Single Machine Scheduling with Periodic Machine Availability *

Paz Perez-Gonzalez[1], Jose M. Framinan[1]

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

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

Abstract

In this paper we address the problem of scheduling jobs on a single machine with cyclical machine availability periods. In this problem, the scheduling horizon is composed of periods where the machine is available followed by other periods where no operation can be performed. In the literature, the problem is denoted as scheduling with periodic maintenance, as it is usually assumed that these unavailability periods are employed to perform maintenance activities. Another situation is the one inspiring our research, i.e. the need of completing manufacturing operations within a shift. More specifically, we focus the single machine scheduling problem with makespan objective subject to periodic machine availability. There are several contributions proposing approximate procedures due to the NP-hardness shown for the problem. However, we are not aware of a computational evaluation among these procedures. Furthermore, the problem is similar to the classical bin packing problem, so it is of interest to explore the relation between both problems. In this paper, we address these two issues, and propose new approximate solution procedures for the problem.

Keywords: Single machine scheduling, makespan, Heuristics, periodic maintenance, periodic machine availability

*This is an Accepted Manuscript of an article published by Elsevier in Computers and Industrial Engineering Volume 123, September 2018, Pages 180-188, available online: http://dx.doi.org/10.1016/j.cie.2018.06.025

1 Introduction

In many manufacturing companies, operations are not performed continuously over the scheduling horizon, since resources stop cyclically due to the end of the shifts, hours/days off (e.g. nights or weekends), periodic maintenance activities, etc. Regardless the cause of the unavailability of the resources, this problem is known in the literature as *scheduling with periodic maintenance* (Low et al., 2010a; Yu et al., 2014). In our case, the research is inspired by a manufacturing process within the aerospace industry, more specifically the assembly of wiring harness. This rather complex operation is performed manually by specialized teams, and it is not desirable that a task started by one team is completed by another, as this usually decreases the efficiency and quality of the task. Therefore, schedules are built so all tasks starting within a working shift must be also completed in this shift. Our objective is to minimise the maximum completion time across all jobs, or makespan. Note that this problem is NP-hard in the strong sense when cyclical availability periods are included (Lee, 1996), even if its classical counterpart is trivial.

For this problem, several approximate solution procedures, many of them based on bin packing methods, have been presented in the literature: Ji et al. (2007); Hsu et al. (2010); Yu et al. (2014) develop several constructive heuristics and Low et al. (2010a) propose a metaheuristic. However, it seems that these procedures have been developed in an independent manner, and consequently no computational analysis has been carried out to compare them. Therefore, their relative efficiency has not been established and the state of the art remains unclear. Furthermore, the problem is clearly related to the well-known bin packing problem, however, such relationship has not been analysed in the literature. Our paper is aimed towards filling these gaps: First we explore the relations between the bin packing problem and our problem by comparing the solutions provided by mixed integer linear programming models for both problems. Second, we carry out a computational experimentation among the heuristics proposed in the literature for the problem under consideration. Based on the excellent results obtained by two fast constructive heuristics, we include them in a local search procedure and propose two new heuristics. The computational experience carried out shows that our proposals outperform the constructive heuristics and the metaheuristic existing for the problem while requiring less CPU time.

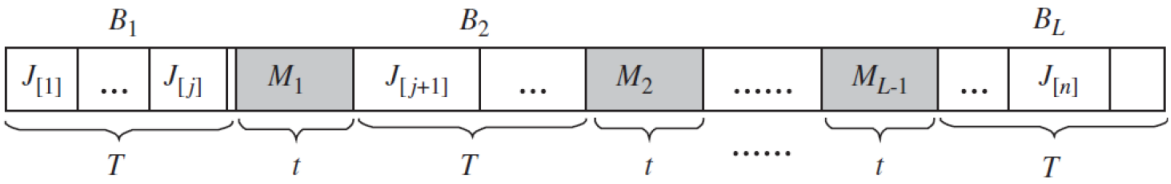

Figure 1: Proposed problem Ji et al. (2007)

The rest of the paper is organised as follows. Section 2 includes the notation for the problem and the state-of-the-art. Section 3 describes the mathematical models employed to compare the bin packing problem and the proposed problem. In Section 4 we review and classify the constructive heuristics identified in the literature, and in Section 5 we describe the new heuristics proposed. In Section 6 we discuss the set of instances employed to carry out the computational experience, and present the results obtained. Finally, in Section 7 the main conclusions are presented, along with future research lines.

2 Problem statement and background

The decision problem consists on scheduling n jobs, $J_1, \ldots, J_n$ on a single machine with processing times p_j, $j = 1, \ldots, n$. The machine is not available at any time in all the scheduling horizon, which is composed of availability periods of T time units followed by a non-availability period of t time units (see Figure 1).

In the literature, the availability periods are denoted batches, bins or blocks (we will use this last term in the paper). Jobs have non-resumable operations, so it has to be assumed that $T \geq p_j \ \forall j = 1, \ldots, n$ in order to guarantee the feasibility of the problem. Additionally, we consider makespan minimization, so this problem is denoted $1|nr - pm|C_{max}$ by Ji et al. (2007) according to the standard notation. 1 means that we are considering a single machine layout, $nr - pm$ indicates that periodical maintenance (pm) unavailability periods are considered as constraint, and jobs are non-resumable (nr), that means, jobs must not be interrupted (or if it, it must be completely processed again). Finally C_{max} indicates that the minimization objective is the completion time of the last job in the machine.

For a given schedule, we denote the *slack* of a block as the idle time of each block, i.e. the difference between T and the sum of the processing times of all jobs assigned to that block. Additionally, the *load* of a block refers to the total processing time of all jobs assigned to that block.

To the best of our knowledge, the $1|nr - pm|C_{max}$ problem was considered firstly by Ji et al. (2007). They propose an algorithm –labelled LPT (Longest Processing Time first)– consisting in first sorting the jobs in descending order of their processing times and then assigning them according to the First Fit bin packing policy, i.e, to the first block with sufficient slack. They also show that the worst-case ratio of the LPT algorithm is 2.

Low et al. (2010a) propose a Particle Swarm Optimization (PSO) algorithm for the $1|nr - pm|C_{max}$ problem. They use different constructive heuristics (for descriptions see Section 4) to generate an initial population with 10 individuals, and compare different versions of the PSO algorithm. From their experiments, it turns out that one of these versions yields the best results.

