Heuristic Methods for Single Machine Scheduling with Periodic Maintenance *

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 with cyclical unavailability periods of machines where no operation can be performed. This problem is denoted in the literature as scheduling with periodic maintenance, assuming that these regular periods are employed to perform maintenance activities. Other application is the one inspiring our research, i.e. the need of completing manufacturing operations within a shift. We focus on the single machine scheduling problem with periodic maintenance and makespan minimisation. This problem has been shown to be NP-hard, and most of the research effort has concentrated on providing approximate procedures. However, we are not aware of a computational evaluation of these methods, and to the best of our knowledge, exact methods have not been developed to solve optimally small size instances of the problem. Therefore, we aim at two objectives: On one hand, we develop a mixed integer linear programming model to solve optimally the problem and to study its relationship with the classical bin-packing problem. On the other hand, we determine the state of the art of the procedures in the literature, and propose new approximate fast methods. The computational experience shows that the proposed heuristics outperform the existing ones regarding efficiency and effectiveness.

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

*This is the Submmited Manuscript of an article published by Elsevier in 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

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

1 Introduction

In many real-life manufacturing scenarios, scheduling of jobs must take into account the existence of –usually cyclical– periods where no operation can be performed. These unavailability periods may be seen as an special case of scheduling with deterministic machine availability constraints, in this case due to periodic maintenance, non-working shifts, holidays, etc. Although the motivation of our work is the natural interruption of operations in the factory between one shift and the next one, the literature usually assumes that the unavailability periods are due to preventive maintenance activities that must be carried out cyclically. Therefore, the problem is denoted as scheduling with periodic maintenance (see e.g. Low et al., 2010; Yu et al., 2014). In our case, it is desirable that the jobs are completed within a shift, so no job should be left unfinished for the next shift. This is an usual practice in many manufacturing companies with relatively complex manual operations (such as the assembly of wiring harness in the aerospace industry, from which our case is taken), as 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.

More specifically, our problem deals with scheduling jobs for a single operation (machine) where the jobs should start and be completed within a working shift. The goal is to minimise the maximum completion time of the jobs, or makespan. Note that, although the classical single machine scheduling problem with makespan objective is trivial if no availability constraints are considered, it turns to be NP-hard in the strong sense if periodic maintenance is included (Lee, 1996). For this problem, several approximate solution procedures, many of them based on bin packing methods, have been presented in the literature, including both constructive heuristics (Ji et al., 2007; Hsu et al., 2010; Yu et al., 2014) and metaheuristics (Low et al., 2010). 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. So, their practical suitability has not been established and the state of the art remains unclear. Furthermore, the problem seems to be 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 try to study the differences between the bin packing problem and our problem by building the corresponding Mixed Integer Linear

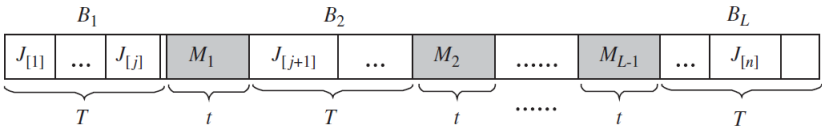

Figure 1: Illustration of the problem Ji et al. (2007)

Programming (MILP) model and comparing the solutions. Second, we carry out an exhaustive computational experimentation among the heuristics proposed in the literature for the problem under consideration. In view of the excellent results obtained by two relatively fast constructive heuristics, we embed them in a fast local search schema and propose two new heuristics which do not only outperform the rest of the existing constructive heuristics, but also the best up-to-now metaheuristic for the problem while consuming less CPU time.

The rest of the paper is organised as follows. Section 2 presents the notation for the problem and the state-of-the-art. Section 3 presents the mathematical models. In Section 4 we present all heuristics identified in the literature, and describe the heuristics proposed. In Section 6 we discuss the test beds generated to carry out the computational experience, and present the results provided by the MILP models and the heuristic methods. Finally, in Section 7 some conclusions are discussed along with future research lines.

2 Problem statement and background

The decision problem consists on scheduling n jobs, $J_1, \ldots, J_n$ in a single machine with processing times p_j, $j = 1, \ldots, n$. The machine is not continuously available due to deterministic causes, and the unavailability periods are fixed, with a common determined duration t which have to be carried out after T units of availability time. 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 operation, 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. The objective is to minimize the makespan or maximum completion time of the jobs. The 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. For a given schedule, we denote the *slack* of a block to the difference between T and the sum of the processing times of all jobs assigned to that block, i.e, the idle time of each block. Additionally, load of a block refers to the processing time of all jobs assigned to that block.

To the best of our knowledge, Ji et al. (2007) is the first paper addressing the $1|nr - pm|C_{max}$ problem. 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. (2010) 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 (described in Section 4.2) yields the best results.

