Efficient constructive and composite heuristics for the Permutation Flowshop to minimise total earliness and tardiness*

Victor Fernandez-Viagas[1], Manuel Dios[1], Jose M. Framinan[1†]

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

Camino de los Descubrimientos s/n, 41092 Seville, Spain, {vfernandezviagas,mdios,framinan}@us.es

May 19, 2016

Abstract

In this paper we address the problem of scheduling jobs in a permutation flowshop with a just-in-time objective, i.e. the minimization of the sum of total tardiness and total earliness. Since the problem is NP-hard, there are several approximate procedures available for the problem, although their performance largely depends on the due dates of the specific instance to be solved. After an in-depth analysis of the problem, different cases or sub-problems are identified and, by incorporating this knowledge, four heuristics are proposed: a fast constructive heuristic, and three different local search procedures that use the proposed constructive heuristic as initial solution.

The proposed heuristics have been compared on an extensive set of instances with the best-so-far heuristic for the problem, as well as with adaptations of efficient heuristics for similar scheduling problems. The computational results show the excellent performance of the proposed algorithms. Finally, the positive impact of the efficient heuristics is evaluated by including them as seed sequences for one of the best metaheuristic for the problem.

Keywords: Scheduling, Flowshop, Heuristics, PFSP, earliness, tardiness

*Preprint submitted to Computers & Operations Research. http://dx.doi.org/10.1016/j.cor.2016.05.006

†Corresponding author. Tel.: +34-954487214; fax: +34-954487329.

1 Introduction

The flowshop is a common manufacturing layout of a shop (Storer et al., 1992) in which a set of machines are visited by a number of jobs following each one the same route. The flowshop scheduling problem deals with establishing the sequence of jobs before each machine in the shop. If the sequence of jobs before each machine is different, extensive use of manpower or advance machines would be needed to reorder the jobs between each pair of machines. To avoid this, the most common simplification of the problem is the so-called Permutation Flowshop Scheduling Problem (denoted as PFSP) where the sequence of jobs is the same for all machines. The PFSP is one of the most studied Operations Research problems in the literature (see e.g. reviews in Reza Hejazi and Saghafian, 2005, Framinan et al., 2004 and Ruiz and Maroto, 2005). Among the goals usually employed for this optimisation problem, the most commonly used are the minimisation of makespan (see e.g. Pan et al., 2008, Nawaz et al., 1983 and Fernandez-Viagas and Framinan, 2014), the minimisation of total completion time (see e.g. Pan and Ruiz, 2013, Fernandez-Viagas and Framinan, 2015c and Dong et al., 2013) and the minimisation of total tardiness (see e.g. Vallada et al., 2008, Framinan and Leisten, 2008 and Fernandez-Viagas and Framinan, 2015d). The first two objectives are related to both a balanced use of the machines and the fast processing of the jobs, whereas total tardiness focuses on customers' satisfaction. None of them fits into a just-in-time philosophy where both the excess of inventory in the shop and the delays on the due dates should be avoided, due to the fact that just-in-time approaches aim to reduce the following aspects (Vollman et al., 1997):

- the complexity of detailed material planning,
- the need for shop-floor control,
- the work-in-process and final inventories, and
- the transactions associated with shop-floor and purchasing systems.

Given the acceptance of just-in-time systems in practice, there is a growing interest in the last decades in analysing scheduling problems where both earliness and tardiness are penalised (see e.g. reviews Baker and Scudder, 1990, Lauff and Werner, 2004, Józefowska, 2007

and Shabtay and Steiner, 2012). This type of problems is collectively known as E/T problems. More specifically, the problem under consideration is the PFSP to minimise total earliness and tardiness, which is denoted as $Fm|prmu|\sum E_j + \sum T_j$ according to the notation in e.g. Pinedo, 1995. Furthermore, insertion of idle time is not allowed, which represents a common assumption in the literature due to its undesirable effects in certain production environments (see e.g. Józefowska, 2007 and Schaller and Valente, 2013).

Some exact approaches and approximate algorithms have been proposed in the literature for the problem under study. However, both the NP-hard nature of the problem (see M'Hallah, 2014) and the huge computation times required by the optimal approaches even for small instances (not more than 20 jobs) justify the need to develop fast approximate algorithms. Thereby, several algorithms have been proposed for the problem, such as those by Zegordi et al. (1995) and by M'Hallah (2014). In this paper, four new efficient heuristics (one constructive heuristic and three composite heuristics) are proposed. These heuristics incorporate several properties and a speed up procedure in order to reduce and accelerate the search space of the heuristics. The subsequent computational experience shows that the proposed heuristics outperform the best-so-far heuristics for the problem, as well as adaptations of other state-of-the-art heuristics for related problems.

The rest of the paper is organised as follows. The background of the problem is discussed in Section 2. In Section 3, the problem under study is described and some properties are established. The proposed algorithms are explained in Section 4. A speed up procedure for the insertion phase of the heuristics as well as a complete comparison among the implemented heuristics is performed in Section 5. Additionally, the influence of using the heuristics as seed sequences in the best so-far metaheuristic (ILS) is discussed. Finally, in Section 6, conclusions are presented.

2 Literature Review

As explained in Section 1, there are several contributions to our problem. Since the heuristics proposed in Section 4 use a speed-up procedure and are based on particular properties of the instance depending on the its due dates, the literature review is divided in the following items:

- Procedures for E/T problems on a permutation flowshop where idle time cannot be inserted.

- Identification of similar instances of the scheduling problem depending on their due dates.
- Speed-up procedures for related PFSP.

Regarding flowshop scheduling problems with E/T objective without insertion of idle times, Moslehi et al. (2009) propose an optimal algorithm for the PFSP with two machines to minimise the sum of maximum earliness and tardiness. Different branch-and-bound algorithms are developed by Madhushini et al. (2009) for several multi-objective functions including total earliness and tardiness. Zegordi et al. (1995) was first in proposing an approximate algorithm (a simulated annealing algorithm) to solve the PFSP to minimise the sum of weighted earliness and tardiness. Schaller and Valente (2013) propose a genetic algorithm (GA) that outperforms several metaheuristics for related problems, also yielding favourable results regarding the CPU times when compared to the algorithm proposed by Zegordi et al. (1995). Finally, M'Hallah (2014) proposes an Iterated Local Search (ILS) where a variable neighborhood descent is iteratively repeated after a perturbation mechanism. This algorithm is shown to yield better results than the GA.

It is worth noting that both the GA and the ILS algorithms use the NEHedd (originally proposed for the $Fm|prmu|\sum T_j$ by Kim, 1993) and the Earliest Due Date or EDD rule respectively, which are either very simple seed sequences or simple adaptations from another research problems. The NEHedd is an adaptation of the NEH heuristic, originally proposed by Nawaz et al. (1983) for the PFSP to minimise makespan. The main steps of the NEH are:

1. An initial order vector, $\Pi^{IO} := (\pi_1^{IO}, \ldots \pi_n^{IO})$, is formed by sorting the jobs according to a given order. In the original formulation for makespan minimisation, jobs are sorted in non-increasing order of the sum of their processing times. If jobs are ordered according to different criteria –such as the EDD rule, or in ascending order of the sum of their processing times–, the algorithm is denoted NEHedd or NEH-flowtime, respectively.

2. A partial sequence, Π^1, is constructed with the first job of the initial order, i.e. $\Pi^1 = (\pi_1^1) = (\pi_1^{IO})$.

3. For $k = 2$ to $k = n$ the following procedure is repeated: Job π_k^{IO} is inserted in each position of Π^{k-1} (k positions are tested) and the corresponding objective function value is evaluated.

Then, the π_k^{IO} job is inserted in the partial sequence Π^{k-1} in the position with the lowest objective function value, denoted as l, $\Pi^k = (\pi_1^{k-1}, \ldots \pi_{l-1}^{k-1}, \pi_k^{IO}, \pi_l^{k-1}, \ldots \pi_{k-1}^{k-1})$.