Finally, Yu et al. (2014) provide three constructive heuristics for $1|nr-pm|C_{max}$, called LS, LPT and MLPT: The List Scheduling algorithm (LS) generates a random order and applies the already mentioned First Fit algorithm; LPT has been described previously since it was proposed by Ji et al. (2007). Finally, MLPT is an adaptation done by Yu et al. (2014) of a method originally applied to the bin packing problem by Yue and Zhang (1995). The complexity of these three algorithms is proved by Yu et al. (2014) to be $\mathcal{O}(n^2)$, showing that all three have a worst case bound of 2. Additionally, they compare the performance bounds, concluding that MLPT is the best among them. However, they do not carry out computational experiments due to the theoretical approach of their paper.

Other related problems have been addressed in the literature. Hsu et al. (2010) study a version of the problem where the machine stops for maintenance after T units of time or after processing K jobs. Clearly, this problem is equivalent to $1|nr - pm|C_{max}$ for $K = n$. For this problem, Hsu et al. (2010) present two approaches. Firstly, a two-stage BIP (Binary Integer Programming) model is proposed in order to optimally solve the problem. In this model, the authors determine the minimum number of blocks L required for processing the n jobs in the first stage and minimize the total slack within the first $L - 1$ blocks (which is equivalent to minimize the makespan) in the second stage. Secondly, they present two heuristics. The first heuristic –called Decreasing order

with Best Fit– sorts the jobs according to LPT and then assigned them to the block with the current minimum slack, i.e. using the Best Fit bin packing policy. The second heuristic, called Butterfly order with Best Fit, sorts the jobs in the so-called butterfly order (given the jobs in LPT, select first the largest one, then the smallest, the second largest, the second smallest, and so on) and then applies the Best Fit bin packing policy. In their experiments, they show that the butterfly order performs better than the decreasing order when $K \leq \lceil \frac{n}{2} \rceil$, and the opposite otherwise.

Low et al. (2010b) study another related problem where the periodic maintenance period are not fixed, but they are flexible, being each maintenance done within a given time window. Additionally, they include the same constraint considered by Hsu et al. (2010) regarding the maximum number K of jobs assigned to an availability period. They propose six constructive heuristics obtained by combining a sequencing priority list (random order, SPT order and LPT order) with the First Fit or Best Fit policies. Additionally, they provide theoretical results about worst case ratios of some of the proposed algorithms for different special cases.

The problem $1|nr-pm, s_{ij}|C_{max}$, with s_{ij} the sequence dependent setup-time of job i scheduled prior to job j, is studied by Angel-Bello et al. (2011) and Pacheco et al. (2012). Angel-Bello et al. (2011) develop a MILP (Mixed Integer Linear Programming) model and show that it can only be used for small-sized instances. Therefore, they propose a GRASP improved by a Tabu Search phase. Pacheco et al. (2012) provide a MILP too, and a Multi-Start Tabu algorithm (MST). The authors conclude that MST is better than the GRASP by Angel-Bello et al. (2011) for the biggest instances. Finally, additional contributions on scheduling with availability constraints can be consulted in Ma et al. (2009).

After reviewing the literature, the following conclusions can be stated: On the one hand, to the best of our knowledge, the relationship between this problem and the well-known bin packing problem has been hardly studied. Ji et al. (2007) identify such relationship as the availability period can be seen as a bin with capacity T (see Figure 1). Taking into account Property 1 by Ji et al., 2007, the optimal schedule has the minimum number of blocks, corresponding to an optimal solution for the bin packing problem. Therefore, although both problems provide different solutions (since $1|nr-pm|C_{max}$ tries to minimize the load of the last block), it might be that the solution of the bin packing problem can be used to obtain high-quality solutions for the $1|nr-pm|C_{max}$ problem.

To analyse this issue, we propose a MILP model for the problem, and compare the results provided by this model with the results obtained by the mathematical model of the classical bin-packing problem. These models are presented in Section 3, and the computational results in Section 6.

On the other hand, regarding to approximate methods, to the best of our knowledge there are no comparison assessing the performance of the different methods proposed. In addition, there has been no exhaustive analysis on whether different choices of sorting criteria would yield better results (for instance, the LS algorithm proposed by Yu et al., 2014 could be tested with the Best Fit bin packing heuristic as suggested by Low et al., 2010a instead of with the First Fit). Furthermore, the same method has been employed with different names, resulting in an unclear state of the art. Since most of the proposed methods follow the same structure, we believe that a template can be developed to classify existing methods. The template could also serve to identify unexplored combinations of heuristics to be tested. In Section 4, we present and classify these methods in order to carry out a computational evaluation in Section 6. As a result of these experiments, we detect some opportunities to develop fast heuristics to further improve the results obtained by existing methods. These proposals are presented in Section 5.

3 Mathematical model

In this section we present two mixed integer linear programming (MILP) models:

- BP-MILP model. This model, proposed by Martello, Silvano; Toth (1990) for the bin packing problem, and using the notation considered for the problem $1|nr-pm|C_{max}$, assigns the jobs to the blocks minimizing the number of blocks.
- PM-MILP model. This model is an adaptation of the previous model for the problem $1|nr-pm|C_{max}$.

For the PM-MILP model, we have not adapted the model by Hsu et al. (2010) since they consider the problem with a maximum number of jobs K scheduled in each block, proposing a formulation based on two stages (i.e. they solve two models).

The notation used for BP-MILP and PM-MILP models is the following:

- Data:
  - n the number of jobs;
  - p_j the processing time of jobs, $1 \leq j \leq n$;
  - T the availability period (size of blocks).
- Variables:
  - b_i binary variables indicating if block i is occupied, $1 \leq i \leq n$;
  - x_{ij} binary variables indicating if job j is assigned to block i, $1 \leq j \leq n$, $1 \leq i \leq n$.

Note that we do not consider the value of t (see Section 1) since it does not affect to the optimal solution, according to the aforementioned Property 1 by Ji et al. (2007).

3.1 BP-MILP model

$$\min \sum_{i=1}^{n} b_i \tag{1}$$

s.t.

$$\sum_{j=1}^{n} p_j x_{ij} \leq T b_i, \quad 1 \leq i \leq n \tag{2}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \quad 1 \leq j \leq n \tag{3}$$

$$x_{ij}, b_i \in \{0, 1\}, \quad 1 \leq i, j \leq n \tag{4}$$

Equation (1) is the objective function to be minimized, i.e, the number of blocks to be occupied. Constraints (2) impose that the sum of the processing times of jobs assigned to the block i must be lower or equal to T, for all blocks. Constraints (3) impose that each job must be assigned to one and only one block, for all jobs. Finally, Constraints (4) define the binary variables used in the model.

