;
30:                Remove $U[bestindexU]$ from $U$;
31:            end if
32:        end for
33:    end while
34:    $S$= NEH_tie2($S_{ini}$,$n$); (NEH with tie breaking using $S_{ini}$ with dimension $n$)
       return $C_{max}(S)$
35: end procedure
```

Algorithm 2 Algorithm IT

```
1: procedure IT($n$, $m$, $T$, $p_{ij}$, $S_{ini}$)
2:     $S = \varnothing$, $sum = 0$ and $j = 1$;
3:     while $j < n + 1$ do
4:         while $sum < T$ and $j < n + 1$ do
5:             for $i = 1, \ldots, m$ do
6:                 $sum = +p_{iS_{ini}[j]}$;
7:             end for
8:             j++;
9:             Insert $S_{ini}[j]$ in the last position of $S$
10:        end while
11:        $S$= NEH_tie2($S$, $j$); (NEH with tie breaking using $S$ with dimension $j$ as initial sequence)
12:        $sum = 0$; (Initialization of $sum$)
13:    end while
       return $C_{max}(S)$
14: end procedure
```

- New heuristics developed for the PFSP-PM, i.e. H1, H2 and IT in its three versions: IT_NEH, IT_H1 and IT_H2.

In addition to compare their relative performance, the contribution of each heuristic can be also assessed by comparing these results with the results obtained by its application to the PFSP and the evaluation of the so-obtained schedule for the PFSP-PM. By doing so, it is possible to discern whether the results obtained by each method are due to its suitability for the general PFSP, or not. Therefore, in addition to provide the results for the heuristics, we also provide the results obtained by their application to the PFSP and the subsequent evaluation of the schedule for the PFSP-PM. These results are denoted by the corresponding heuristic with an asterisk, only for the heuristics adapted from the PFSP (note that this is not possible for H1, H2 and IT, as they explicitly require an availability period). As a consequence, the results of the heuristics NEH_tie0*, NEH_tie1* and NEH_tie2* are also presented.

We apply each heuristic $H \in$ {NEH_tie0, NEH_tie1, NEH_tie2, NEH_tie0*, NEH_tie1*, NEH_tie2*, H1, H2, IT_NEH, IT_H1, IT_H2} to an instance I and compute its RPD^H as follows:

$$RPD^H = \frac{C_{max} - C_{max}^{MIN}}{C_{max}^{MIN}} \tag{14}$$

where C_{max} is the makespan value of the schedule provided by heuristic H applied to instance I, and C_{max}^{MIN} the minimum value obtained for instance I among all heuristics. All Taillard instances have been solved using different values of T as previously explained in Section 4. On the one hand, and due to the results obtained in Subsections 5.1 and 5.2, values $T \in \{100, 200, 300, 400\}$ have been used in the case with constant T. On the other hand, instances have been solved with variable values of T, depending on parameters r and t. The results are discussed in the next subsections.

7.1 Results with constant T

Table 4 shows the Average RPD values for each problem size, and the total in the last row, labelled as Total. The best values obtained are marked in bold. It can be seen that, as expected, the methods applied to the PFSP-PM usually provide better makespan values than the sequence provided for the classical problem evaluated for the PFSP-PM (cases with *) when T is 100 and 200. In general, NEH_tie0*, NEH_tie1* and NEH_tie2* provide better ARPD values when T is 300 and 400. These results may be explained by the fact that, in these cases, this problem is more similar to (and easier than) the PFSP than for $T = 100$ or $T = 200$ (see Subsections 5.1 and Subsection 5.2). Additionally, it can be seen that the three heuristics developed specifically for the problem provide better results than the adaptations of heuristics from the PFSP, particularly those provided by the algorithm IT (note that the computational cost of this method is higher), being IT_H2 the best method with total ARDP of 2.01.