Regarding the characterization of the problem depending on the due dates of the instance, Bagchi et al. (1986) have divided the single-machine E/T problem with common due date in two different single-machine problems depending on the common due date. Chandra et al. (2009) have extended the argumentation for the PFSP with common due dates and classify the problem into three different cases. Fernandez-Viagas and Framinan (2015b) have divided the $Fm|prmu|\sum T_j$ with different due dates for each job in four areas depending on the means and variances of the due dates, each area representing a different optimization problem.

Finally, due to the NP-hard nature of PFSP, researchers have substantially improved their results by using speed-up procedures. The most well-known of these procedures is the speed-up algorithm for the NEH developed by Taillard (1990), employed for the FPSP with makespan objective. Using this speed-up procedure, the complexity of the insertion step decreases an order of n (i.e. from $n^2 \cdot m$ to $n \cdot m$) and therefore, the complexity of the NEH decreases from $O(n^3m)$ to $O(n^2m)$. This speed-up procedure has been successfully adapted to similar scheduling problems involving makespan minimisation (see e.g. Naderi and Ruiz, 2010, Fernandez-Viagas and Framinan, 2015a, Rios-Mercado and Bard, 1998 and Fernandez-Viagas and Framinan, 2015b), although it cannot be directly applied for other objectives in the PFSP since it computes the completion time of the last job in the last machine (i.e. makespan), but not the completion time of each job. Nevertheless, Li et al. (2009) take advantages of the invariance of the completion times of jobs prior to the insertion position for the PFSP to minimise flowtime,so saving between 30-50% of CPU time are achieved. A similar procedure is also proposed by Vallada and Ruiz (2010) for the $Fm|prmu|\sum T_j$. This procedure can be directly applied for the problem under consideration and therefore is introduced in each insertion mechanism of the algorithms implemented in this paper.

3 Analysis of the problem

The problem under consideration can be stated as follows: n jobs have to be scheduled in a flowshop composed of m machines. Each job j has a processing time on each machine denoted as t_{ij}, and a due date d_j. Given a sequence of jobs $\Pi := (\pi_1, \ldots \pi_n)$, let us denote $C_{ij}(\Pi)$ the completion time of job j on machine i according to sequence Π. The completion time of the last job on the last machine, $C_{m,\pi_n}(\Pi) = C_{max}(\Pi)$, is denoted as makespan or maximum completion time of the sequence. Note that $C_{ij}(\Pi)$ can be recursively calculated as follows:

$$C_{ij}(\Pi) = \max\{C_{i-1,j}(\Pi), C_{i,j-1}(\Pi)\} + t_{ij}$$

The tardiness and earliness of job j in sequence Π are defined as $T_j(\Pi) = max\{C_{mj}(\Pi) - d_j, 0\}$ and $E_j(\Pi) = max\{d_j - C_{mj}(\Pi), 0\}$ respectively. Finally, the total tardiness (earliness) is defined as $\sum T_j(\Pi) = \sum_{\forall j} max\{C_{mj}(\Pi) - d_j, 0\}$ ($\sum E_j(\Pi) = \sum_{\forall j} max\{d_j - C_{mj}(\Pi), 0\}$).

It is clear that extremely tight or loose due dates in one instance may lead to a different problems. For the problem under study ($Fm|prmu|\sum E_j + \sum T_j$), this fact is evident according to the following two properties:

Property 3.1. *Let $\mathcal{I}$ be an instance of the $Fm|prmu|\sum E_j + \sum T_j$ problem, and WM the worst (maximum) makespan for the instance. If $d_j \geq WM\ \forall j$, an optimal solution for $\mathcal{I}$ is obtained by solving the corresponding $Fm|prmu|-\sum C_j$ problem for $\mathcal{I}$.*

Proof. Since each due date is greater or equal than the worst makespan WM, then each due date d_j is greater or equal than its completion time, $C_{mj}(\Pi)$ (i.e. $d_j \geq WM \geq C_{m,j}(\Pi), \forall j, \Pi$). Hence, minimising $\sum_{\forall j} max\{C_{m,j}(\Pi) - d_j, 0\} + \sum_{\forall j} max\{d_j - C_{m,j}(\Pi), 0\} = 0 + \sum_{\forall j} d_j - C_{m,j}(\Pi) = \sum_{\forall j} d_j - \sum_{\forall j} C_{m,j}(\Pi) = const - \sum_{\forall j} C_{m,j}(\Pi)$. □

Property 3.2. *Let $\mathcal{I}$ be an instance of the $Fm|prmu|\sum E_j + \sum T_j$ problem verifying that $d_j \leq \sum_{i=1}^{m} t_{ij}\ \ \forall j$. Then, an optimal solution for $\mathcal{I}$ can be obtained by solving the corresponding $Fm|prmu|\sum C_j$ problem for $\mathcal{I}$.*

Proof. Considering $d_j \leq t_j\ \forall j$, each completion time $C_{m,j}(\Pi)$ ($\forall\ \Pi$) is greater or equal than its due date, d_j, since t_j is a lower bound of the makespan of the job j. Hence $\sum_{\forall j} max\{C_{m,j}(\Pi) -$

$d_j, 0\} + \sum_{\forall j} max\{d_j - C_{m,j}(\Pi), 0\} = \sum_{\forall j}(C_{m,j}(\Pi) - d_j) + 0 = \sum_{\forall j} C_{m,j}(\Pi) - \sum_{\forall j} d_j = \sum_{\forall j} C_{m,j}(\Pi) + const.$ □

From these properties, it is clear that extremely loose due dates transform the problem into a PFSP with the objetive of flowtime maximization. Extremely tight due dates lead to a problem similar to the PFSP with flowtime minimisation. Both bounds represent opposite objective functions and therefore, algorithms specifically focused on yielding good solutions for instances with loose due dates would necessarily perform bad for tight due dates. Thereby, depending on the due dates, three different scheduling problems can be solved: the $Fm|prmu|\sum C_j$ problem in case of tight due dates, the $Fm|prmu|-\sum C_j$ problem in case of loose due dates and the original $Fm|prmu|\sum E_j + \sum T_j$ problem in the rest of the cases. This fact speaks for the difficulties to find constructive heuristics that perform well for the problem, which in our opinion is reflected by the fact that the NEH –an algorithm not designed for this specific problem– is the only constructive heuristic proposed so far.

Typically, the good performance of a constructive heuristic is due to the fact that the objective computed in the iterations of the algorithm is similar to the objective function of the problem. Thereby, when minimising e.g. total flowtime in the PFSP, the choice of a partial sequence fulfilling the minimisation of total flowtime clearly seems to have a good performance when the sequence is completed. This is a consequence of having a regular measure as objective. Since this is not the case for our problem, the algorithm may not work well. In fact, the aforementioned properties confirm this fact and show how the objective of the constructive heuristics in their iterations could be distorted: A partial sequence could have loose due dates but, once completed, these due dates would become tight and thus, the algorithm would solve a completely different objective during its iterations than the objective function. This fact could also explain the good performance of composite heuristics as compared to constructive heuristics (as discussed in Section 5.5).

In order to overcome the aforementioned problems, efficient heuristics for the problem under consideration should be designed according to the following ideas:

- They should be very fast in order to work as soon as possible with complete sequences. In

this manner, it is easier to identify whether the instance has loose due dates, or tight ones.

- They should incorporate an analysis of both sequenced and non-sequenced jobs in each iteration.
- They should avoid the use of local search procedures operating with non-complete sequences.

Based on these ideas and on the properties discussed before, a number of algorithms are proposed. These are discussed in the next section.

4 Proposed Algorithms

Following the recommendations in the previous section regarding very fast heuristics and complete local search methods, four heuristics are proposed for the $Fm|prmu|\sum E_j + \sum T_j$ problem:

- an adaptive constructive heuristic (see Section 4.1), denoted as ACH1,
- a composite heuristic (see Section 4.2), denoted as ACH2, composed by ACH1 plus a bounded local search procedure labelled BLS,
- a composite heuristic (see Section 4.2), denoted as ACH3, formed by ACH1 plus an iterative bounded relative local search method, iBRLS,
- a composite heuristic (see Section 4.2), denoted as ACH4, formed by ACH1 and an iterative local search method, iLS.