The solution obtained for this model is the assignment of jobs to blocks. In order to generate an (approximate) solution for the $1|nr - pm|C_{max}$ problem, the makespan can be computed in the

following way: Let k be the occupied block with lowest load, then the makespan is the number of occupied blocks minus one multiplied by T plus the load the block k as it is shown in Equation (5).

$$C_{max} = T \cdot \left(\sum_{i=1}^{n} b_i - 1 \right) + \sum_{j=1}^{n} p_j x_{kj} \tag{5}$$

3.2 PM-MILP model

PM-MILP model is based in the previous model, but taking into account the following new information: M is a parameter with a high value ($M \geq T$), max_slack is a continuous variable which computes the maximum slack (empty space) of the blocks, and finally δ_i are binary variables indicating if the maximum slack is given by the block i.

$$\min T \cdot \sum_{i=1}^{n} b_i - max\_slack \tag{6}$$

s.t.

$$\sum_{j=1}^{n} p_j x_{ij} \leq T b_i, \quad 1 \leq i \leq n \tag{7}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \quad 1 \leq j \leq n \tag{8}$$

$$max\_slack \leq M(1-\delta_i) + T b_i - \sum_{j=1}^{n} p_j x_{ij}, \quad 1 \leq i \leq n \tag{9}$$

$$\sum_{j=1}^{n} x_{ij} \geq b_i, \quad 1 \leq i \leq n \tag{10}$$

$$\sum_{i=1}^{n} \delta_i = 1 \tag{11}$$

$$\delta_i \leq b_i, \quad 1 \leq i \leq n \tag{12}$$

$$x_{ij}, b_i, \delta_i \in \{0, 1\}, \quad 1 \leq i, j \leq n \tag{13}$$

$$max\_slack \geq 0 \tag{14}$$

Equation (6) is the objective to be minimized, i.e, the makespan given by T multiplied by the number of occupied blocks minus the maximum slack of the blocks. Constraints (7) and (8) are the same constraints than Constraints (2) and (3) respectively. Constraints (9) set the maximum slack

among the used blocks. For each block, Constraints (10) imply that, if a block is not occupied, then no job is assigned to this block. Constraint (11) states that only one block provides the maximum slack, and Constraints (12) imply that, if a block is not occupied, then this block does not provide the maximum slack. Finally, Constraints (13) and (14) define the binary and continuous variables used in the model, respectively.

It is clear that, among all optimal solutions for the BP-MILP model, the one containing the block with the greatest slack value is an optimal solution for the PM-MILP model. In this regard, the optimal solutions for the PM-MILP model are a subset of the optimal solutions for the BP-MILP. Therefore, although both problems are different, it is possible that, in practice, the solutions provided by the two models are very similar, which would indicate that, essentially, the $1|nr - pm|C_{max}$ problem is a bin packing problem and that solution procedures borrowed from the latter could be used to solve the former. This idea is used in the next section, where a constructive heuristic that employs concepts from the bin-packing problem is proposed for our problem.

4 Existing Heuristics: Analysis and Classification

In this section, we classify the constructive heuristics from different authors into a common template. As a part of this classification effort, new variants of these constructive heuristics are proposed to be subsequently tested.

The constructive heuristics employed in the literature consist of two phases. In the first phase, jobs are sorted by certain sorting criterion depending on the processing times of the jobs. In the second phase, a specific bin packing policy is adopted. Therefore, a general template for the constructive heuristics can be proposed to classify the existing heuristics for the problem.

Regarding the first phase, without loss of generality, let us assume that the processing times of the jobs to be scheduled are given in non decreasing order, i.e. $p_i \leq p_{i+1}$, $i = 1, \ldots, n-1$. Therefore, the so-indexed jobs can be sorted according to the following sorting criteria:

- Random (R), i.e. sorting the jobs randomly.
- Decreasing (D), i.e. sorting the jobs according to the LPT (Longest Processing Time first) rule. More specifically: $p_n, \ldots, p_1$

- Increasing (I), i.e. sorting the jobs according to the SPT (Shortest Processing Time first) rule. More specifically: $p_1, \ldots, p_n$
- V-Sharp (V), i.e. sorting the jobs with lowest processing times towards the middle of the sequence. More specifically: $p_n, p_{n-2}, \ldots, p_1, \ldots, p_{n-3}, p_{n-1}$
- A-Sharp (A), i.e. sorting the jobs with highest processing times towards the middle of the sequence: $p_2, p_4, \ldots, p_{n-1}, p_n, p_{n-2}, \ldots, p_3, p_1$
- Inserting High and Low processing times (HILO). More specifically: $p_n, p_1, p_{n-1}, p_2, p_{n-2}, p_3, \ldots$ (following the notation by Framinan et al., 2003)
- Inserting Low and High processing times (LOHI). More specifically: $p_1, p_n, p_2, p_{n-1}, p_3, p_{n-2}, \ldots$ (following the notation by Framinan et al., 2003)

Note that the above criteria include all sorting orders proposed in the literature, including the LPT rule by Ji et al. (2007) and Yu et al. (2014) (which is denoted D in the following), the List Scheduling order by Yu et al. (2014) (denoted here as R), the A-Sharp, V-Sharp, and SPT rules by Low et al. (2010a) (denoted as A, V, and I, respectively), and the decreasing order and butterfly order by Hsu et al. (2010) (denoted here as D and HILO, respectively).

In the second phase, jobs are assigned to the blocks applying a bin packing policy to the sequence obtained from the first phase. Assuming that a sequence $\Pi := (\pi_1, \ldots, \pi_n)$ has been obtained from the first phase, one of the following bin packing policies can be applied:

- First Fit (FF) bin packing policy, i.e. take jobs in Π one by one and try to assign them in the first existing block where there is sufficient slack.
- Best Fit (BF) bin packing policy, i.e. take jobs in Π one by one and try to assign them in the first existing block with the minimum slack where it fits.
- Modified First Fit (MFF): See (Yu et al., 2014) for the description of this heuristic.

Using the two phases described above, all constructive heuristics can be classified, see Table 1. From this table it can be seen that not all combinations of both phases have not been considered in the literature up to now.