Figure 5 shows the 95% confidence intervals, on the left for all methods aggregated for all values of T, and on the right for each value of T. On the left of the figure, it can be seen that IT_H2 is the best method, with statistically significant differences with the rest of the methods, except with IT_NEH. Among the adapted heuristics, the performance of the NEH_tie2 does not have statistical differences with H1 and IT_H1, but H2, IT_NEH and IT_H2 are better with statistical differences. Additionally, on the right, this method has a good performance for all values of T. Similar to the results in Table 4, it can be observed that the methods NEH_tie0*, NEH_tie1* and NEH_tie2* improve as T increases, without statistical differences for $T = 300$ (red) with respect to NEH_tie0, NEH_tie1 and NEH_tie2 respectively, and even better for $T = 400$ (yellow). NEH_tie2 has a good performance for $T = 100$ (blue), but the differences are not statistically significant with the results

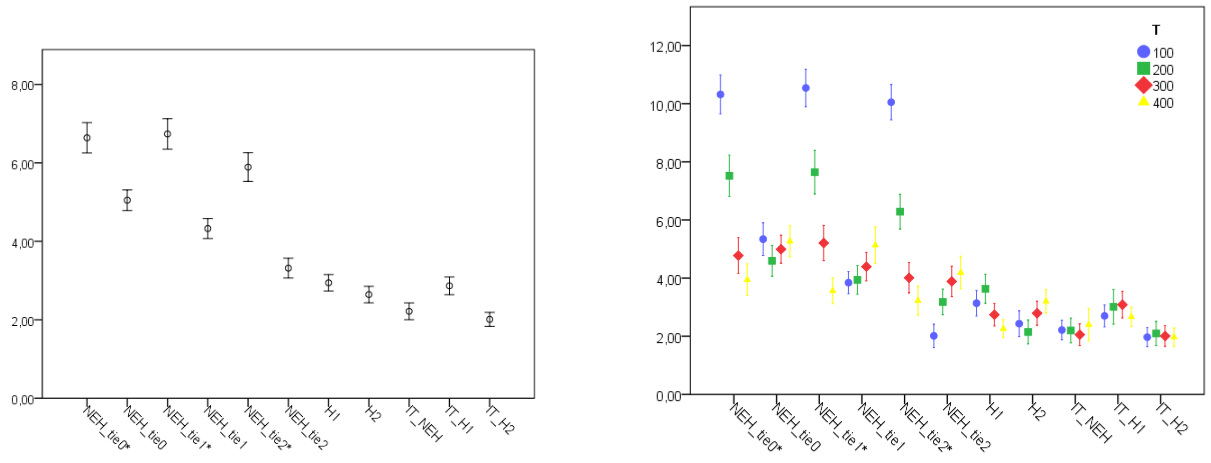

Figure 5: 95% Confidence intervals for constant T (Left: Total ARPD, Right: ARPD for each value of T)

obtained by H2, IT_NEH or IT_H2.

7.2 Results for variable T

Table 5 shows the results in the same format as in Table 4. The results for the different scenarios defined in Section 4.2 are presented. Therefore, for scenario HoT, the values of T are generated using $t = 0.2$ and $r = 0.6$, for scenario HoL using $t = 0.6$ and $r = 0.6$, for scenario HeT using $t = 0.2$ and $r = 1.2$, and for scenario HeL with $t = 0.4$ $r = 1.8$. In bold we indicate the best value obtained for each problem size and scenario. Table 5 shows that, IT_H2 is the best method according to the Total ARPD, with 1.689.

Figure 6 shows similar results than Figure 5. In the left, it can be observed that IT_H2 is the best method. In the right side it can be seen that the scenario homogeneously tight (blue) presents a bad performance for the methods NEH_tie0*, NEH_tie1* and NEH_tie2*, and the opposite happens for NEH_tie0, NEH_tie1 and NEH_tie2, with all methods providing more similar results in the scenario heterogeneously loose (orange). It can be observed that the scenario heterogeneously tight (red) and homogeneously loose (green) do not present statistical differences for all methods. Finally, the good performance of IT_H2 for all scenarios can be seen.