Finally, Yu et al. (2014) address the $1|nr - pm|C_{max}$ problem, for which they provide three constructive heuristics, called LS, LPT and MLPT. The first one, LS, is the List Scheduling algorithm, which consists of generating a random order and apply the First Fit algorithm. LPT is the same algorithm by Ji et al. (2007). Finally, MLPT is a method originally applied to the bin packing problem by Yue and Zhang (1995) and adapted by Yu et al. (2014) for the $1|nr-pm|C_{max}$ problem, which is described in detail in Section 4.1. Yu et al. (2014) prove that the complexity of the three algorithms is $\mathcal{O}(n^2)$, show that the worst case bound in all cases is 2, and compare the performance bounds, concluding that MLPT outperforms the other methods. However, they do not carry out computational experiments to support their proposal.

Other related problems have been addressed in the literature. Hsu et al. (2010) study an extended version of the problem where the machine should stop for maintenance after a periodic interval T, 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) provide a two-stage BIP (Binary Integer Programming) model: The first-stage serves to determine the minimum number of batches L required for processing

the n jobs while the second-stage minimizes the total slack within the first $L-1$ batches, which is equivalent to minimize the makespan. Moreover, they present some heuristic approaches: The first one is called Decreasing order with Best Fit, where jobs are sorted according to LPT but the assignment to the batches is done according to the so-called Best Fit bin packing policy, i.e. jobs are assigned to the block with the current minimum slack. The second one is the so-called Butterfly order with Best Fit, which arranges the jobs according to the so-called butterfly order, i.e. given the jobs in LPT order, select first the largest one, then the smallest, the second largest, the second smallest, and so on. 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.

Another similar problem is considered by **?**, 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 periodod. They solve the problem by 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 proposed algorithms for different special cases.

The problem $1|nr-pm, s_{ij}|C_{max}$, with s_{ij} the setup-time of job j depending on the job i processed just before 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, but they show that it is not efficient. Therefore, they propose a GRASP with an improvement phase based on Tabu Search. Pacheco et al. (2012) provide a MILP too, and an algorithm called Multi-Start Tabu (MST). They compare their results to those provided by Angel-Bello et al. (2011), being the MST better than the GRASP for their biggest instances. Finally, additional contributions on scheduling with availability constraint 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, there are not mathematical programming models available for this problem, and its similarity to the bin packing problem can induce the idea that both problems are the same. Ji et al. (2007), who formally state the problem $1|nr-pm|C_{max}$, identify such similarities as the interval between two consecutive maintenance periods can be considered as a block with capacity

T (see Figure 1). It is clear (see Property 1 in Ji et al., 2007) that the optimal schedule must have the minimum number of blocks (i.e. it corresponds to an optimal solution for the bin packing problem), denoted L. Note however that $1|nr - pm|C_{max}$ is not the same problem than the bin packing problem, since $1|nr - pm|C_{max}$ tries to minimize the load of the last block. Since we are not aware of contributions analysing the differences between both problems and the opposite, 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 results are shown 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., 2010 instead of 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.

3 Mathematical model

This section presents the mixed integer linear programming (MILP) model proposed for the problem $1|nr - pm|C_{max}$. As mentioned before, in addition to provide a method to solve the problem optimally, we try to analyse the differences between the bin packing problem and our problem.

More specifically, we first apply to the $1|nr - pm|C_{max}$ problem an adaptation of the MILP model for the classical bin packing problem. We compare the differences between this model and a MILP model specifically proposed for our problem. Although these models are different and consequently, the problems are not the same, it could be that both yield similar solutions for most instances. Therefore, we solve a testbed of instances using both models and compare their results.

Therefore, we first adapt to our problem the MILP model proposed by Martello, Silvano; Toth (1990) for the bin packing problem. When applied to the $1|nr-pm|C_{max}$ problem, the bin packing problem consists on assigning the n jobs to the minimum number of blocks. The model presented in this context, denoted BP-MILP, 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, $1 \leq j \leq n$, $1 \leq i \leq n$.

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

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. 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).

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

Next, we present a MILP model for problem $1|nr - pm|C_{max}$, denoted PM-MILP. We have not adapted the models provided in the literature for similar problems by different reasons: on the one hand, Hsu et al. (2010) 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). We do not consider its adaptation since our proposal is easier to implement. On the other hand, the model by Angel-Bello et al. (2011) and its improvement by Pacheco et al. (2012) are developed for the problem considering setuptimes. This model cannot be adapted in a straightforward way since it is based on a graph where an arc $a(i, j)$ is related to the setup time $setup_{i,j}$ and the processing time p_j, loosing for our problem the mean of the arc since we do not consider setup times.

Hence, our 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.

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) sets 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.

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

As BP-MILP and PM-MILP models are different, the optimal solutions provided by both models

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

s.t.

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

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

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

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

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

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

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

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

are expected to be different. However, in view of their similarities, it would be 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 solution procedures borrowed from the latter could be used to solve the former. In contrast, if the solutions are different, specific solution procedures would have to be developed. We experimentally analyse these differences in Section 6.

4 Existing Heuristics: Analysis and Classification

In this section, we describe different heuristics found in the reviewed literature. First, we classify the constructive heuristics (CH) 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. Then, we describe the PSO (Particle Swarm Optimization) metaheuristic by Low et al. (2010). Note that we have not included the methods presented by Angel-Bello et al. (2011); Pacheco

et al. (2012) since it methods, GRASP and Tabu Search, respectively, consider some characteristics that are not standard since include intelligence based on the structure of the problem related to the setup times.