Phase 1	**Phase 2**	**Notation**	**Description**	**Reference**
Random	FF	FFR	First Fit Random	Yu et al. (2014)
				Low et al. (2010b)
LPT	FF	FFD	First Fit Decreasing	Ji et al. (2007)
				Low et al. (2010a)
				Low et al. (2010b)
				Yu et al. (2014)
SPT	FF	FFI	First Fit Increasing	Low et al. (2010a)
				Low et al. (2010b)
V-Sharp	FF	FFV	First Fit V-Sharp	Low et al. (2010a)
A-Sharp	FF	FFA	First Fit A-Sharp	Low et al. (2010a)
HILO	FF	FFHILO	First Fit HILO	Low et al. (2010a)
LOHI	FF	FFLOHI	First Fit LOHI	Not considered
Random	BF	BFR	Best Fit Random	Low et al. (2010b)
LPT	BF	BFD	Best Fit Decreasing	Hsu et al. (2010)
				Low et al. (2010a)
				Low et al. (2010b)
SPT	BF	BFI	Best Fit Increasing	Low et al. (2010a)
				Low et al. (2010b)
V-Sharp	BF	BFV	Best Fit V-Sharp	Low et al. (2010a)
A-Sharp	BF	BFA	Best Fit A-Sharp	Low et al. (2010a)
HILO	BF	BFHILO	Best Fit HILO	Hsu et al. (2010)
				Low et al. (2010a)
LOHI	BF	BFLOHI	Best Fit LOHI	Not considered
LPT	MFF	MFFD	Modified FFD	Yu et al. (2014)

Table 1: Constructive Heuristics

5 Proposed heuristic

As discussed previously, the problem $1|nr-pm|C_{max}$ is similar to the classical bin packing problem. Therefore, as seen in the related literature discussed in Section 2, most methods use bin packing policies (such as Best First or First Fit).

In order to take full advantage of the bin packing policies for designing a heuristic method for the problem, we propose using a new solution encoding. More specifically, we will use a given bin packing assignment policy as an *operator* to be applied to permutation sequences.

Considering the definition of semiactive schedules as a feasible schedule where no operation that can be completed ealier without changing the order of jobs (Framinan et al., 2014), in our solution encoding, giving an order of the jobs $\Pi := (\pi_1, \ldots, \pi_n)$ and an operator, we obtain only one semi-active schedule, unequivocally determined by the order of the jobs obtained after apply a specific operator to Π, denoted $S_{operator}(\Pi)$. Note that the number of blocks used is not known in advance, but it is univocally determined by the schedule.

Taking into account this consideration, we use the following options:

- Not applying any operator (none): It is possible not apply any operator, and in this case, the feasible semi-active schedule, denoted $S_{none}(\Pi)$, is constructed assigning the jobs directly in the blocks in the order given by the sequence Π. $C_{max}(S_{none}(\Pi))$ is the value of the makespan of solution $S_{none}(\Pi)$.

- First Fit operator (FF): Starting from $j = 1$ and one block with slack T, take job π_j and try to assign it in the first existing block where there is sufficient slack (i.e. apply the FF bin packing policy). If there is no block with enough slack, then use a new block and assign the job to the newly created block. Then, repeat the procedure until all jobs in Π have been assigned to the blocks. Let us denote $S_{FF}(\Pi)$ to the so-obtained assignment of the jobs, and $C_{max}(S_{FF}(\Pi))$ the value of the makespan of solution $S_{FF}(\Pi)$.

- Best Fit operator (BF): Starting from $j = 1$ and one block with slack T, take job π_j and try to assign it in the first existing block with the minimum slack where it fits (i.e. apply the BF bin packing policy). If there is no block with enough slack, then use a new block and assign the job to the newly created block. Then, repeat the procedure until all jobs in Π have been

assigned to the blocks. Let us denote $S_{BF}(\Pi)$ to the so-obtained assignment of the jobs, and $C_{max}(S_{BF}(\Pi))$ the value of the makespan of solution $S_{BF}(\Pi)$.

We give an example in Figure 2, where we consider six jobs, and a given permutation $\Pi = (1, 2, 3, 4, 5, 6)$. We have the following options according to the operator applied:

a) If we do not apply any operator, jobs are scheduled following the given permutation and we obtain the schedule $S_{none}(\Pi)$ represented in Figure 2 a).

b) If we apply the FF operator to Π, job 1 is assigned to the first block, job 2 to the first block too, then, job 3 is assigned to the second block since it does not fit in the first block. Job 4 is assigned to the first block since this is the first block where it fits. In the same way, job 5 is assigned to the third block and job 6 is assigned to the second block. Therefore, $S_{FF}(\Pi)$ is the solution obtained by the FF operator, which implies a different schedule than the previous one, represented in Figure 2 b).

c) If we apply the BF operator to Π, job 1 is assigned to the first block, job 2 to the first block too, then, job 3 is assigned to the second block since it does not fit in the first block. Now, job 4 is assigned to the second block since this is the block with the minimum slack where it fits. In the same way, job 5 is assigned to the third block and job 6 is assigned to the first block. Therefore, $S_{BF}(\Pi)$ is the solution, which implies a different schedule than the previous cases, represented in Figure 2 c).

Therefore, our proposed method consists of two phases: First we construct a solution using the LPT rule, and then apply one of the aforementioned operators before computing the makespan. Then we apply a simple local search to explore the space of solutions. The neighbourhood structure is based on insertion of a randomly selected job in all positions, considering a first improvement strategy, i.e. if the new solution obtained after the insertion in a given position improves the best solution so far, the current solution is changed and the rest of positions are not checked. If all positions are checked without improvement the method stops. Note that, after the insertion, the evaluation is not applied to the obtained sequences, but to the solution obtained by applying one of the operators in order to provide a good assignation to the blocks. Therefore, the scheme followed is:

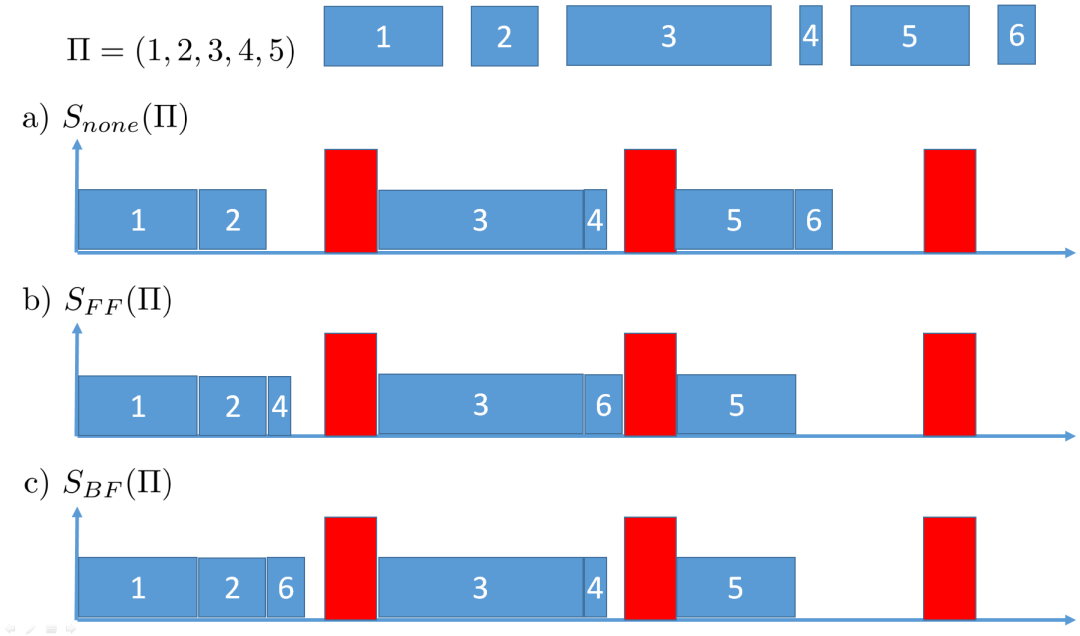

Figure 2: Example for different operators

Step 1. Let Π be the sequence obtained sorting the jobs according to the LPT rule.

Step 2. Let $S_{operator}(\Pi)$ be the sequence obtained by applying the corresponding operator to Π. Compute the makespan being $best := C_{max}(S_{operator}(\Pi))$.

Step 3. Remove one job j from Π at random. Do $i = 1$.

Step 4. Insert j in the position i of Π. Let Π' be the obtained solution.

Step 5. Apply the corresponding operator to Π', and calculate the value of the makespan $curr := C_{max}(S_{operator}(\Pi'))$.

Step 6. If $curr < best$, then $best := curr$, $\Pi := \Pi'$ and go to Step 3. Else, $i = i + 1$ and go to Step 4.

Three different heuristics have been designed with this scheme, NEW, NEW_FF and NEW_BF depending on the operator none, FF or BF applied.

6 Computational Experiments

Our objective is to determine the state of the art for the problem by comparing the existing constructive heuristics found in the literature and the new methods proposed in this paper described in Section 4 and Section 5 respectively. The experiments are carried out using the sets of instances presented in the following subsection.

6.1 Sets of instances

Among the references who tackle the same problem, only Low et al. (2010a) carry out a computational experience providing a set of instances. More specifically, they consider $n \in \{10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 150, 200, 250, 300\}$ and $p_j \sim U[1, 50]$ with 50 instances for each size (700 instances). For each instance, they generate a value of T, with $T \sim U[150, 200]$. In our paper, we consider the following set of instances:

- LOW set of instances: Replicating the one by Low et al. (2010a).
- MOD set of instances: We consider the same values of n and p_j than in LOW, but using tighter values of T, i,e, $T \sim U[50, 100]$ (note that the problem is feasible if and only if $p_j \leq T$, $\forall j = 1, \ldots, n$).

As discussed before, note that we do not include the parameter of the problem t (see the description of the problem in Section 1) since it does not have influence on the results.

For comparison, we consider the RPD (Relative Percentage Deviation) of each instance computed as follows

$$RPD = \frac{C_{max}(METH) - MIN}{MIN} \cdot 100$$

where MIN is the optimal or best bound makespan value provided for the compared methods, $METH$ is the makespan value provided for each method. ARPD refers to Average RPD.

6.2 Comparison of heuristics

In order to determine the state of the art for the $1|nr - pm|C_{max}$ problem, we test a total of 15 constructive heuristics (BFR, BFD, BFI, BFV, BFA, BFHILO, BFLOHI, FFR, FFD, FFI, FFV,

FFA, FFHILO, FFLOHI and MFFD), and the three versions of the new heuristic (NEW, NEW_BF and NEW_FF).

We have solved all instances for all methods. Table 2 shows the ARPD values obtained for each heuristic for LOW and MOD sets of instances, for the different cases of n and the Total. ARPD results are shown graphically in Figures 3 and 4, giving the 95% confidence intervals for each method. It can be seen that, for both sets of instances, the best ARPD is obtained by MILP, although it does not provide the optimal values for all instances. In fact, in both set of instances, it does not provide the minimum for all instances, mainly for the biggest instances of MOD. The best heuristics among the tested algorithms are NEW_BF and NEW_FF. Note that NEW does not provide good results, so the Local Search is not the key aspect of our new heuristic, but its combination with the operators BF or FF. It is worth to note that the average ARPD of NEW_BF is lower than that of the rest of the heuristics for all values of n. Note also that MFFD by Yu et al. (2014) provides the same results than FFD for LOW (ARPD are shown in the same row). The explanation is that both methods provide the same makespan values unless $\exists p_j > \frac{T}{2}$. For MOD set of instances, these heuristics yield different ARPD values since this set of instances provides tighter values of T, and it can be seen that MFFD is not better than FFD.

Regarding the computation times, only the best methods with the lowest ARPD values have been considered: BFD, FFD, MFFD, NEW_BF, NEW_FF and MILP. Table 3 shows the average for each value of n. It can be seen that the MILP is the slowest method, requiring more CPU times for MOD than for LOW. However MILP provides optimal solutions for LOW in less than one minute for sizes up to 80 jobs. The constructive heuristics are almost instantaneous, and, although the three new heuristics need higher computation times, these are negligible as compared to that required by MILP. For LOW, the average computation times for NEW_BF and NEW_FF are similar (2.183 and 2.112 respectively). In the case of MOD, the average computation times for LOW are 2.892 and 2.364 on average for NEW_BF and NEW_FF respectively. For both sets of instances, NEW_FF is the fastest method.

In order to see the ARPD differences in a clearer way, we analyse the seven algorithms with the lowest ARPD values. For LOW (see Figure 5), the best overall results are provided by MILP, although it requires much higher computation times than the rest (see Table 2). The best results