Additionally, a speed-up procedure is described in Section 4.3 to accelerate the insertion phases of all implemented algorithms.

4.1 Proposed Constructive Heuristic

ACH1 tries to find a good solution using very short computational times so it can embedded in more sophisticated constructive and composite heuristics such as the ones proposed in the next subsections. The procedure of this heuristic is relatively simple: Beginning with a partial

sequence with a single job, the procedure constructs a final sequence appending one by one jobs at the end of the partial sequence according to an index $\xi_{u_{jk}}(\Pi)$. Let us denote by $\Pi^k := (\pi_1, ..., \pi_k)$ the partial sequence in iteration k and by $\mathcal{U}_k$ the set of unsequenced jobs of that sequence (u_{jk} the jth unsequenced job with $j \in [1, n-k]$). Additionally, let NT_k be the number of tardy jobs in iteration k in $\mathcal{U}_k$. The algorithm chooses the job from $\mathcal{U}_k$ with the lowest value of $\xi_{u_{jk}}(\Pi^k)$ and places it at the end of sequence Π^k, i.e. in position $k+1$, forming the sequence Π^{k+1} of the next iteration. This procedure has been shown to be very efficient for other decision problems, being the appropriate choice of the index the critical issue for the efficiency of the algorithm (see e.g. Fernandez-Viagas and Framinan, 2015c). This difficulty increases in our case due to the strong dependence of the best solutions on the due dates of the jobs. The index must be adapted to solve different problems depending on the due dates (loose due dates, tight ones, or neither of them). In Section 3, three different situations have been identified: tight due dates ($Fm|prmu|\sum C_j$ decision problem), loose due dates ($Fm|prmu|-\sum C_j$ decision problem) and normal due dates ($Fm|prmu|\sum E_j + \sum T_j$). Therefore, at each iteration, the algorithm would check whether the sequence is within one of these cases:

- Case 1: Tight due dates (i.e., the problem is similar to the $Fm|prmu|\sum C_j$). There are hundred of heuristics solving the $Fm|prmu|\sum C_j$ in the literature. Particularly, Fernandez-Viagas and Framinan (2015c) designed an efficient constructive heuristic following a similar procedure of insertion in last position of the partial sequence. There, jobs are chosen according to the $\xi^1_{u_{jk}}(\Pi^k)$ index, Equation (1), which considers the minimization of the completion time and the weighted idle time of the candidates jobs (i.e. u_{ik} with $j \in [1, n-k]$) to be inserted:

$$\xi_{u_{jk}}(\Pi^k) = \xi^1_{u_{jk}}(\Pi^k) = \frac{(n-k-2)}{4} \cdot IT_{u_{jk}}(\Pi^k) + C_{m,u_{jk}}(\Pi^k) \tag{1}$$

  where $IT_j(\Pi^k)$ are:

$$IT_{u_{jk}}(\Pi^k) = \sum_{i=2}^{m} \frac{m \cdot max\{C_{i-1,u_{jk}}(\Pi^k) - C_{i,\pi_k}(\Pi^k), 0\}}{i-1+k\cdot(m-i+1)/(n-2)} \tag{2}$$

In this paper, this index is directly incorporated into the ACH1 heuristic when due dates of jobs are tight, assuming that this is the case if, in iteration k, the fraction of tardy jobs in the partial sequence is equal to or greater than a. More specifically, the index is used if $a \cdot 100\%$ $(NT_k/(n-k) \geq a)$ and there are at least four tardy jobs. Note that a is a parameter of the algorithm which is introduced in order to determine when the algorithm is adapted to solve $Fm|prmu|\sum C_j$. The suitable values for a are discussed later.

- Case 2: Loose due dates (the problem is similar to $Fm|prmu|-\sum C_j$). We divide this case in two cases depending on how loose the due dates are. The idea behind these two subcases is to separate the case where all due dates are extremely high (the earliness can be omitted and the problem is similar to the $Fm|prmu|-\sum C_j$) and where only some of the due dates are extremely high (earliness should be also considered). To the best of our knowledge, there are no algorithms for the PFSP to maximise total flowtime as it is not a common objective to be followed by companies. Therefore, we consider the inverse index for the first subcase, $\xi_{u_{jk}}(\Pi^k) = -\xi^1_{u_{jk}}(\Pi^k)$. The conditions to apply this index are fulfilled when there are at least four candidates $(n-k>3)$, all candidates in iteration k are in earliness (i.e. $C_{m,u_{jk}}(\Pi^k) < d_{u_{jk}}\ \forall j$), and $NE_k = n-k$ where NE_k is the number of jobs in $\mathcal{U}_k$ whose earliness is higher than $(n-k)\cdot c$. On the other hand, when due dates are not so loose (this fact is measured by the condition $b\cdot(n-k) \leq NE_k < n-k$) we consider the index ξ^2 which adds the earliness of job $E_{u_{jk}}(\Pi^k)$ to the index $-\xi^1$:

$$\xi_{u_{jk}}(\Pi^k) = \xi^2_{u_{jk}}(\Pi^k) = -\frac{(n-k-2)}{4}\cdot IT_{u_{jk}}(\Pi^k) - C_{m,u_{jk}}(\Pi^k) + E_{u_{jk}}(\Pi^k) \qquad (3)$$

  Note that b and c are parameters of the proposed algorithm.

- Case 3: Neither loose nor tight due dates, i.e. the instance is a pure $Fm|prmu|\sum E_j + \sum T_j$ problem. When a job is inserted at the end of the sequence, the algorithm should focus in the minimization of total earliness and tardiness. Then, we try to place each candidate at the end of the partial sequence and, in order to select the job to be inserted, we focus on the minimisation of earliness by using the index in Equation (4):

Case	Condition	$\xi_{u_{jk}}(\Pi^k)$ index
Tight due dates	$NT_k/(n-k) \geq a$ $NT_k > 3$	$\xi^1_{u_{jk}}(\Pi^k)$
Loose due dates (Subcase 1)	$C_{m,u_{jk}}(\Pi^k) < d_{u_{jk}}$, $\forall j$ $n-k > 3$ $NE_k = n-k$	$-\xi^1_{u_{jk}}(\Pi^k)$
Loose due dates (Subcase 2)	$C_{m,u_{jk}}(\Pi^k) < d_{u_{jk}}$, $\forall j$ $n-k > 3$ $b \cdot (n-k) \leq NE_k < n-k$	$\xi^2_{u_{jk}}(\Pi^k)$
Neither loose nor tight due dates	Otherwise	$\xi^3_{u_{jk}}(\Pi^k)$

Table 1: Summary of cases in the ACH

$$\xi_{u_{jk}}(\Pi^k) = \xi^3_{u_{jk}}(\Pi^k) = E_{u_{jk}}(\Pi^k) \tag{4}$$

Note that the minimisation of earliness implicitly takes into account also tardy jobs as their earliness is equal to zero and then, they would be the first to be chosen.

The different values adopted by $\xi_{u_{jk}}(\Pi^k)$ are summarized in Table 1 together with the corresponding conditions. Once the index has been identified, we choose the job with the lowest value and place it at the end of the current partial sequence. Ties are broken according to the weighted idle time of the candidate jobs ($IT_{u_{jk}}(\Pi^k)$) for all cases.

Finally, note that, when inserting a job at the end of the sequence, the completion time of previous last job remains the same and hence, only the completion times of the inserted job on each machine has to be computed, which can be easily done in $O(m)$. Analogously, earliness time and/or weighted idle time of each candidate job can be computed with complexity m. Thereby, the proposed ACH1 constructive heuristic has the same order of complexity than the NEH. However, the heuristic heavily decreases the CPU time due to the complexity of each evaluation, which is simply m. Therefore, the complexity of the proposed heuristic is $\frac{(n+1)\cdot n}{2} \cdot m \sim O(n^2 \cdot m)$.

The pseudo code of the ACH1 heuristic is shown in Figure 1.