4.1 Constructive Heuristics

The constructive heuristics (CH) 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. (2010) (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) (Yu et al., 2014): The following sets are constructed: $C_1 = \{j : p_j > T/2\}$, $C_2 = \{j : T/3 < p_j \leq T/2\}$, $C_3 = \{j : \frac{T-p_n}{5} < p_j \leq T/3\}$ and $C_4 = \{j : 0 < p_j \leq \frac{T-p_n}{5}\}$. Then perform the following steps:

  Step 1. Assign the jobs in C_1 (according to sequence Π) to the first $|C_1|$ blocks so that the levels of the blocks form a non-increasing sequence (i.e., the level of a block is the total processing times of the jobs in the block).

  Step 2. From right to left (i.e., from block $X_{|C_1|}$ to X_1): if the two smallest unscheduled jobs from C_3 do not fit together in X_i , or if there is only one such job left, do nothing. Otherwise assign the smallest unscheduled job J_s from C_3 in X_i, together with the largest remaining unscheduled job J_l from C_3 that will fit, and then take J_s out of X_i and assign the largest remaining unscheduled job from C_3 that will fit together with J_l into X_i.

  Step 3. Use BF to assign the remaining jobs (according to sequence Π) to blocks starting from X_1.

  Note that MFF is FF for $C_1 = \varnothing$.

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

Table 1: Constructive Heuristics considered in the experimental evaluation

Using the two phases described above, all constructive heuristics can be classified. Not all combinations of sorting rules and policies have been tested in the literature. Table 1 shows the CH described previously, indicating whether it has been previously proposed and, if so, the corresponding reference.

4.2 PSO

In this section we briefly describe the Particle Swarm Optimization (PSO) by Low et al. (2010). In this metaheuristic, each sequence is coded as a particle in the following way: $X_i^r = (x_{i11}^r, x_{i12}^r, \ldots, x_{inn}^r)$, with $x_{ijk}^r = 1$ if job j of particle i is in the position k in the iteration r, and 0 in other case. Each particle i in the iteration r has a velocity $V_i^r = (v_{i11}^r, v_{i12}^r, \ldots, v_{inn}^r)$. Let $P_i^r = (p_{i11}^r, p_{i12}^r, \ldots, p_{inn}^r)$ be the best particle i obtained in r iterations. Let $P_g^r = (p_{g11}^r, p_{g12}^r, \ldots, p_{gnn}^r)$ be the best particle of the population obtained in r iterations. The following parameters appear:

K corresponding to the size of the initial population, $iter_{max}$ the maximal number of iterations, w_{max} and w_{min} needed to compute the value of w, $w = w_{max} - \frac{w_{max}-w_{min}}{iter_{max}} r$, c_1, c_2 and $rand_1$ and $rand_2$ to update the velocities of the particles. The specific values used in the experiments for each parameter are given in the Section 6. The specified PSO applied by Low et al. (2010) (for a more detailed description of this metaheuristic we refer to the reader to the original paper) is the following:

Step 1. $r = 0$: Initialize a population of K particles with random velocities V_i^0.

Step 2. For each particle i, evaluate $C_{max}(X_i^r)$.

Step 3. For each particle i, if $C_{max}(X_i^r) < C_{max}(P_i^r)$ then $P_i^r = X_i^r$.

Step 4. For each particle i, if $C_{max}(X_i^r) < C_{max}(P_g^r)$ then $P_g^r = X_i^r$.

Step 5. Update the velocities as follows: $v_{ijk}^r = wv_{ijk}^{r-1} + c_1 rand_1(p_{ijk}^r - x_{ijk}^r) + c_2 rand_2(p_{gjk}^r - x_{ijk}^r)$. Positions of particles are updates by the algorithm LPV (Largest Position Value) as follows:

1. For each particle i, $\pi = \emptyset$, and $S = \{s(v_{i11}^r), s(v_{i12}^r), \ldots, s(v_{inn}^r)\}$ with $s(x) = \frac{1}{1+e^{-x}}$.
2. Select the LPV of S, $s(v_{ilm}^r)$ and place the unscheduled job l in the position m in π. Remove $s(v_{ilm}^r)$ $\forall l, m$ from S. Repeat this step until $S = \emptyset$.

Step 6. If all particles are identical and $e = \frac{C_{max}(P_g^r) - C_{low}}{C_{low}} \geq 0.01$, with C_{low} a lower bound of C_{max} for the instance, then one of particles is kept and the rest $k-1$ particles are randomly generated.

Step 7. Go to step 2 until one of the following criteria is met:

1. $C_{max}(P_g^r) = C_{low}$
2. All particles are identical and $e < 0.01$.
3. $r = iter_{max}$.

5 Proposed heuristic

As commented 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.