n	m	T	NEH_tie0*	NEH_tie0	NEH_tie1*	NEH_tie1	NEH_tie2*	NEH_tie2	H1	H2	IT_NEH	IT_H1	IT_H2
20	5	100	13.702	9.577	12.894	3.950	13.564	5.200	4.681	4.739	4.259	4.541	**3.161**
		200	13.215	8.475	12.236	4.478	9.011	6.981	5.095	4.363	4.201	7.445	**2.620**
		300	7.391	8.655	8.820	6.367	4.737	8.081	4.050	4.664	**2.416**	5.102	4.697
		400	7.601	8.972	6.046	8.005	4.910	8.587	2.638	5.364	5.756	**2.613**	3.443
	10	100	9.698	6.074	11.194	6.295	10.860	5.041	3.741	5.213	**3.160**	4.306	4.144
		200	8.700	6.903	10.753	6.760	7.590	5.220	6.349	**3.619**	4.427	5.827	5.210
		300	7.016	7.806	9.229	5.749	7.117	5.672	5.011	5.842	4.809	6.194	**4.266**
		400	5.607	7.867	5.388	8.483	5.105	6.040	**3.071**	4.919	4.678	3.137	4.409
	20	100	7.771	5.053	8.193	5.368	7.643	2.134	4.095	3.803	2.838	3.896	**1.247**
		200	6.835	5.097	7.038	5.633	6.221	**2.282**	6.000	2.506	3.114	6.128	3.480
		300	4.728	4.910	4.487	5.089	4.311	3.825	3.274	2.905	3.343	5.131	**2.732**
		400	4.395	4.134	3.885	5.972	2.984	2.874	2.811	4.123	2.658	3.773	**1.326**
50	5	100	14.217	2.865	14.750	2.142	12.253	2.470	5.416	2.339	**1.127**	3.467	2.514
		200	11.138	3.720	12.149	4.587	9.037	2.303	5.071	**1.508**	2.465	2.073	2.580
		300	7.590	3.856	6.642	3.747	5.924	2.190	3.860	1.913	**0.970**	2.523	1.786
		400	5.372	3.962	4.403	2.590	5.439	2.828	2.752	2.980	**1.971**	2.712	2.145
	10	100	9.761	5.268	10.153	3.508	9.307	2.068	3.924	2.599	**1.563**	2.570	2.154
		200	8.408	6.401	9.491	4.717	6.904	4.279	3.943	3.154	2.955	3.834	**2.479**
		300	5.296	5.540	5.121	5.584	4.108	5.490	3.188	3.466	**2.095**	2.723	2.117
		400	3.046	6.183	2.397	5.190	2.241	4.068	2.045	3.662	**1.723**	3.817	2.205
	20	100	8.149	4.680	8.670	3.666	8.392	2.547	2.873	**1.879**	3.398	2.874	2.249
		200	5.600	5.213	5.132	5.535	4.285	3.240	3.247	3.469	**1.262**	1.680	1.428
		300	3.733	5.452	4.782	6.907	3.475	4.545	2.797	3.411	2.570	3.457	**2.314**
		400	2.731	7.126	3.974	7.178	2.759	4.675	2.513	3.442	1.811	3.020	**1.743**
100	5	100	15.774	3.959	14.960	2.865	14.632	**0.600**	4.591	1.556	2.906	2.498	1.798
		200	11.204	1.590	10.520	1.801	10.674	1.892	5.102	**0.815**	2.218	2.329	0.986
		300	7.723	3.042	8.257	1.851	7.462	1.870	3.972	1.163	1.150	1.991	**0.756**
		400	7.391	2.703	5.756	1.867	6.750	2.837	3.358	**1.336**	1.594	1.794	1.396
	10	100	10.793	5.941	10.244	3.946	10.715	**1.275**	2.334	1.863	1.629	2.431	1.833
		200	7.088	4.301	6.047	3.233	6.762	3.068	2.812	**1.430**	1.540	2.207	2.207
		300	4.251	5.077	5.000	3.787	3.616	3.894	1.931	2.125	1.588	2.682	**1.126**
		400	4.416	4.674	4.352	5.359	3.576	3.515	2.104	2.161	2.377	2.879	**1.367**
	20	100	7.203	4.030	7.925	3.960	6.815	**1.354**	2.099	1.752	1.719	2.156	1.580
		200	4.575	4.065	4.899	4.187	3.818	3.010	2.490	1.435	**1.121**	1.862	1.225
		300	2.599	5.497	3.299	3.683	1.767	2.897	1.812	2.513	1.979	2.900	**1.208**
		400	1.811	5.064	1.692	4.289	**0.695**	3.623	1.669	3.026	2.159	1.800	1.480
200	10	100	11.381	4.866	11.567	2.916	11.198	**0.512**	1.352	0.594	1.624	1.176	1.241
		200	6.531	2.348	7.045	1.658	5.410	1.419	1.594	**0.622**	1.595	1.137	1.162
		300	4.564	2.821	4.352	2.702	3.758	2.561	1.241	1.362	1.174	1.856	**0.709**