4.2 Proposed Composite heuristics

Once a sequence has been obtained by the fast ACH1 constructive heuristic, it can be improved by three different insertion-based local search methods, leading to composite heuristics. The first two local search methods try to reduce the computational effort required in each iteration by avoiding

Procedure *ACH1*

Determination of each $IT_{j,0}$, $CT_{j,0}$ and $E_{j,0}$

π_1 := Job with least value of $E_{j,0}$ breaking ties in favor of the job with the lowest $IT_{j,0}$;

$\Pi^1 = (\pi_1)$

for $k = 1$ **to** $n-1$ **do**

Determination of each $IT_{u_{jk}}(\Pi^k)$, $C_{m,u_{jk}}(\Pi^k)$, NT_k, NE_k, and $E_{u_{jk}}(\Pi^k)$, $\forall j \in [1, n-k]$;

if $NT_k/(n-k) \geq a$ & $NT_k > 3$ **then**

for $j = 1$ **to** $k-n$ **do**

$\xi_{u_{jk}}(\Pi^k) = \xi^1_{u_{jk}}(\Pi^k) = \frac{(n-k-2)}{4} \cdot IT_{u_{jk}}(\Pi^k) + C_{m,u_{jk}}(\Pi^k)$

end

else if *AllEarliness* & $n-k > 3$ & $NE_k = n-k$ **then**

for $j = 1$ **to** $k-n$ **do**

$\xi_{u_{jk}}(\Pi^k) = -\xi^1_{u_{jk}}(\Pi^k) = -\frac{(n-k-2)}{4} \cdot IT_{u_{jk}}(\Pi^k) - C_{m,u_{jk}}(\Pi^k)$

end

end

else if *AllEarliness* & $n-k > 3$ & $b \cdot (n-k) \leq NE_k < n-k$ **then**

for $j = 1$ **to** $k-n$ **do**

$\xi_{u_{jk}}(\Pi^k) = \xi^2_{u_{jk}}(\Pi^k) = -\frac{(n-k-2)}{4} \cdot IT_{u_{jk}}(\Pi^k) - C_{m,u_{jk}}(\Pi^k) + E_{u_{jk}}(\Pi^k)$

end

end

else

for $j = 1$ **to** $k-n$ **do**

$\xi_{u_{jk}}(\Pi^k) = \xi^3_{u_{jk}}(\Pi^k) = E_{u_{jk}}(\Pi^k)$

end

end

α:= Job with the lowest value of $\xi_{u_{jk}}(\Pi^k)$ in iteration k, breaking ties in favor of the job with the lowest $IT_{u_{jk}}(\Pi^k)$.

Π^{k+1}:= Permutation obtained by inserting job α at the end of sequence Π^k.

end

end

Figure 1: ACH1

Procedure *ACH2()*
$(\Pi, \sum E_j + \sum T_j) = ACH1();$
$(\Pi, \sum E_j + \sum T_j) = BLS(\Pi, \sum E_j + \sum T_j);$
end

Figure 2: ACH2

the insertion of a job far from its optimal position in order to accelerate the intensification of the procedures. The idea is to bound the positions where the jobs are inserted between $P1$ and $P2$, which are defined in Equations (5) and (6).

$$P1 = \min\{P_{AS}, P_{edd}\} \tag{5}$$

$$P2 = \max\{P_{AS}, P_{edd}\} \tag{6}$$

where P_{edd} is the position in the EDD rule of π_j, and P_{AS} is the position that π_j should have in the actual sequence in order to get a minimum value of E_{π_j} and T_{π_j}.

Recently, bounded local search methods have been successfully applied to scheduling problems (see e.g. Fernandez-Viagas and Framinan, 2015a and Fernandez-Viagas and Framinan, 2015b). The key of this success comes from the reduction of the search space in each iteration of the local search methods in order to avoid sequences that are far from the optimum, therefore decreasing the computational effort of the algorithms. More specifically, the proposed composite heuristics are:

- ACH2. It performs a bounded local search (denoted BLS) after the ACH1 heuristic. The BLS tries to insert each π_j in each position between $P1$ and $P2$. Pseudo code of both ACH2 and BLS methods are shown in Figures 2 and 3 respectively.

- ACH3. It performs an iterative bounded local search method, denoted as iBRLS after the ACH1 heuristic. Similarly to the BLS method, the iBRLS tries to iteratively insert each job π_j between $P1$ and $P2$ until there is no improvement after trying n consecutive jobs. Pseudo code of ACH3 and iBRLS are detailed in Figure 4 and 5 respectively.

- ACH4. This heuristic carries out a iterative local search method (iLS) after the ACH1

Procedure *BLS(*Π, OF*)*
  $OF_b = OF$
  for $j = 1$ **to** n **do**
    Π^0 := remove job π_j from Π;
    Calculate $P1$ and $P2$;
    Test job π_j between the positions $P1$ and $P2$ of Π^0;
    Π := permutation obtained by inserting π_j in the position $j \in [P1, P2]$ of Π^0 with less total earliness and tardiness, $OF^{'}$;
    if $OF^{'} < OF_b$ **then**
      $OF_b = OF^{'}$;
      $\Pi^b := \Pi$;
    end
  end
  return Π^b and OF_b;
end

Figure 3: Bounded Local Search, BLS

Procedure *ACH3()*
  $(\Pi, \sum E_j + \sum T_j) = ACH1()$;
  $(\Pi, \sum E_j + \sum T_j) = iBRLS(\Pi, \sum E_j + \sum T_j)$;
end

Figure 4: ACH3

Procedure $iBRLS(\Pi, OF)$
  $OF_b = OF$
  $h = 1$;
  $i = 1$;
  $\Pi^b := \Pi$;
  while $i <= n$ **do**
    $j := h \bmod n$;
    $\Pi^0 :=$ remove job π_j from Π;
    Calculate $P1$ and $P2$;
    Test job π_j between the positions $P1$ and $P2$ of Π^0;
    $\Pi :=$ permutation obtained by inserting π_j in the position $j \in [P1, P2]$ of Π^0 with less total earliness and tardiness, $OF^{'}$;
    if $OF^{'} < OF_b$ **then**
      $OF_b = OF^{'}$;
      $i = 1$;
      $\Pi^b := \Pi$;
    else
      $i++$;
    end
    $h++$;
  end
  return Π^b and OF_b;
end

Figure 5: Iterative Bounded Relative Local Search, iBRLS

```
Procedure ACH4()
    (Π, ∑E_j + ∑T_j) = ACH1();
    (Π, ∑E_j + ∑T_j) = iLS(Π, ∑E_j + ∑T_j);
end
```

Figure 6: ACH4

```
Procedure iLS(Π, OF)
    OF_b = OF
    flag := false;
    while flag = false do
        flag := false;
        for j = 1 to n do
            Π^0 := remove job π_j from Π;
            Test job π_j in each position of Π^0;
            Π := permutation obtained by inserting π_j in the position j of Π^0 with less
            total earliness and tardiness, OF';
            if OF' < OF_b then
                OF_b = OF';
                Π^b := Π;
                flag := true;
            end
        end
    end
    return Π^b and OF_b;
end
```

Figure 7: Iterative Local Search, iLS

heuristic. This local search method simply tries to place each job π_j in the rest of positions of the current sequence and has been extensively used in the literature (see e.g. Ruiz and Stützle, 2007, Li and Wu, 2005 and Pan and Wang, 2012). The procedure is repeated until there are no more improvements. Pseudo codes for ACH4 and iLS methods are shown in Figures 6 and 7.

4.3 Speed Up Procedure

In this Section, a simple speed-up procedure to accelerate the insertion phases of the algorithms for the $Fm|prmu|\sum E_j + \sum T_j$ problem is described. Let Π^k be a partial sequence with k jobs and l the job which is to be inserted in position $j \in [1, k+1]$. Similarly to the speed up methods proposed by Li et al. (2009) and Vallada and Ruiz (2010), this method stores the completion

time of each job on each machine of the partial sequence Π^k. When the job l is inserted in each position j of the partial sequence, the completion times of the jobs prior to this position j are already known and do not have to be recomputed. According to several studies, this procedure reduces the CPU times between 30% and 50% and is therefore introduced in each insertion phase of all algorithms implemented in this paper.