So, in our solution encoding, a given order of the jobs $\Pi := (\pi_1, \ldots, \pi_n)$ and a operator we obtain only one feasible and semi-active schedule, unequivocally determined by the order of the jobs obtained after apply the operator to Π, denoted $S(\Pi)$. Additionally, it is possible not apply any operator, and in this case, the feasible semi-active schedule is constructed assigning the jobs directly in the blocks in the order given by the sequence Π. Note that the number of blocks used is not known in advance, and it is unequivocally determined by the schedule, so it is not needed to define a feasible solution. In this context, a feasible solution refers to the semi-active schedule obtained in this way, and it is represented by a sequence.

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

- 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 $S_{FF}(\Pi)$. Clearly, $S_{FF}(\Pi)$ and Π are not the same sequence, although they both represent feasible solutions of the problem.

- 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 $S_{BF}(\Pi)$. Clearly, $S_{BF}(\Pi)$ and Π are not the same sequence, although they both represent feasible solutions of the problem.

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 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, in our proposed method we explore the space of solutions by a simple local search. The neighbourhood structure is based on insertion of a randomly selected job in all positions, considering first improvement strategy, i.e. if the new solution obtained after insertion in a given position improves the best solution so far, the current solution is changed and the rest of positions are not checked. Note that, after 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. Two different heuristics have been designed, NEW_FF and NEW_BF respectively, according to the following scheme:

Step 1. Sort the jobs according to the LPT rule, obtaining sequence Π.

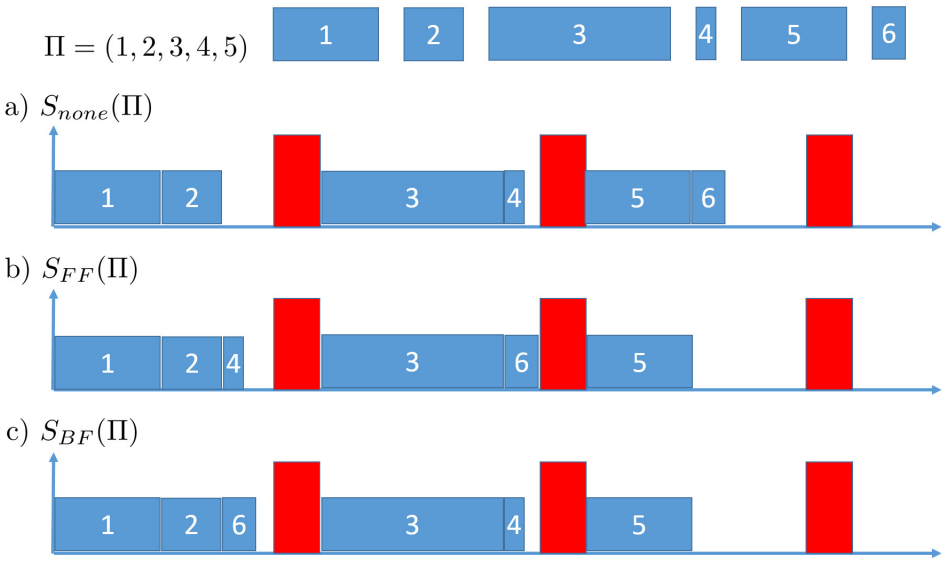

Figure 2: Example for different operators

Step 2. Apply the FF (BF) operator to Π, thus obtaining a sequence $S_{FF}(\Pi)$ ($S_{BF}(\Pi)$) and set the corresponding makespan to be *best*, i.e. $best := C_{max}(S_{FF}(\Pi))$ ($best := C_{max}(S_{BF}(\Pi))$).

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

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

Note that, if all positions are checked without improvement the method stops.

Additionally, in order to test the advantages of introducing the FF and BF operators, we propose a third version of our heuristic, denoted NEW, that can be seen as using the identity operator instead of BF or FF (case a) in Figure 2).

6 Computational Experiments

We perform several computational experiments with two objectives: First, to compare the MILP models (bin packing model and problem model included in Section 3) in order to analyse the differences between the two problems. Second, to determine the state of the art for the problem comparing the existing methods found in the literature and the new methods proposed in this paper described in Section 4 and Section 5. The experiments are carried out by the test beds presented in the following subsections.

6.1 Test beds

To perform the experiments, we need a test bed for our problem. Among the references who tackle the same problem only Low et al. (2010) do a computational experience providing a test-bed. Other test beds proposed for similar problems cannot be adapted for our problem: Angel-Bello et al. (2011) use two test beds, one of the took from the literature about the Asymmetrical Vehicle Routing Problem due to the setup time consideration, and the second one is generated randomly different matrix of the values of setup times and processing times in an aggregated way. Additionally, the values of T depend on the setup times; Pacheco et al. (2012) use the randomly generated test bed by Angel-Bello et al. (2011). Finally, **?** uses a similar test bed than the used by Low et al. (2010) with $n \in \{20, 30, 40, 50, 60, 70, 80, 90, 100, 150, 200, 250, 300\}$, $p_j \sim U[1, 20]$ and $T \in \{20, 30, 40, 50\}$, but including more information since the problem implies more data. We think that Low et al. (2010) is more suitable, since it is harder.