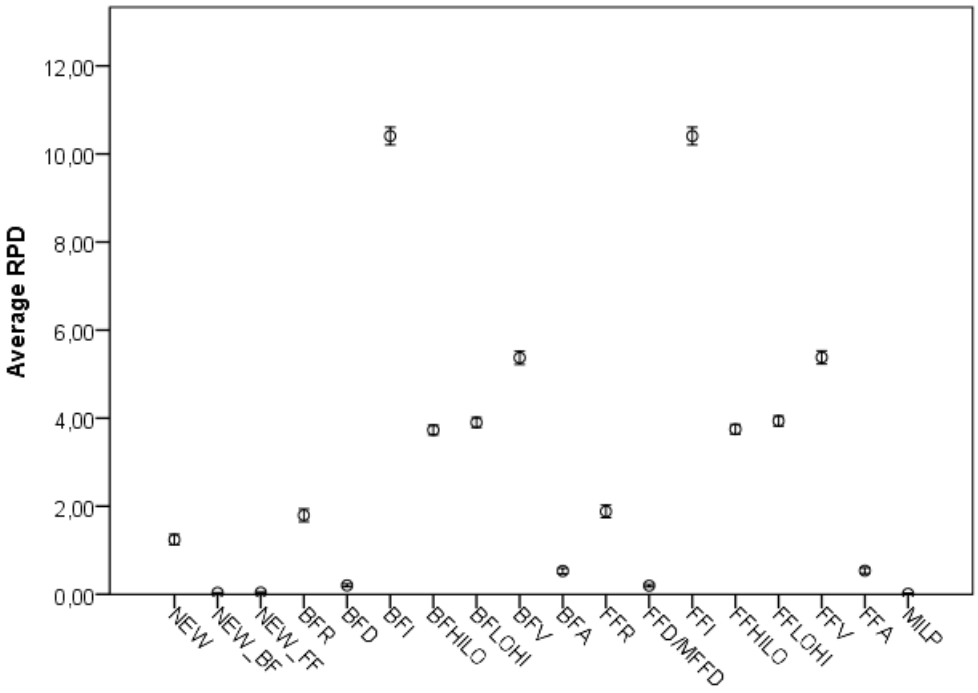

Figure 3: 95% Confidence Interval for ARPD: All heuristics and MILP. Case LOW Set of Instances

among the heuristics are provided by NEW_BF and NEW_FF. These two heuristics are statistically significant different to the rest and have the lowest standard deviation, which highlights their robustness. A similar pattern can be observed for MOD (see Figure 6). The main differences are that, in this case, there are no statistically significant differences among MILP, NEW_BF and NEW_FF, and among FFD, BFD. Finally, the worst results are obtained by MFFD, being statistically significant.

Method	N 10	20	30	40	50	60	70	80	90	100	150	200	250	300	Total
LOW Set of Instances															
NEW	0.695	1.258	1.124	1.472	1.219	0.712	0.791	1.254	1.422	1.294	1.308	1.590	1.264	1.967	1.241
NEW_BF	0.000	0.035	0.087	0.049	0.085	0.062	0.029	0.037	0.038	0.020	0.008	0.004	0.002	0.001	0.033
NEW_FF	0.000	0.035	0.088	0.056	0.096	0.083	0.048	0.061	0.053	0.035	0.009	0.004	0.004	0.002	0.041
BFR	4.240	3.232	2.871	1.981	2.255	1.472	1.709	1.538	1.409	1.299	0.921	0.775	0.727	0.696	1.795
BFD	0.651	0.738	0.415	0.265	0.236	0.161	0.107	0.093	0.072	0.057	0.010	0.004	0.004	0.002	0.201
BFI	8.996	9.805	9.577	10.655	10.316	10.067	10.273	10.700	10.363	11.278	10.792	11.125	10.859	10.899	10.407
BFHILO	3.238	3.451	4.367	3.975	4.043	3.528	3.703	3.628	3.818	3.618	3.534	3.894	3.862	3.574	3.731
BFLOHI	3.858	3.539	4.066	4.130	4.558	3.955	3.815	4.187	3.971	3.694	3.576	3.879	3.865	3.529	3.902
BFV	5.929	5.168	5.094	5.752	5.578	4.885	5.109	5.233	5.282	5.562	5.332	5.544	5.391	5.348	5.372
BFA	1.639	1.450	1.035	0.781	0.621	0.458	0.281	0.305	0.336	0.217	0.109	0.087	0.053	0.034	0.529
FFR	4.240	3.274	3.006	2.041	2.289	1.589	1.775	1.620	1.465	1.409	1.044	0.985	0.809	0.796	1.882
FFD\ MFFD	0.565	0.698	0.415	0.278	0.239	0.176	0.113	0.098	0.075	0.060	0.011	0.006	0.005	0.002	0.196
FFI	8.996	9.805	9.577	10.655	10.316	10.067	10.273	10.700	10.363	11.278	10.792	11.125	10.859	10.899	10.407
FFHILO	3.238	3.451	4.297	4.064	4.043	3.528	3.754	3.651	3.843	3.626	3.580	3.922	3.902	3.587	3.749
FFLOHI	3.858	3.689	4.066	4.130	4.637	4.014	3.815	4.282	4.035	3.680	3.609	3.867	3.888	3.542	3.937
FFV	5.929	5.168	5.094	5.884	5.578	4.885	5.109	5.233	5.282	5.562	5.332	5.544	5.391	5.348	5.381
FFA	1.639	1.446	1.038	0.803	0.618	0.472	0.308	0.314	0.343	0.231	0.118	0.096	0.054	0.037	0.537
MILP	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.002	0.006	0.037	0.043	0.085	0.012
MOD Set of instances															
NEW	4.314	7.775	8.395	17.005	13.464	12.382	13.151	15.351	18.182	14.780	21.734	21.685	24.622	27.919	15.768
NEW_BF	0.151	0.422	0.474	0.384	0.365	0.459	0.241	0.172	0.101	0.112	0.031	0.007	0.008	0.003	0.209
NEW_FF	0.206	0.448	0.607	0.450	0.510	0.557	0.313	0.223	0.155	0.122	0.043	0.028	0.021	0.017	0.264
BFR	7.841	7.318	7.129	5.977	5.899	5.808	5.239	5.264	4.290	4.054	3.419	3.103	2.933	2.732	5.072
BFD	1.846	1.293	0.871	0.691	0.671	0.655	0.365	0.285	0.237	0.157	0.050	0.036	0.030	0.023	0.515
BFI	21.292	25.256	28.502	29.821	31.210	31.808	30.809	30.312	30.885	30.889	31.335	32.628	31.755	32.235	29.910
BFHILO	6.198	6.802	7.182	7.679	9.439	8.485	7.950	8.010	8.136	8.011	7.052	8.010	8.000	8.854	7.843
BFLOHI	7.314	8.168	7.960	8.287	10.205	9.486	8.446	8.305	8.426	8.449	7.288	8.173	8.285	8.935	8.409
BFV	7.740	11.858	12.958	13.267	13.853	14.943	14.281	14.117	14.501	14.509	14.855	15.947	15.423	15.777	13.859
BFA	8.177	8.280	8.554	8.848	8.054	8.301	6.847	6.631	7.202	5.443	5.762	7.140	7.520	7.208	7.426
FFR	7.812	8.747	8.335	7.114	6.704	6.225	5.637	6.364	5.101	4.725	4.118	3.864	3.653	3.685	5.863
FFD	1.825	1.191	0.875	0.621	0.653	0.658	0.367	0.283	0.221	0.156	0.045	0.037	0.026	0.021	0.499
FFI	21.292	25.256	28.502	29.821	31.210	31.808	30.809	30.312	30.885	30.889	31.335	32.628	31.755	32.235	29.910
FFHILO	6.000	6.930	7.245	7.679	9.397	8.542	8.001	8.025	8.200	8.116	7.052	8.090	8.032	8.879	7.871
FFLOHI	7.449	8.716	8.152	8.287	10.207	9.654	8.559	8.305	8.492	8.527	7.288	8.206	8.302	8.948	8.507
FFV	7.740	12.074	12.958	13.360	13.853	15.018	14.281	14.117	14.620	14.549	14.829	15.929	15.423	15.805	13.897
FFA	7.360	7.708	8.496	8.776	8.111	8.233	6.919	6.668	7.261	5.465	5.787	7.150	7.526	7.227	7.335
MFFD	2.084	1.880	1.643	1.698	1.311	1.723	0.970	1.186	1.219	0.765	0.687	0.694	0.848	0.924	1.259
MILP	0.000	0.000	0.000	0.002	0.000	0.001	0.025	0.010	0.052	0.128	0.344	0.762	0.741	0.850	0.208