5 Computational Experience

In this paper, the proposed algorithms are compared against the most efficient heuristics in the literature. The procedure adopted to evaluate the algorithms is the following: First, we introduce the set of instances used for both the experimental parameter tuning and the comparison among heuristics. In Section 5.2, a full factorial design of experiments is carried out to find the best values of the parameters of the algorithms proposed. The algorithms under comparison are listed in Section 5.3. The indicators to define the efficient heuristics are introduced in Section 5.4. Using these indicators, constructive and composite heuristics are compared in Section 5.5, leading to the identification of the set of efficient heuristics for the problem. Finally, in Section 5.6, the efficient heuristics are compared as seed sequences of one of the best metaheuristic for the problem.

5.1 Benchmark

In this Section, the following two different sets of instances are presented to evaluate the algorithms. Note that different sets of instances are used in order to avoid an over calibration of the proposed heuristics when the parameters a, b and c are defined. These sets are:

- Benchmark $\mathcal{B}_1$ for the calibration of the proposed heuristics. This benchmark is composed of a set of 1,080 instances generated according to the procedure by Vallada and Ruiz (2010). The benchmark is formed by 10 instances for each combination of $n = \{50, 150, 250, 350\}$, $m = \{10, 30, 50\}$, $T = \{0.2, 0.4, 0.6\}$ and $R = \{0.2, 0.6, 1.0\}$, where T and R are parameters related to the mean and standard deviation of the due dates respectively. These due dates are generated using the procedure described by Potts and Van Wassenhove (1982), i.e. following a uniform distribution between $P \cdot (1 - T - R/2)$ and $P \cdot (1 - T + R/2)$, where

P is a lower bound for the makespan. Processing times are generated using a uniform distribution [1, 99].

- Benchmark $\mathcal{B}_2$ for comparison among the implemented heuristics. This benchmark is composed of a set of 540 instances of Vallada et al. (2008) (available in http://soa.iti.es) and is the most extended benchmark for the PFSP with due dates. This benchmark consists of five instances for each combination of $n = \{50, 150, 250, 350\}$, $m = \{10, 30, 50\}$, $T = \{0.2, 0.4, 0.6\}$ and $R = \{0.2, 0.6, 1.0\}$. Processing times are generated using a [1,99] uniform distribution.

5.2 Experimental Parameter Tuning

The proposed heuristic ACH1 uses three parameters: a, b and c. In this section, a full factorial design of experiments is carried out to determine their best values on the set of instances $\mathcal{B}_1$. The following values are chosen for the experiments:

- $a = \{0.8, 0.85, 0.9, 0.95, 1\}$,
- $b = \{0.4, 0.45, 0.5, 0.55, 0.6\}$,
- $c = \{25, 30, 35, 40, 45, 50, 55\}$

In each instance, the ACH1 heuristic is evaluated according to Equation (7):

$$RPD1 = \frac{OF - Base}{Base} \cdot 100 \tag{7}$$

where OF and $Base$ are the solutions obtained by the ACH1 heuristic and a reference algorithm (NEHedd) respectively.

Since normality and homoscedasticity assumptions are not fulfilled, a non-parametric Kruskal-Wallis test is carried out. The p-values are 0.267, 0.865 and 0.000 for parameters a, b and c respectively. Results show that there is statistically significant differences only between the levels of parameter c. Additionally, among the 175 combinations of a, b and c, the best results are found for $a = 0.90$, $b = 0.55$ and $c = 30$. These values are subsequently used in each heuristic which incorporates the ACH1, i.e. ACH2, ACH3 and ACH4.

5.3 Implemented algorithms

The performance of the proposed heuristics is tested against the most efficient heuristics for the problem, as well as for some of the most efficient heuristics for similar scheduling problems. More specifically, due to their excellent performance (see computational evaluations by Pan and Ruiz, 2013 and Rad et al., 2009), the following heuristics are considered:

- NEHedd$^{\text{or}}$: NEHedd$^{\text{or}}$ is the NEHedd heuristic proposed by Kim (1993) for $Fm|prmu|\sum T_j$. The speed up procedure described in Section 4.3 is not applied to maintain its original version.

- NEHedd$^{\text{et}}$: Heuristic NEHedd$^{\text{or}}$ using the speed up procedure in Section 4.3. Additionally, the evaluation of total tardiness in each iteration is replaced by the evaluation of the sum of total earliness and tardiness.

- Raj: Adaptation of the Raj heuristic by Rajendran (1993), originally proposed for the $Fm|prmu|\sum C_j$ problem. To adapt the heuristic to our problem, the speed up procedure in Section 4.3 is applied, and the original evaluation of total flowtime is replaced by the evaluation of total earliness and tardiness. Additionally, the original initial order is replaced by the EDD rule.

- RZ: Adaptation of the RZ heuristic by Rajendran and Ziegler (1997) proposed for the $Fm|prmu|\sum C_j$ problem, with the initial order replaced by the EDD rule. The speed up procedure is applied, and the evaluation of total flowtime is replaced by the evaluation of total earliness and tardiness.

- RZ_LW: Adaptation of the RZ_LW heuristic by Rajendran and Ziegler (1997), originally proposed for the $Fm|prmu|\sum C_j$ problem. The speed up procedure is applied and the evaluation of total flowtime is replaced by the evaluation of total earliness and tardiness. Furthermore, the EDD rule is used as initial order.

- FRB4$_{\text{k}}$: Adaptation of the FRB4$_{\text{k}}$ heuristic by Rad et al. (2009), originally proposed for the $Fm|prmu|C_{max}$ problem. The evaluation of the makespan is replaced by the evaluation of

the total earliness and tardiness, and the speed up procedure by Taillard (1990) is replaced by the proposed one. As in the NEHedd^{et}, the original order is replaced by the EDD rule.

All heuristics are fully recoded for the $Fm|prmu|\sum E_j + \sum T_j$ problem under the same computer conditions, which means:

- Under the same computer (an Intel Core i7-3770 with 3.4 GHz and 16 GB RAM).
- Using the same programming language (algorithms have been coded in C#).
- Using the same libraries and common functions.

5.4 Indicators to evaluate the heuristics

Since each heuristic requires different CPU time, their comparison is not straightforward, as there is a trade-off between the quality of the solutions (usually measured by means of the Average Relative Percentage Deviation $ARPD$) and the computational effort required (usually measured using the Average Computational Effort ACT). These indicators are defined as follows:

$$ACT_i = \frac{\sum_{\forall j} T_{i,j}}{J} \tag{8}$$

$$ARPD_i = \frac{\sum_{\forall j} RPD2_{i,j}}{J} \tag{9}$$

with $T_{i,j}$ the average computational time (in seconds) required by heuristic i in instance j among 5 independent runs, J is the number of instances, and $ARPD_i$ is the average $RPD2_{i,j}$ of heuristic i over all instances, which is defined by Expression (10):

$$RPD2_{i,j} = \frac{OF_{i,j} - BestConst_j}{BestConst_j} \cdot 100 \tag{10}$$

with $OF_{i,j}$ the total earliness and tardiness for heuristic i in instance j and $BestConst_j$ is the best value found among the implemented constructive heuristics (see bounds in online materials).

The use of both indicators to compare heuristics is very extended in the literature. However, a direct comparison between both indicators presents several problems related to the weight of each problem size, as shown by Fernandez-Viagas and Framinan (2015c). To avoid them, in addition

to report the results in terms of ACT, we also use the average relative percentage time, $ARPT_i$ indicator of heuristic i, defined by (11) (note that 1 is added to the quotient to avoid negative numbers).

$$ARPT_i = \frac{\sum_{\forall j} RPT_{i,j}}{J} + 1 \tag{11}$$

with $RPT_{i,j}$ the relative percentage time of heuristic i in instance j, defined by Equation (12).

$$RPT_{i,j} = \frac{T_{i,j} - ACT_j}{ACT_j} \tag{12}$$

In the following, we assume that one heuristic outperforms another in an instance if both their RPD and RPT are lower. A heuristic e is thus denoted *efficient for an instance* if it outperforms the rest of heuristics in this instance. Similarly, one heuristic is labelled *efficient* is there is no other heuristic with lower values of both $ARPD$ and $ARPT$ measured over a full testbed. The set of efficient heuristics is denoted as $\mathcal{A}$.