More specifically, the test bed by Low et al. (2010) 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 fifty instances for each size (a total of 700 instances). In their paper, for each instance they generate a value of T, with $T \sim U[150, 200]$. So Low Testbed refers to the instances with this values of T. Additionally, we consider tightest values of T, specifically $T \sim U[50, 100]$ (note that the problem is feasible if and only if $p_j \leq T$, $\forall j = 1, \ldots, n$). Modified Testbed refers to the instances with this tightest values of T.

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 MILP comparison

In order to compare the BP-MILP and PM-MILP models presented in Section 3 we have solve the Low Testbed as well as the Modified Testbed using the both models. The solutions found by the solver Gurobi for the two models with a time limit of 900 seconds has been recorded. Note that BP-MILP does not provide directly the makespan value for an instance, but the optimal number of bins, so the makespan value can be computed using Equation (5) as explained in Section 3. So ARPD values have been computed and CPU times (in seconds) have been recorded.

Table 2 and Table 3 show the results obtained grouped respect n for the 50 instances for each test bed. The column MTR (Maximum Time Reached) provides the number of instances where the maximum time limit of 900 seconds has been reached, i.e. we can not guarantee the optimality for these instances, the column Equal indicates the number of instances where the makespan computed from the BP-MILP is equal to the provided by PM-MILP, and Lower indicates the number of instances where the approximate makespan computed from the BP-MILP is lower than the provide by PM-MILP (since this model could not reach the optimal solution in 900 seconds). In general, from the results regarding ARPD, it can be seen that both models do not provide the same makespan values, as we it was expected.

For the Low Tesbed (Table 2), PM-MILP provides the optimal values for the instances up to 80 jobs (since MTR is 0 for all the cases). For these sizes, BP-MILP find the optimal number of bins (MTR=0), but the assignation given by the model do not provide the optimal makespan value. For the biggest sizes, PM-MILP is not able to provide the optimal values, and although BP-MILP find the optimal number of bins (MTR=0), the ARPD values are higher than those provided by

	PM-MILP			BP-MILP				
n	ARPD	CPU Time	MTR	ARPD	CPU Time	MTR	Equal	Lower
10	0.000	0.064	0	3.333	0.003	0	18	0
20	0.000	0.327	0	1.189	0.007	0	26	0
30	0.000	18.739	1	2.290	0.015	0	12	0
40	0.000	0.698	0	1.443	0.023	0	10	0
50	0.000	6.634	0	1.041	0.043	0	15	0
60	0.000	7.987	0	1.535	0.044	0	8	0
70	0.000	41.710	0	0.870	0.084	0	13	0
80	0.000	48.328	0	0.894	0.086	0	7	0
90	0.001	151.300	2	0.876	0.112	0	9	1
100	0.003	209.599	4	0.885	0.140	0	8	2
150	0.004	369.508	13	0.497	0.401	0	6	1
200	0.000	704.783	30	0.489	0.924	0	4	0
250	0.012	887.438	44	0.391	2.123	0	3	3
300	0.008	901.835	50	0.308	3.193	0	3	3
Total	0.002	239.211	144	1.146	0.514	0	142	10

Table 2: Results for MILP models. Low Testbed.

PM-MILP. Note that PM-MILP does not give always the best bound (it can be seen in the ARPD values and the column Lower). Regarding CPU times, it is clear that BP-MILP needs lower time to reach the optimal number of bins, this was expected since the model has less variables and constraints. Note that PM-MILP provides the optimal value in less of one minute on average for instances up to 80 jobs, and BP-MILP provides close t optimal/good bounds values in few seconds for all instances.

Modified Tesbed is proved to be harder than Low Tesbed taking into account the CPU Times and the values of MTR. For this testbed (Table 3) it can bee seen that PM-MILP provides the optimal values for the instances up to 90 jobs. Regarding ARPD, BP-MILP provides an total average similar to these provided for the Low Testbed. Note that for this testbed, BP-MILP provides approximate makespan values lower than the best bound found by PM-MILP for the biggest instances. Regarding CPU times, the time required by BP-MILP is much higher than for the Low Testbed. In this case, PM-MILP provides the optimal values for instances up to 90 in more than one hour for the bigger sizes.

	PM-MILP			BP-MILP				
n	ARPD	CPU Time	MTR	ARPD	CPU Time	MTR	Equal	Lower
10	0.000	0.090	0	1.742	0.005	0	18	0
20	0.000	1.256	0	2.033	0.018	0	7	0
30	0.000	34.289	1	2.029	0.826	0	4	0
40	0.000	97.396	3	1.490	9.129	0	6	0
50	0.000	241.888	10	1.495	3.392	0	3	0
60	0.000	400.627	13	1.410	22.352	0	2	0
70	0.000	471.155	20	1.367	44.363	2	2	0
80	0.000	735.962	34	1.010	91.735	2	1	0
90	0.000	834.422	42	0.815	198.745	9	4	0
100	0.041	811.741	42	0.882	123.689	4	3	2
150	0.046	874.693	48	0.710	331.678	17	0	5
200	0.249	900.408	50	0.356	319.796	16	1	15
250	0.218	900.878	50	0.352	481.803	25	1	15
300	0.277	901.630	50	0.372	618.418	32	0	17
Total	0.059	514.745	363	1.147	160.425	107	52	54

Table 3: Results for MILP models. Modified Testbed.