		400	3.420	3.736	2.800	3.971	3.490	3.128	1.342	1.611	**0.837**	1.964	0.878
	20	100	7.547	5.805	7.995	3.968	7.252	**0.778**	1.589	1.702	1.356	1.606	1.017
		200	3.494	3.715	3.076	2.516	2.687	2.429	1.004	1.525	1.002	**0.781**	1.195
		300	1.325	3.984	1.246	4.245	**1.029**	3.052	1.327	2.238	1.447	1.464	1.347
		400	0.901	4.812	1.453	4.720	**0.418**	4.514	1.952	2.919	1.869	2.937	1.816
500	20	100	7.816	5.971	7.951	3.532	7.976	**0.180**	0.941	1.151	0.993	0.862	0.692
		200	3.462	3.309	3.326	2.155	3.037	2.024	0.871	1.316	**0.486**	0.848	0.577
		300	1.100	3.267	1.254	3.023	0.803	2.555	**0.433**	1.902	1.111	0.998	1.045
		400	0.625	3.959	0.590	3.946	**0.329**	3.559	0.867	2.849	1.329	1.555	1.411
	Total		6.639	5.048	6.738	4.327	5.893	3.316	2.942	2.643	2.216	2.866	**2.010**

Table 4: ARPD for constant T

n	m	Scenario	NEH_tie0*	NEH_tie0	NEH_tie1*	NEH_tie1	NEH_tie2*	NEH_tie2	H1	H2	IT_NEH	IT_H1	IT_H2
20	5	HoT	14.858	6.412	14.162	6.523	16.758	4.712	5.619	4.693	2.569	4.657	**2.198**
		HeT	9.256	8.594	12.199	5.820	9.776	7.222	5.246	6.532	5.184	5.881	**3.015**
		HoL	11.469	10.255	13.807	9.006	8.222	7.440	5.375	4.628	4.911	5.297	**2.112**
		HeL	8.284	8.443	9.552	6.579	8.821	5.515	**3.298**	3.595	6.070	6.241	3.540
	10	HoT	14.319	5.127	13.140	5.183	11.820	3.725	6.331	2.174	**1.773**	4.612	2.533
		HeT	10.394	6.424	7.515	7.176	9.664	5.761	6.178	5.713	3.233	**3.019**	3.719
		HoL	10.043	7.320	10.535	5.832	8.417	5.408	5.097	3.848	4.008	5.304	**2.994**
		HeL	6.586	8.752	7.462	7.184	5.819	7.281	4.114	3.978	6.311	3.094	**2.385**
	20	HoT	9.952	3.983	11.533	5.057	9.428	4.726	6.033	3.566	3.595	4.270	**1.096**
		HeT	6.071	2.374	7.574	3.241	6.097	2.984	4.453	**2.020**	3.020	4.316	2.919
		HoL	8.013	4.735	8.520	4.833	6.428	3.842	4.125	2.373	**2.240**	4.388	3.040
		HeL	5.894	6.227	6.377	6.265	5.210	5.291	2.747	4.713	**2.162**	4.529	3.093
50	5	HoT	17.235	4.476	19.194	4.076	15.637	3.514	7.653	4.140	2.006	4.052	**1.586**
		HeT	13.928	4.873	14.178	3.920	13.314	3.082	5.144	2.544	**1.527**	3.986	1.559
		HoL	12.371	3.725	11.294	4.147	10.972	3.260	6.435	1.983	**1.655**	3.552	1.669
		HeL	8.954	3.886	9.571	4.184	8.676	2.705	4.386	2.894	2.078	3.855	**1.563**
	10	HoT	11.790	4.955	10.725	4.444	9.230	5.117	3.868	3.371	**1.520**	3.319	1.670
		HeT	9.685	4.616	9.762	3.276	8.502	4.304	4.103	3.451	**2.180**	3.306	2.962
		HoL	7.403	4.546	7.844	5.516	7.569	3.377	2.911	2.211	2.682	3.223	**1.936**
		HeL	5.207	5.388	4.990	4.703	6.289	4.204	2.743	3.240	2.790	2.744	**1.311**
	20	HoT	7.778	4.178	10.789	2.167	8.478	**1.216**	3.696	2.231	1.811	3.859	2.192
		HeT	7.434	5.442	9.564	5.065	8.302	2.233	2.810	3.129	1.723	3.206	**0.419**
		HoL	6.171	5.757	7.202	4.627	5.130	4.479	2.600	3.854	**1.950**	2.544	1.971
		HeL	2.946	6.336	3.696	5.131	3.263	3.304	2.611	3.278	2.024	2.570	**1.758**
100	5	HoT	15.409	2.176	15.617	2.315	13.499	1.886	5.824	**0.854**	1.249	1.654	1.051
		HeT	13.916	2.656	14.198	2.139	12.464	1.649	5.259	**0.856**	2.415	2.314	1.939
		HoL	11.211	2.304	10.404	1.047	11.009	1.005	4.817	**0.584**	1.911	3.328	2.046
		HeL	12.329	2.225	10.713	1.782	9.643	1.615	4.248	1.539	1.737	1.405	**0.946**