5.5 Efficient set of heuristics

In this section, all implemented heuristics are compared using benchmark $\mathcal{B}_2$. Average results in terms of $ARPD$ are shown in Table 2 for each combination of n and m, and in Table 4 for each value of the parameters. The CPU time required by each heuristic is shown in Table 3 for each n and m. The last two rows show the ACT and the $ARPT$ of each heuristic. A summary of the results is graphically shown in Figure 8 using ACT to evaluate the computational effort, while $ARPT$ is used as indicator in Figure 9. In view of the results, the NEHedd$^{\text{et}}$ heuristic clearly outperforms the NEHedd$^{\text{or}}$ in terms of quality of the solution and computational effort. The best $ARPDs$ are clearly found by the proposed heuristic ACH4 (1.19), and by the RZ_LW heuristic (2.40). Regarding heuristics adapted from other problems, the best results are found by Raj, RZ and RZ_LW, which are either very fast heuristics, or local search methods (using dispatching rules as seed sequences). The good performance achieved by the composite heuristics RZ, RZ_LW, ACH2, ACH3, and ACH4 confirms the conclusions obtained after the analysis of the problem in Section 3 which advocated for fast heuristics employing as soon as possible local

search methods of full sequences. This fact is also confirmed by the performance of the family of heuristics FRB4_k. Each of these heuristics is outperformed in terms of quality of the solutions and computational effort by RZ and ACH3. According to Figure 9, the efficient heuristics (set $\mathcal{A}$) are: ACH1, Raj, $\text{NEHedd}^{\text{et}}$, ACH2, RZ, ACH3 and ACH4. To statistically justify this statement, we perform a Holm's procedure (Holm, 1979) with the following hypotheses:

- H_1: ACH2 = $\text{NEHedd}^{\text{or}}$.
- H_2: RZ = FRB4_2.
- H_3: RZ = FRB4_4.
- H_4: RZ = FRB4_6.
- H_5: ACH3 = FRB4_8.
- H_6: ACH3 = FRB4_{10}.
- H_7: ACH3 = FRB4_{12}.
- H_8: ACH4 = RZ_LW.

Results are shown in Table 5, where the p-values have been calculated using a non-parametric Mann-Whitney test since the normality and homoscedasticity assumptions were not confirmed (see e.g. Pan et al., 2008). Assuming a confidence of 0.95, only two hypotheses (H2 and H3) are not rejected and the proposed heuristics (ACH2, ACH3 and ACH4) can be therefore considered as statistically efficient. The heuristics of the sets $\mathcal{A}$ are shown in Figure 9.

Instance	Raj	NEHedd^{et}	NEHedd^{or}	RZ	FRB4$_2$	FRB4$_4$	FRB4$_6$	FRB4$_8$	FRB4$_{10}$	FRB4$_{12}$	RZ_LW	ACH1	ACH2	ACH3	ACH4
50x10	26.52	22.38	44.23	13.53	14.16	11.62	10.00	9.91	8.79	8.07	2.53	30.15	13.65	8.74	1.41
50x30	20.61	14.20	12.27	10.55	7.86	6.22	5.39	4.56	4.41	4.14	2.59	18.33	9.31	6.11	1.81
50x50	15.06	7.94	6.89	8.03	4.54	3.68	2.85	2.84	1.76	1.60	2.11	11.49	5.95	4.00	1.40
150x10	41.36	37.56	66.72	14.39	25.56	24.13	23.19	21.88	21.59	19.33	1.69	46.12	18.06	9.83	1.65
150x30	31.16	25.09	39.50	14.76	15.89	15.07	14.15	13.07	11.49	11.04	2.53	37.47	15.88	7.40	1.19
150x50	30.61	24.40	25.86	14.93	18.92	17.11	16.36	14.73	14.02	13.10	3.18	33.14	14.33	5.23	0.44
250x10	36.04	29.13	76.55	13.94	22.63	20.62	19.42	18.69	17.61	17.25	2.31	50.20	13.88	7.85	1.09
250x30	40.84	35.95	57.49	15.95	24.08	23.16	21.62	21.04	20.13	18.94	2.50	45.28	19.56	8.96	1.14
250x50	32.37	26.65	39.12	15.29	17.54	16.23	15.34	15.13	13.28	11.87	2.69	37.35	16.44	5.99	1.29
350x10	38.57	30.75	77.89	13.37	23.70	23.05	22.46	20.94	20.71	20.09	1.41	56.46	14.83	8.76	0.82
350x30	48.71	41.83	66.24	18.72	32.40	29.66	28.60	27.43	26.10	26.66	2.83	51.33	22.34	9.05	1.02
350x50	32.56	26.18	47.91	13.49	18.94	16.45	16.34	15.17	15.51	13.08	2.44	39.64	17.25	7.44	0.98
Average	32.87	26.84	46.72	13.91	18.85	17.25	16.31	15.45	14.61	13.76	2.40	38.08	15.12	7.45	1.19

Table 2: Relative Percentage Deviation (RPD) for the implemented heuristics under the set of instances of Vallada et al. (2008)

Instance	Raj	NEHedd^{et}	NEHedd^{or}	RZ	FRB4$_2$	FRB4$_4$	FRB4$_6$	FRB4$_8$	FRB4$_{10}$	FRB4$_{12}$	RZ_LW	ACH1	ACH2	ACH3	ACH4
50x10	0.00	0.00	0.00	0.01	0.02	0.02	0.02	0.03	0.03	0.04	0.03	0.00	0.00	0.02	0.03
50x30	0.00	0.01	0.01	0.02	0.04	0.05	0.07	0.09	0.10	0.11	0.09	0.00	0.01	0.06	0.08
50x50	0.00	0.01	0.02	0.03	0.06	0.09	0.11	0.14	0.16	0.19	0.17	0.00	0.02	0.08	0.13
150x10	0.01	0.06	0.10	0.13	0.26	0.41	0.55	0.68	0.80	0.91	0.98	0.00	0.09	0.56	0.95
150x30	0.04	0.16	0.35	0.41	0.82	1.30	1.75	2.17	2.56	2.92	3.42	0.01	0.28	1.74	3.20
150x50	0.07	0.26	0.61	0.72	1.44	2.31	3.12	3.89	4.58	5.26	5.84	0.02	0.49	2.64	5.27
250x10	0.06	0.21	0.46	0.56	1.12	1.81	2.47	3.09	3.68	4.24	4.93	0.01	0.38	2.84	4.73
250x30	0.18	0.64	1.60	1.82	3.67	5.92	8.00	9.93	11.80	13.58	19.33	0.03	1.25	10.24	18.88
250x50	0.31	1.13	2.80	3.24	6.50	10.50	14.22	17.74	21.06	24.30	33.00	0.06	2.21	18.64	32.80
350x10	0.15	0.53	1.24	1.52	3.02	4.91	6.72	8.46	10.12	11.73	14.74	0.02	1.03	8.88	14.33
350x30	0.46	1.69	4.30	4.88	9.90	16.15	21.96	27.51	32.70	37.91	55.90	0.06	3.33	31.85	61.23
350x50	0.81	2.98	7.54	8.60	17.37	28.02	38.17	47.65	57.07	65.66	95.51	0.09	5.88	55.56	96.84
ACT	0.17	0.64	1.59	1.83	3.68	5.96	8.10	10.12	12.06	13.90	19.49	0.03	1.25	11.09	19.87
$ARPT$	0.03	0.11	0.24	0.30	0.61	0.96	1.27	1.57	1.85	2.12	2.43	0.01	0.21	1.33	2.32