6.3 Constructive heuristics comparison

In order to determine the state of the art for the problem $1|nr - pm|C_{max}$, we test in total 15 constructive heuristics (BFR, BFD, BFI, BFV, BFA, BFHILO, BFLOHI, FFR, FFD, FFI, FFV, FFA, FFHILO, FFLOHI and MFFD), the PSO and the three versions of the new heuristic (NEW, NEW_BF and NEW_FF).

We have solved all instances for all methods. The parameters used for the PSO are the same than in Low et al. (2010): $K = 20$ with the initial population composed by the ten sequences generated by FFD, FFI, FFHILO, FFV, FFA, BFD, BFI, BFHILO, BFV and BFA and ten sequences randomly generated. Moreover, $c_1 = c_2 = 2$, $w_{max} = 0.9$, $w_{min} = 0.4$, $rand_1$, $rand_2$ uniformly generated in (0,1), and $iter_{max} = 50$.

Additionally, we solve all instances using the PM-MILP model for the problem by Gurobi. In the same way than in the previous section, we stablish 900 seconds as stopping criteria, so we do not obtain the optimal values for all the instances.

Table 4 shows the ARPD values obtained by each heuristic for the Low Testbed and Modified Testbed. ARPD results are shown for the different cases of n and the Total. Figure 3 and Figure 4 show the ARPD results graphically using the 95% confidence intervals for each method.

For both test beds, it can be observed that the best ARPD results are provided by the MILP, although it does not provide the optimal values for all instances. In fact, in both test beds it does not provide the minimum for all instances, mainly for the biggest instances of the Modified Testbed. The best heuristics are NEW_BF and NEW_FF. 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 remarkable that the average of NEW_BF are lower than rest of the heuristics (including PSO) for all values of n.

Regarding the CHs, MFFD by Yu et al. (2014) provide the same results than FFD for Low Testbed (ARPD are shown in the same row). The explanation is that both methods provide the same makespan values unless $\exists p_j > \frac{T}{2}$. Modified Testbed shows different ARPD values for these methods since this test bed provides tightest values of T. Results reveal that MFFD is not better than FFD. Our results about BFD and BFHILO are the same than those provided by Hsu et al. (2010) (since our problem is their case $K = n$), being BFD better than BFHILO for both test beds. Note that BFD and FFD are the best CHs while the opposite rule to sort the jobs, SPT combined with FF and BF, provide the worst results. PSO is the third best heuristic, but the results for BFD and FFD are not much worst than it.

Regarding computation times, Table 5 shows the average for each value of n only for the best methods with the lower ARPD values: BFD, FFD, MFFD, PSO, NEW_BF, NEW_FF and the MILP, trying to identify the behaviour of the MILP compared to the rest of methods due to its good ARPD results. It can be observed that the MILP is the slowest method, with higher times for Modified Testbed than for Low Testbed. This last observation reveals that Modified Testbed is harder to be solved optimally than Low Testbed. Although MILP is not a fast method, it provides optimal solutions for the Low Testbed in less than one minute for sizes up to 80 jobs. However, constructive heuristics are almost instantaneous, and although PSO and the three new heuristics need higher computation times, these times are negligible in comparison to the times needed by the MILP. For the Low Testbed, the average computation times for PSO, NEW_BF and NEW_FF are similar (2.629, 2.183 and 2.112 respectively). In the case of Modified Testbed, the average computation time for PSO increases more than for the rest of heuristics with respect to the times for the Low Testbed (12.767, 2.892 and 2.364 on average for PSO, NEW_BF and NEW_FF

respectively). For both testbeds NEW_FF is the fastest among these methods.

In order to see the differences of ARPD among the best methods in a clearer way, we analyse the seven with lower ARPD values detailed previously. Two ANOVA methods have been carried out with significance $\alpha = 0.05$ to determine the influence of the factor method (BFD, FFD, MFFD, PSO, NEW_BF, NEW_FF and MILP) on the ARPD value (response variable). The factor method has influence on the ARPD value for both testbeds.

In the case of Low Testbed, HDS Tukey results are provided graphically in Figure 5, which shows the 95% confidence intervals of ARPD. It can be observed that FFD(MFFD) and BFD do not present statistically significant differences. PSO provides results statistically different, but the result is close to the constructive heuristics. It can be observed that NEW_BF and NEW_FF are statistically different to the rest of heuristics, providing the best results among the heuristics, revealing robustness due to the small deviation values. The best result is provided by the MILP, but it implies the highest average computational time (see Table 4).