Table 2: Average RPD depending on n for each method.

	N														
Method	10	20	30	40	50	60	70	80	90	100	150	200	250	300	Total
LOW Set of Instances															
NEW_BF	0.000	0.000	0.008	0.017	0.041	0.072	0.105	0.169	0.241	0.341	1.003	2.517	8.938	17.103	2.183
NEW_FF	0.000	0.000	0.008	0.017	0.040	0.072	0.102	0.153	0.217	0.313	0.986	2.516	8.539	16.607	2.112
BFD	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.001	0.001	0.001	0.000
FFD\MFFD	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.001	0.000	0.000	0.000	0.000	0.000
MILP	0.064	0.327	18.739	0.698	6.634	7.987	41.710	48.328	151.300	209.599	369.508	704.783	887.438	901.835	239.211
MOD Set of Instances															
NEW_BF	0.001	0.003	0.007	0.018	0.040	0.069	0.110	0.164	0.265	0.345	1.261	3.784	10.888	23.527	2.892
NEW_FF	0.000	0.002	0.007	0.016	0.031	0.059	0.087	0.134	0.222	0.293	1.058	2.902	9.093	19.192	2.364
BFD	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.001	0.002	0.000
FFD	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
MFFD	0.001	0.002	0.002	0.003	0.004	0.004	0.005	0.006	0.006	0.008	0.012	0.015	0.012	0.021	0.007
MILP	0.091	1.256	34.289	97.395	241.888	400.628	471.155	735.963	834.422	811.741	874.694	900.409	900.878	901.630	514.746

Table 3: CPU Times in seconds depending on n for each method.

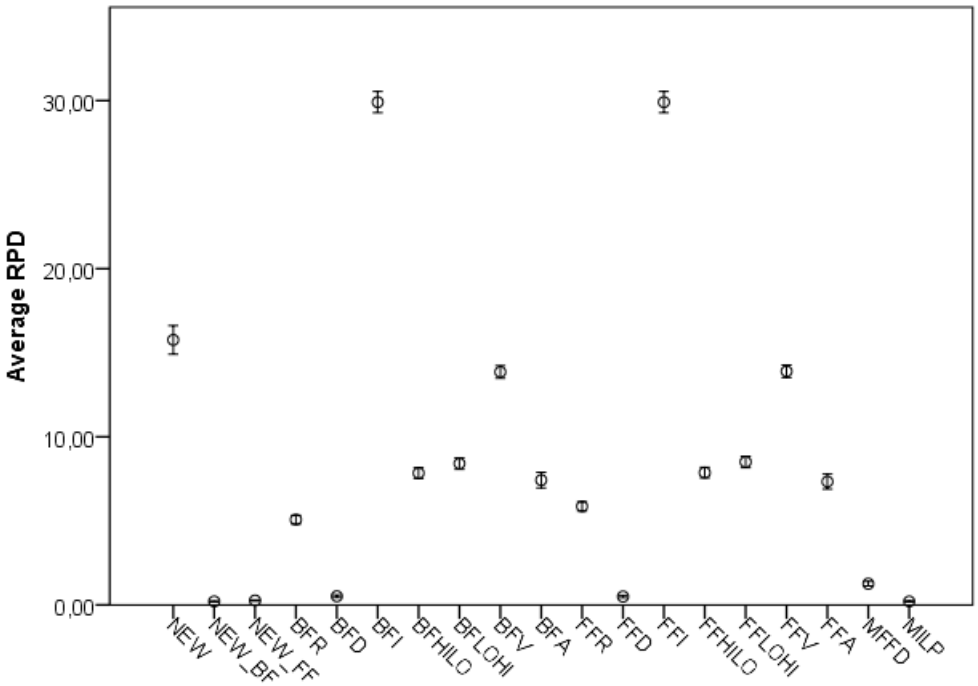

Figure 4: 95% Confidence Interval for ARPD: All heuristics and MILP. Case MOD Set of Instances