Table 3: Average Computational Effort (ACT) (in seconds) and Average Relative Percentage Time ($ARPT$) of the implemented heuristics under the set of instances of Vallada et al. (2008)

Parameter		Raj	NEHedd^{et}	NEHedd^{or}	RZ	FRB4$_2$	FRB4$_4$	FRB4$_6$	FRB4$_8$	FRB4$_{10}$	FRB4$_{12}$	RZ_LW	ACH1	ACH2	ACH3	ACH4
T	0.2	49.71	38.62	92.83	16.45	28.60	26.05	24.71	23.59	21.92	19.90	1.60	65.76	25.09	13.01	2.49
T	0.4	31.61	29.44	36.71	15.93	20.61	19.15	18.10	17.18	16.76	16.40	2.72	36.04	14.30	6.15	0.71
T	0.6	17.28	12.46	10.63	9.36	7.34	6.55	6.12	5.58	5.16	5.00	2.88	12.44	5.98	3.19	0.36
R	0.2	18.40	16.25	21.39	9.35	11.14	10.06	9.60	9.31	8.95	8.83	2.31	22.21	8.93	5.05	0.43
R	0.6	24.85	18.31	35.67	12.50	12.01	11.05	10.27	10.02	9.63	9.26	2.59	28.50	11.52	6.62	0.86
R	1	55.35	45.95	83.11	19.89	33.40	30.64	29.07	27.02	25.27	23.20	2.30	63.53	24.92	10.68	2.27
n	50	20.73	14.84	21.13	10.70	8.85	7.17	6.08	5.77	4.98	4.60	2.41	19.99	9.64	6.28	1.54
n	150	34.38	29.02	44.03	14.69	20.12	18.77	17.90	16.56	15.70	14.49	2.47	38.91	16.09	7.49	1.09
n	250	36.42	30.57	57.72	15.06	21.42	20.00	18.79	18.29	17.00	16.02	2.50	44.28	16.63	7.60	1.17
n	350	39.95	32.92	64.01	15.19	25.01	23.05	22.47	21.18	20.77	19.94	2.23	49.14	18.14	8.42	0.94
m	10	35.62	29.95	66.34	13.81	21.51	19.85	18.77	17.85	17.17	16.19	1.99	45.73	15.11	8.80	1.24
m	30	35.33	29.26	43.88	15.00	20.06	18.53	17.44	16.52	15.53	15.20	2.61	38.10	16.77	7.88	1.29
m	50	27.65	21.29	29.95	12.93	14.99	13.37	12.72	11.97	11.14	9.91	2.60	30.40	13.49	5.67	1.03

Table 4: Average relative percentage deviation ($ARPD$) for each heuristic grouped by the values of the parameters

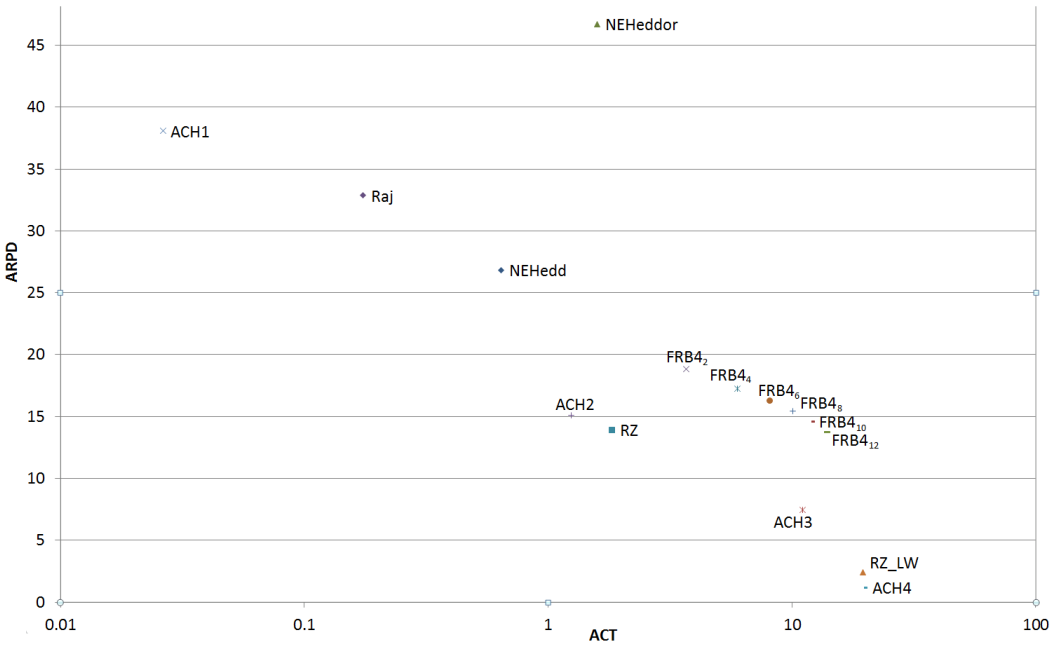

Figure 8: $ARPD$ vs ACT of implemented heuristics. X-axis (ACT in seconds) is shown logarithmic scale.

5.6 Comparison among efficient heuristics

As there is a trade-off between quality of the solution and computational effort, heuristics in set $\mathcal{A}$ cannot be directly compared in terms of $ARPD$ due to their different computational efforts. In this Section, they are included as initial solution for one of the best metaheuristic for this problem, i.e. the ILS by M'Hallah (2014), replacing the original seed sequence of the metaheuristic (EDD rule). Thus, the metaheuristic is run using eight different initial sequences (EDD rule and each heuristic in set $\mathcal{A}$) where the EDD rule is included in the comparison as it is the original seed sequence of the metaheuristic. Each variation of the metaheuristic is run under the same computational conditions described in Section 5.3 using the benchmark in Section 5.1. In this case, five runs are performed per instance and the average values are recorded. The variations of the ILS are stopped depending on the size of the problem according to expression $n \cdot m \cdot t/2$ (milliseconds) where $t = 5, 10, 15, 20, 25, 30$ (see e.g. Ruiz and Stützle, 2007 for a similar stopping criterion). Obviously, the CPU time required by each heuristic is included in the CPU time of the metaheuristic, i.e. the clock starts before applying the heuristic. Results of the ILS metaheuristic using different heuristics as initial solution are shown in terms of $ARPD$ in Table 6. Note that

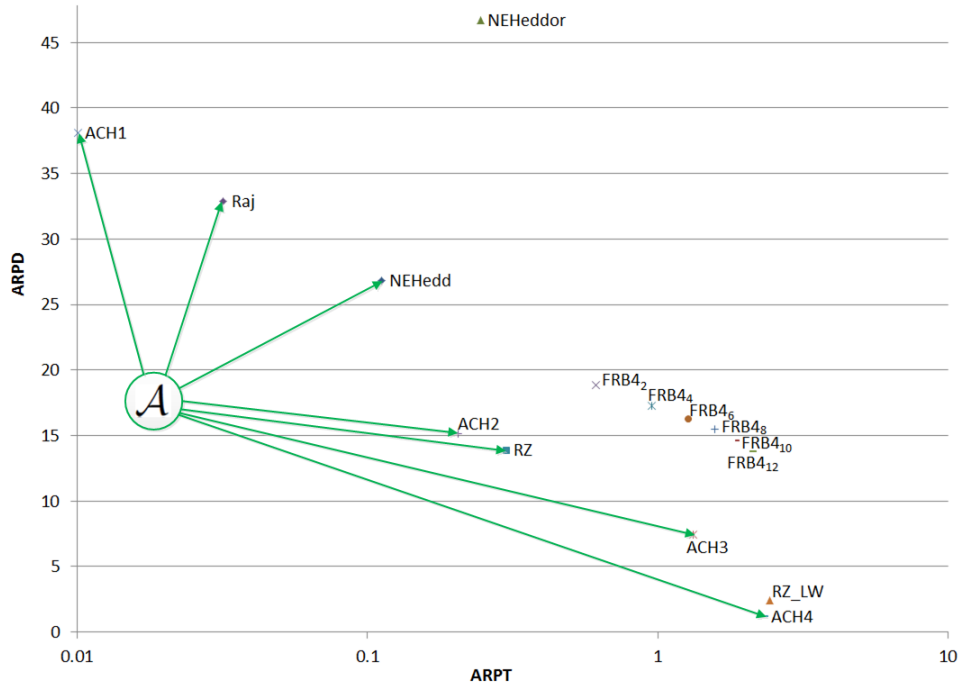

Figure 9: $ARPD$ vs $ARPT$ of implemented heuristics. X-axis (ACT in seconds) is shown logarithmic scale.