	10	HoT	13.693	3.280	12.293	2.050	10.276	1.536	3.565	**0.791**	1.727	2.057	2.214
		HeT	8.495	3.600	8.511	2.360	9.441	2.145	1.827	1.135	**0.692**	2.051	0.796
		HoL	7.042	4.282	6.880	3.550	6.647	2.133	2.185	1.817	1.295	2.006	**1.000**
		HeL	5.036	4.345	4.794	4.310	4.348	3.576	2.672	1.878	1.489	2.968	**0.866**
	20	HoT	6.168	4.134	8.102	2.895	6.961	1.990	2.175	1.421	**0.878**	1.527	1.063
		HeT	4.200	3.614	4.683	3.617	4.075	2.588	2.465	1.719	**0.486**	2.460	0.937
		HoL	4.636	3.962	6.151	3.119	4.931	2.894	2.738	2.314	**0.686**	2.399	1.531
		HeL	3.213	4.006	3.270	3.551	3.140	2.985	1.947	1.099	1.018	1.224	**0.843**
200	10	HoT	10.028	2.988	9.956	0.836	9.112	1.064	2.588	**0.721**	0.930	1.111	0.848
		HeT	6.467	2.191	6.678	1.481	5.880	1.399	1.661	**0.580**	1.320	1.429	0.939
		HoL	7.088	2.247	7.404	1.922	6.644	1.356	0.990	**0.730**	0.972	1.845	0.750
		HeL	5.356	2.194	4.722	1.966	4.652	1.832	1.423	0.848	0.997	1.197	**0.234**
	20	HoT	7.733	4.067	8.951	2.211	7.051	1.100	2.043	1.118	1.440	1.331	**0.946**
		HeT	2.875	3.381	3.205	3.042	2.383	2.411	1.408	1.841	**0.377**	1.843	0.645
		HoL	2.971	3.177	3.902	2.478	2.865	2.813	0.760	2.163	0.992	1.148	**0.709**
		HeL	1.971	3.626	2.474	3.102	2.241	2.392	0.960	1.729	**0.883**	1.409	0.892
500	20	HoT	5.776	3.432	5.971	1.724	5.570	1.164	0.903	1.146	0.744	0.586	**0.391**
		HeT	3.111	2.904	2.940	1.957	3.055	1.596	0.527	1.080	0.519	0.502	**0.361**
		HoL	4.192	3.384	4.124	2.080	4.174	1.736	0.581	1.470	0.513	0.415	**0.306**
		HeL	1.678	3.423	1.671	2.922	1.218	2.677	**0.270**	2.056	0.777	1.052	1.063
	Total		7.263	4.407	7.516	4.019	6.678	3.269	3.172	2.463	1.936	2.794	**1.689**

Table 5: ARPD for variable T

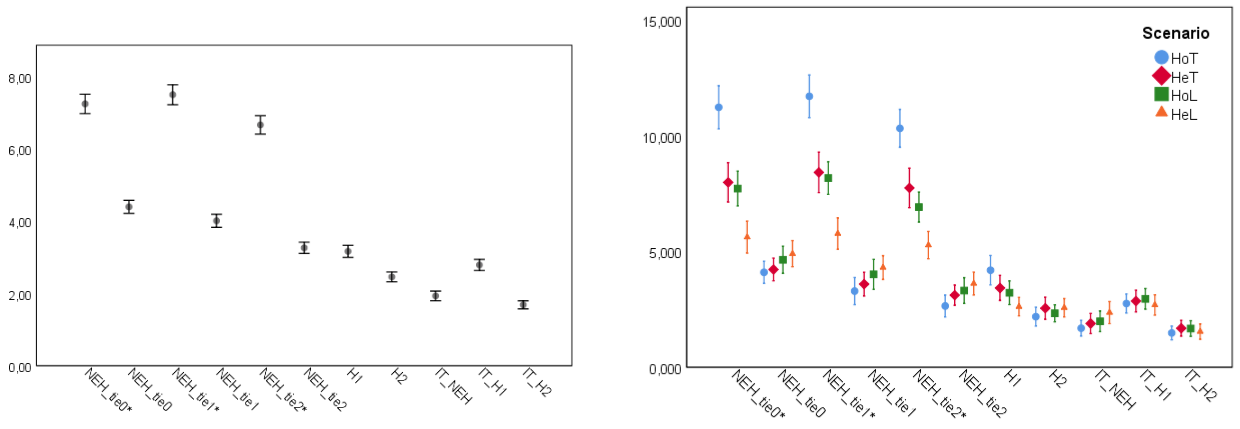

Figure 6: 95% Confidence intervals for variable T (Left: Total ARPD, Right: ARPD for each scenario)

8 Conclusions

This paper considers a permutation flowshop scheduling problem inspired in the scenario of a manufacturing company that stops all the machines during the changes of shifts, implying that machines are not available periodically, and the operations cannot be preempted. In the literature, this machine availability constraint has been denoted as periodic maintenance for other scheduling layouts, such as the single machine or parallel machines. To the best of our knowledge, this case has not been previously considered for the permutation flowshop layout. Depending on the hypotheses regarding the preemption of operations/jobs, different scheduling problems can be defined. In this paper, the problem studied considers the non-resumable preemption of operations, which is denoted as $Fm|prmu, nr - pm|C_{max}$.