In the case of Modified Testbed, HDS Tukey results are provided graphically in Figure 6 which shows the 95% confidence intervals of ARPD. It can be observed that MFFD provides the worst results. In this case FFD, BFD and PSO do not present significant differences. In the same way, NEW_BF and NEW_FF provide the best results among the heuristic methods, revealing robustness in view to the small deviation values. In this case the best result is provided by the MILP, but there are not significant differences among it, NEW_BF and NEW_FF, and it implies the highest average computational time (see Table 4).

Table 4: 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 Testbed															
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
PSO	0.214	0.482	0.349	0.226	0.224	0.148	0.094	0.086	0.068	0.051	0.009	0.004	0.002	0.002	0.140
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
Modified Testbed															
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
PSO	1.256	1.108	0.861	0.611	0.641	0.629	0.356	0.275	0.216	0.152	0.044	0.034	0.024	0.018	0.445
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

Method	N 10	20	30	40	50	60	70	80	90	100	150	200	250	300	Total
Low Testbed															
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
PSO	0.007	0.060	0.170	0.356	0.568	0.884	1.025	1.704	1.997	2.298	3.220	5.990	9.019	9.501	2.629
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
Modified Testbed															
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
PSO	0.017	0.077	0.217	0.432	0.765	1.262	1.657	2.297	3.353	4.290	11.602	28.330	47.639	76.795	12.767
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 5: CPU Times in seconds depending on n for each method.

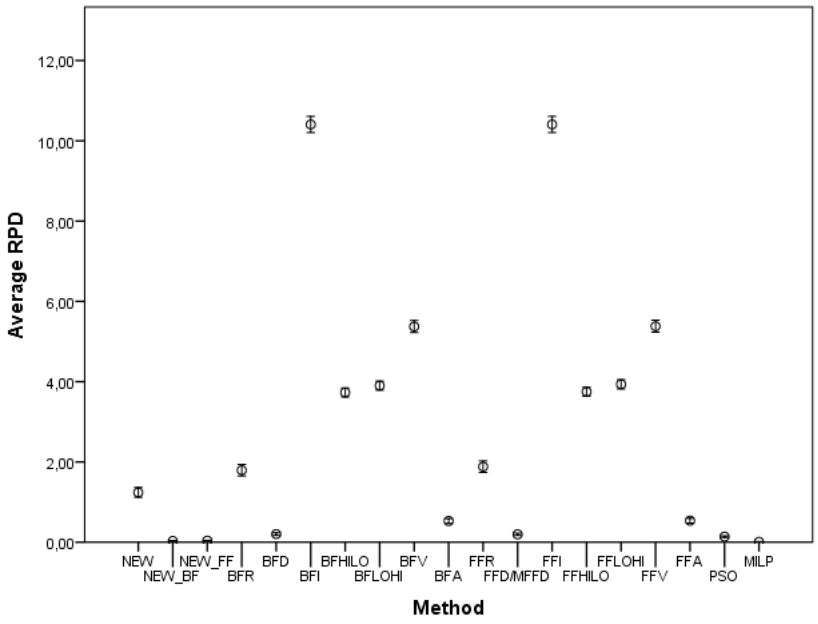

Figure 3: 95% Confidence Interval for ARPD: All heuristics. Case Low Testbed

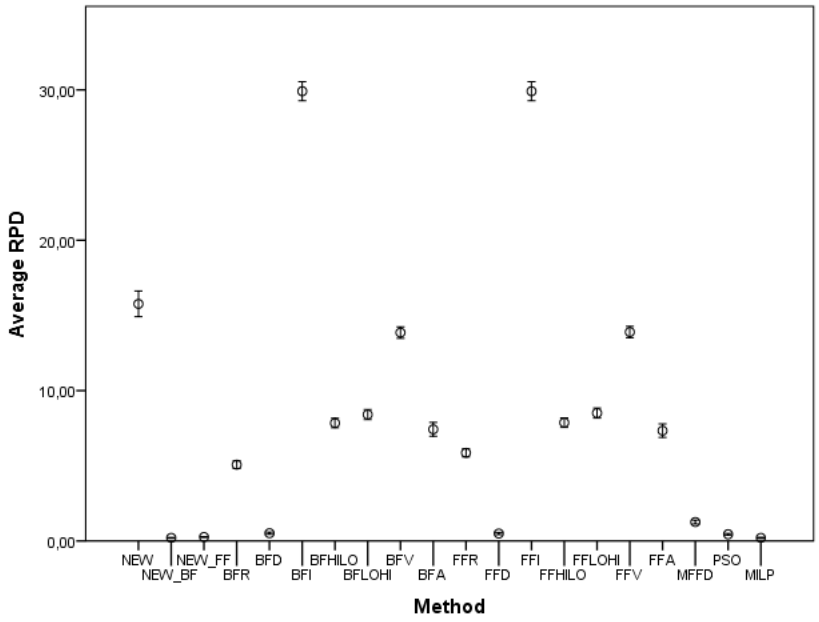

Figure 4: 95% Confidence Interval for ARPD: All heuristics. Case Modified Testbed