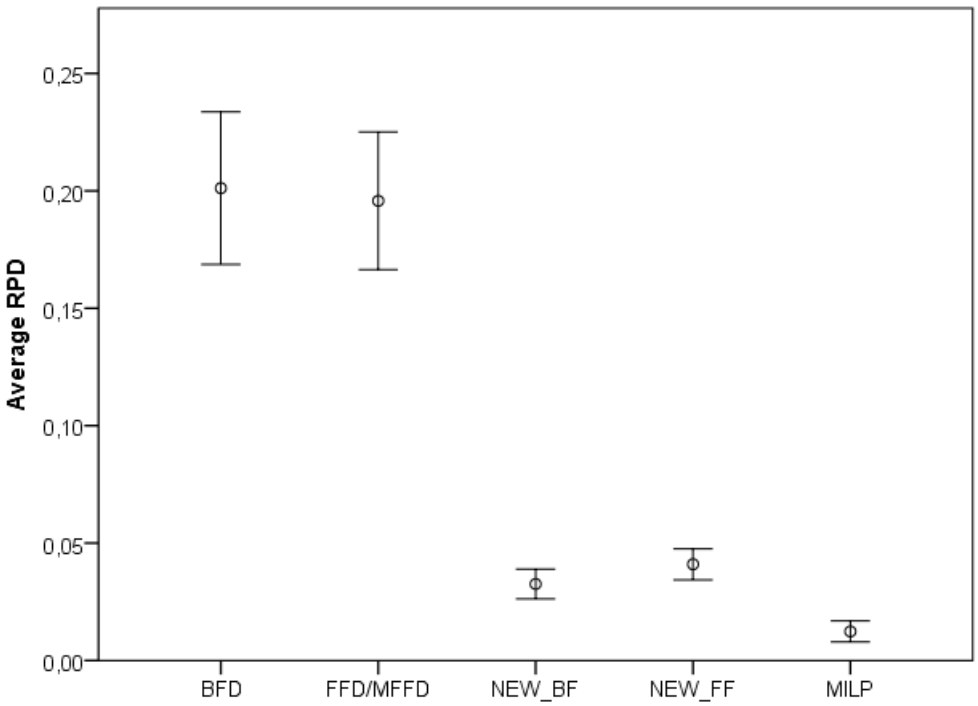

Figure 5: 95% Confidence Interval for ARPD: Best constructive heuristics and MILP. Case LOW Set of Instances

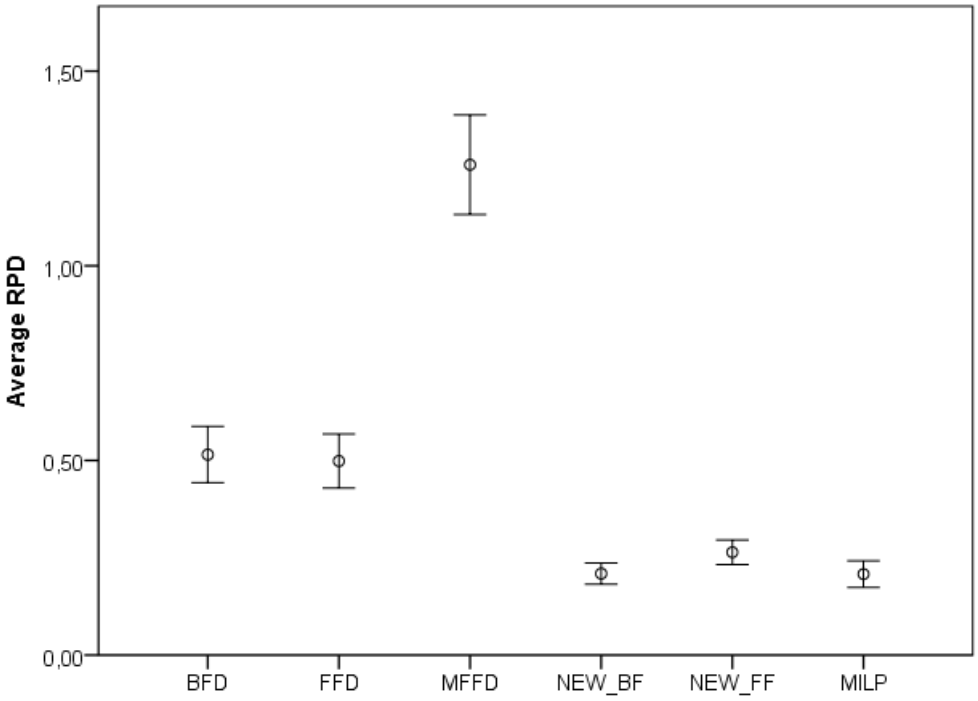

Figure 6: 95% Confidence Interval for ARPD: Best constructive heuristics and MILP. Case MOD Set of Instances

7 Conclusions

In this paper we have addressed the single machine scheduling problem with cyclic availability of machines (usually denoted as periodic maintenance in the literature). This problem is NP-hard when the objective is the makespan, and several approximate procedures were proposed, but they have not been compared up to now. In this work we first explore the relationship between our problem and the classical bin packing problem. To do so, we develop an MILP model for each problem, named PM-MILP model (for the proposed problem), and BP-MILP model (for the bin-packing problem). The optimal solutions of the PM-MILP model are a subset of the optimal solutions of the BP-MILP model. Therefore, we can borrow some ideas from approximate methods for the bin packing problem to develop constructive heuristics for the proposed problem. More specifically, we develop new heuristics that use bin packing assignment rules as operators on a job sequence. These operators are embedded into a local search. The idea is to apply the bin-packing operators before evaluating each schedule obtained in the simple local search method. In this manner, three new heuristics are tested: NEW, NEW_BF and NEW_FF using different bin-packing assignment policies. NEW does not apply any bin-packing operator and it is used as a base heuristic for comparison, while the other two use the Best Fit (BF) and First Fit (FF) bin-packing assignment rules, respectively. These proposals are compared with constructive heuristics existing in the literature, together with some new variants proposed. In total, 12 heuristics are tested against our heuristics. To assess the efficiency of our proposals, an exhaustive computational evaluation is conducted using two different sets of instances. The results show that NEW_BF and NEW_FF are the best methods in terms of quality of solutions and in CPU time requirements.

For future research lines, and due to the practical interest of scheduling problems with periodic maintenance, it could be useful to study this problem in other layouts as flowshop or parallel machines. Additionally, other objectives different to makespan have not been considered in the literature, so it could be another interesting option for future research.

References

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.

Framinan, J., 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 García, R. R. (2014). *Manufacturing scheduling systems: An integrated view on models, methods and tools*, volume 9781447162.

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.

Ji, M., He, Y., and Cheng, T. (2007). Single-machine scheduling with periodic maintenance to minimize makespan. *Computers & Operations Research*, 34(6):1764–1770.

Lee, C.-Y. (1996). Machine scheduling with an availability constraint. *Journal of Global Optimization*, 9(3-4):395–416.

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

Low, C., Ji, M., Hsu, C.-j., and Su, C.-t. (2010b). Minimizing the makespan in a single machine scheduling problems with flexible and periodic maintenance. *APPLIED MATHEMATICAL MODELLING*, 34(2):334–342.

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

Martello, Silvano; Toth, P. (1990). Bin-packing problem. In *Knapsack Problem. Algorithms and Computer Implementation*, chapter 8, page 221. John Wiley and Sons Ltd.

Pacheco, J., Angel-Bello, F., and Alvarez, A. (2012). A multi-start tabu search method for a single-machine scheduling problem with periodic maintenance and sequence-dependent set-up times. *Journal of Scheduling*, 16(6):661–673.

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.

Yue, M. and Zhang, L. (1995). A simple proof of the inequality MFFD(L)$\leq$ 71/60 OPT(L) + 1,L for the MFFD bin-packing algorithm. *Acta Mathematicae Applicatae Sinica*, 11(3):318–330.