For this problem, an in-depth analysis has been carried out to characterise the space of solutions and the differences with the classical PFSP, for different values of the availability period, T, in different small-size sets of instances. The results show that, on the one hand, for bigger values of T ($T > 400$), the optimal solutions for the PFSP and the PFSP-PM are very similar, as expected, as well as their empirical hardness, so for these values, approximate methods from the literature of the classical problem may provide good solutions for the problem under consideration. On the other hand, for smaller values of T ($T \leq 400$), the optimal solutions for both problems are very different, and also their empirical hardness (PFSP-PM is easier for $T = 300$ or 400 and harder for $T = 100$ and 200). Therefore, good solutions for the classical problem are not necessarily good for the proposed problem. For these cases, we propose some constructive heuristics, both by adapting existing heuristics from the PFSP, and also by developing new specific heuristics for the problem. The computational results using the Taillard testbed show that, for all the tested values of T, the specific heuristics provide good results with lower variability than the adapted heuristics.

As future research lines, new objective functions could be analysed for this constraint in the permutation flowshop, such as the total completion time and the total tardiness, in order to determine if, for these objectives, a pattern similar to the one observed for the makespan is detected. If this is the case, it would be interesting to develop approximate methods for these problems.

References

Allaoui, H., Artiba, A., Elmaghraby, S., and Riane, F. (2006). Scheduling of a two-machine flowshop with availability constraints on the first machine. *International Journal of Production Economics*, 99(1-2):16–27.

Angel-Bello, F., Alvarez, A., Pacheco, J., and Martínez, I. (2011). A heuristic approach for a scheduling problem with periodic maintenance and sequence-dependent setup times. *Computers & Mathematics with Applications*, 61(4):797–808.

Armentano, V. A. and Ronconi, D. P. (1999). Tabu search for total tardiness minimization in flowshop scheduling problems. *Computers & Operations Research*, 26(3):219–235.

Dios, M., Fernandez-Viagas, V., and Framinan, J. M. (2018). Efficient heuristics for the hybrid flow shop scheduling problem with missing operations. *Computers and Industrial Engineering*, 115.

Dong, X., Huang, H., and Chen, P. (2008). An improved NEH-based heuristic for the permutation flowshop problem. *Computers & Operations Research*, 35(12):3962–3968.

Fernandez-Viagas, V. and Framinan, J. M. (2014). On insertion tie-breaking rules in heuristics for the permutation flowshop scheduling problem. *Computers and Operations Research*, 45:60–67.

Fernandez-Viagas, V. and Framinan, J. M. (2015). NEH-based heuristics for the permutation flowshop scheduling problem to minimise total tardiness. *Computers & Operations Research*, 60:27–36.

Fernandez-Viagas, V. and Framinan, J. M. (2017). Reduction of permutation flowshop problems to single machine problems using machine dominance relations. *Computers & Operations Research*, 77:96–110.

Fernandez-Viagas, V., Ruiz, R., and Framinan, J. M. (2017). A new vision of approximate methods for the permutation flowshop to minimise makespan: State-of-the-art and computational evaluation. *European Journal of Operational Research*, 257(3):707–721.