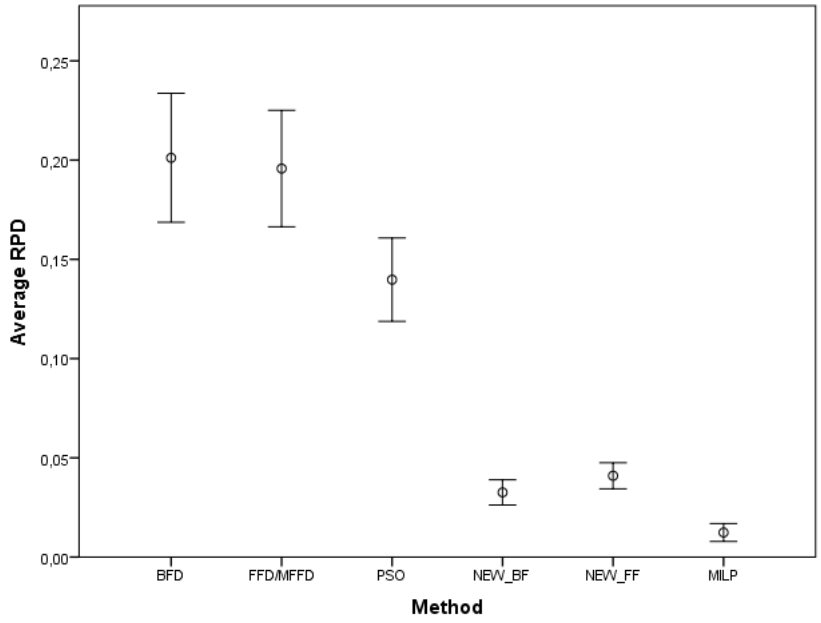

Figure 5: 95% Confidence Interval for ARPD: FFD/MFFD, BFD, PSO, NEW_BF, NEW_FF and MILP. Case Low Testbed

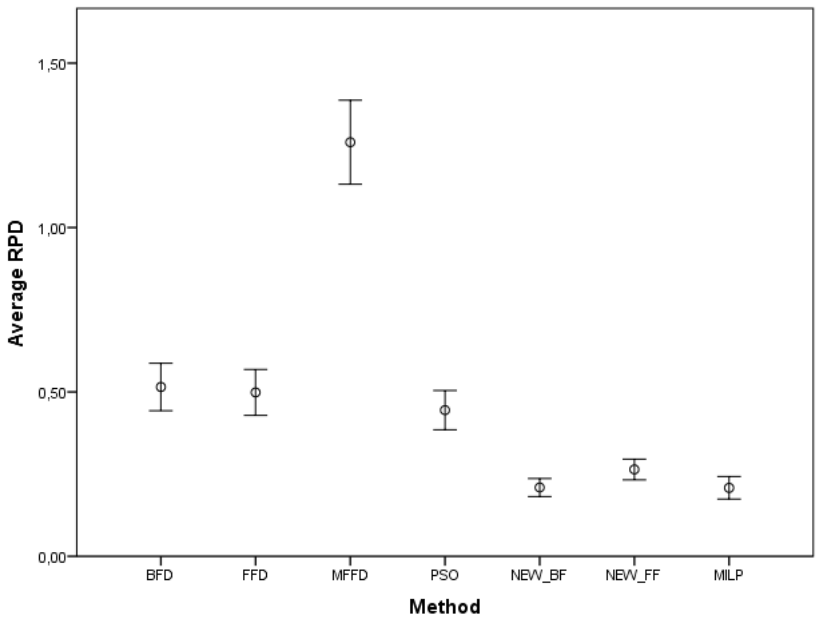

Figure 6: 95% Confidence Interval for ARPD: BFD, FFD, MFFD, PSO, NEW_BF, NEW_FF and MILP. Case Modified Testbed

7 Conclusions

In this paper we have addressed the single machine scheduling problem with periodic maintenance and makespan minimisation. This NP-hard problem has been studied previously in the literature, but it has not been modelled as a MILP. Additionally, although several approximate procedures have been proposed, no computational evaluation has been carried out.

First, we model the problem to identify its differences with the classical bin packing problem, and conclude that the solutions provided by both problems are different. Second, we conduct an exhaustive computational evaluation of the state-of-the-art heuristics, including heuristics based on the performance of two constructive heuristics denoted BFD and FFD. We propose two heuristics that use bin packing assignment rules as operators to be embedded into a local search. These heuristics are labelled NEW_FF and NEW_BF, and are tested together with the so-called NEW heuristic, which is a base case to test the influence of the operators.

Experiments have been carried out on two different test beds, one proposed in the literature, and a variation which is shown harder to be solved optimally according to the CPU time requirements of the MILP solver. In total 15 constructive heuristics, a metaheuristic (PSO), and three versions of the new heuristic, NEW, NEW_FF and NEW_BF are tested and compared including optimal solutions or best bounds provided by the MILP solver. The results show that NEW_FF and NEW_BF outperform the best existing metaheuristic for the problem (PSO) 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 have not been considered in the literature, so it could be an other 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.

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. (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.

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.