Framinan, J. M., Leisten, R., and Rajendran, C. (2003). Different initial sequences for the heuristic of Nawaz, Enscore and Ham to minimize makespan, idletime or flowtime in the static permutation flowshop sequencing problem. *International Journal of Production Research*, 41(1):121–148.

Framinan, J. M., Leisten, R., and Ruiz, R. (2014). *Manufacturing scheduling systems: An integrated view on models, methods and tools*, volume 9781447162. Springer-Verlag London Ltd.

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

Gurobi Optimization, L. (2018). Gurobi optimizer reference manual.

Hsu, C.-J., Low, C., and Su, C.-T. (2010). A single-machine scheduling problem with maintenance activities to minimize makespan. *Applied Mathematics and Computation*, 215(11):3929–3935.

Kaabi, J. and Harrath, Y. (2019). Scheduling on uniform parallel machines with periodic unavailability constraints. *International Journal of Production Research*, 57(1):216–227.

Kalczynski, P. J. and Kamburowski, J. (2007). On the NEH heuristic for minimizing the makespan in permutation flow shops. *Omega*, 35(1):53–60.

Kalczynski, P. J. and Kamburowski, J. (2008). An improved NEH heuristic to minimize makespan in permutation flow shops. *Computers & Operations Research*, 35(9):3001–3008.

Labidi, M., Kooli, A., Ladhari, T., Gharbi, A., and Suryahatmaja, U. S. (2018). A Computational Study of the Two-Machine No-Wait Flow Shop Scheduling Problem Subject to Unequal Release Dates and Non-Availability Constraints. *IEEE Access*, 6:16294–16304.

Lee, C.-Y. (1997). Current trends in deterministic scheduling. *Annals of Operations Research*, 70:1–41.

Low, C., Hsu, C.-J., and Su, C.-T. (2010). A modified particle swarm optimization algorithm for a single-machine scheduling problem with periodic maintenance. *Expert Systems with Applications*, 37(9):6429–6434.

Ma, Y., Chu, C., and Zuo, C. (2009). A survey of scheduling with deterministic machine availability constraints. *Computers & Industrial Engineering*, 58(2):199–211.

Nawaz, M., Enscore, E., and Ham, I. (1982). A Heuristic Algorithm for the m-Machine , n-Job Flow-shop Sequencing Problem. *Omega*, 11(1):91–95.

Perez-Gonzalez, P. and Framinan, J. M. (2009). Scheduling permutation flowshops with initial availability constraint: Analysis of solutions and constructive heuristics. *Computers & Operations Research*, 36(10):2866–2876.

Perez-Gonzalez, P. and Framinan, J. M. (2018). Single machine scheduling with periodic machine availability. *Computers & Industrial Engineering*, 123:180–188.

Pinedo, M. L. (2008). *Scheduling: Theory, Algorithms and Systems.* Springer Berlin / Heidelberg, third edition.

Potts, C. N. and Van Wassenhove, L. N. (1982). A decomposition algorithm for the single machine total tardiness problem. *Operations Research Letters*, 1(5):177–181.

Stafford, E. F. J., Tseng, F. T., and Gupta, J. N. (2005). Comparative evaluation of MILP flowshop models. *Journal of the Operational Research Society*, 56(1):88–101.

Taillard, E. D. (1990). Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research*, 47:65–74.

Taillard, E. D. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64:278–285.

Vallada, E., Ruiz, R., and Framinan, J. M. (2015). New hard benchmark for flowshop scheduling problems minimising makespan. *European Journal of Operational Research*, 240(3):666–677.

Xu, Z., Xu, D., He, J., Wang, Q., Liu, A., and Xiao, J. (2018). Mixed Integer Programming Formulations for Two-Machine Flow Shop Scheduling with an Availability Constraint. *Arabian Journal for Science and Engineering*, 43(2):777–788.

Yazdani, M., Jolai, F., Taleghani, M., and Yazdani, R. (2018). A modified imperialist competitive algorithm for a two-agent single-machine scheduling under periodic maintenance consideration. *International Journal of Operational Research*, 32(2):127.

Yu, X., Zhang, Y., and Steiner, G. (2014). Single-machine scheduling with periodic maintenance to minimize makespan revisited. *Journal of Scheduling*, 17(3):263–270.