i	H_i	p-value	Mann-Whitney	$\alpha/(k-i+1)$	Holm's Procedure
1	ACH2 = NEHedd^{or}	0.000	R	0.0063	R
2	ACH3 = FRB4$_8$	0.000	R	0.0071	R
3	ACH3 = FRB4$_{10}$	0.000	R	0.0083	R
4	ACH3 = FRB4$_{12}$	0.000	R	0.0100	R
5	ACH4 = RZ_LW	0.000	R	0.0125	R
6	RZ = FRB4$_2$	0.001	R	0.0167	R
7	RZ = FRB4$_4$	0.069		0.0250	
8	RZ = FRB4$_6$	0.690		0.0500	

Table 5: Holm's procedure.

Parameter t	EDD rule	Raj	NEHedd^{et}	RZ	ACH1	ACH2	ACH3	ACH4
5	3.26	4.58	4.77	3.60	2.97	2.72	1.90	1.39
10	2.20	2.91	3.09	2.43	1.87	1.94	1.56	1.32
15	2.11	2.80	2.96	2.33	1.82	1.85	1.46	1.22
20	2.03	2.69	2.84	2.24	1.76	1.75	1.41	1.16
25	1.95	2.60	2.75	2.17	1.70	1.69	1.34	1.11
30	1.94	2.58	2.71	2.14	1.67	1.67	1.31	1.08

Table 6: Average relative percentage deviation ($ARPD$) of the metaheuristic ILS using different heuristics as initial solution

using the original seed sequence (EDD rule) in the metaheuristic outperforms several other initial sequences (Raj, NEHedd^{et} and RZ). However, the best $ARPDs$ are found when embedding the proposed heuristics (ACH1, ACH2, ACH3 and ACH4) in the ILS metaheuristic being e.g. 1.87, 1.94, 1.56 and 1.32 respectively the $ARPD$ of these heuristics for $t = 10$, as compared to 2.20 obtained by the EDD rule. The best value is found using ACH4 as initial solution regardless the stopping criteria, being 1.08 the lowest $ARPD$ found for $t = 30$. Additionally, in order to confirm the excellent results found by the ILS using the ACH4 heuristic as seed sequence, a Holm's procedure is carried out comparing the ILS both with the ACH4 heuristic and with the EDD rule. More specifically, the hypotheses tested are:

- H_1: For $t = 5$, ILS(ACH4) = ILS(EDD rule).
- H_2: For $t = 10$, ILS(ACH4) = ILS(EDD rule).
- H_3: For $t = 15$, ILS(ACH4) = ILS(EDD rule).
- H_4: For $t = 20$, ILS(ACH4) = ILS(EDD rule).
- H_5: For $t = 25$, ILS(ACH4) = ILS(EDD rule).
- H_6: For $t = 30$, ILS(ACH4) = ILS(EDD rule).

Results of the Holm's procedure are shown in Table 7. Each p-value is equal to 0.000 and therefore, each hypothesis is rejected statistically, confirming the previous results.

i	H_i	p-value	Mann-Whitney	$\alpha/(k-i+1)$	Holm's Procedure
1	ILS(ACH4) = ILS(EDD rule) (for $t = 5$)	0.000	R	0.0083	R
2	ILS(ACH4) = ILS(EDD rule) (for $t = 10$)	0.000	R	0.0100	R
3	ILS(ACH4) = ILS(EDD rule) (for $t = 15$)	0.000	R	0.0125	R
4	ILS(ACH4) = ILS(EDD rule) (for $t = 20$)	0.000	R	0.0167	R
5	ILS(ACH4) = ILS(EDD rule) (for $t = 25$)	0.000	R	0.0250	R
6	ILS(ACH4) = ILS(EDD rule) (for $t = 30$)	0.000	R	0.0500	R

Table 7: Holm's procedure for comparisons with metaheuristics.

6 Conclusions

In this paper, we have addressed the PFSP with a just-in-time objective. The problem has been analysed in detail and several properties have been established for extreme values of the due dates. Particularly, extremely tight due dates lead to a problem similar to $Fm|prmu|\sum C_j$ while extremely loose due dates lead to the opposite problem, i.e. $Fm|prmu|\sum -C_j$. By incorporating this knowledge, we propose four different heuristics. Firstly, a fast constructive heuristic, ACH1, inserts the jobs one by one at the end of the partial sequence based on an dynamic index. This index is automatically calculated in each iteration depending on the idle times, completion times, earliness and tardiness of the jobs. Then, three composite heuristics ACH2, ACH3 and ACH4 are proposed by incorporating three different local search procedures after ACH1.

The proposed heuristics have been compared under a complete set of instances with the best heuristic for the problem as well as with adaptations of efficient heuristics for similar scheduling problems. The computational results show the excellent performance of the proposed algorithms. In fact, the heuristics ACH1, Raj, NEHedd^{et}, ACH2, RZ, ACH3 and ACH4 are established as efficient, being the ACH4 the heuristic with the lowest $ARPD$ (1.19).

Finally, the impact of the efficient heuristics is evaluated including them as seed sequences for one of the best metaheuristic for the problem, and for six different stopping criteria. As a result, the four proposed heuristics statistically outperform every other heuristic, thus establishing a new state-of-the-art of approximate solutions for the problem.

References

Bagchi, U., Sullivan, R. S., and Chang, Y. (1986). Minimizing mean absolute deviation of completion times about a common due date. *Naval research logistics quarterly*, 33(2):227–240.

Baker, K. R. and Scudder, G. D. (1990). Sequencing with earliness and tardiness penalties. a review. *Operations Research*, 38(1):22–36.

Chandra, P., Mehta, P., and Tirupati, D. (2009). Permutation flow shop scheduling with earliness and tardiness penalties. *International Journal of Production Research*, 47(20):5591–5610.

Dong, X., Chen, P., Huang, H., and Nowak, M. (2013). A multi-restart iterated local search algorithm for the permutation flow shop problem minimizing total flow time. *Computers & Operations Research*, 40(2):627–632.

Fernandez-Viagas, V. and Framinan, J. (2015a). A bounded-search iterated greedy algorithm for the distributed permutation flowshop scheduling problem. *International Journal of Production Research*, 53(4):1111–1123.

Fernandez-Viagas, V. and Framinan, J. (2015b). Efficient non-population-based algorithms for the permutation flowshop scheduling problem with makespan minimisation subject to a maximum tardiness. *Computers & Operations Research*, 64(0):86 – 96.

Fernandez-Viagas, V. and Framinan, J. (2015c). A new set of high-performing heuristics to minimise flowtime in permutation flowshops. *Computers & Operations Research*, 53:68–80.

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

Fernandez-Viagas, V. and Framinan, J. M. (2014). On insertion tie-breaking rules in heuristics for the permutation flowshop scheduling problem. *Computers & Operations Research*, 45(0):60 – 67.

Framinan, J., Gupta, J., and Leisten, R. (2004). A review and classification of heuristics for permutation flow-shop scheduling with makespan objective. *Journal of the Operational Research Society*, 55(12):1243–1255.

Framinan, J. and Leisten, R. (2008). Total tardiness minimization in permutation flow shops: A simple approach based on a variable greedy algorithm. *International Journal of Production Research*, 46(22):6479–6498.

Holm, S. (1979). A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6:65–70.

Józefowska, J. (2007). *Just-in-time Scheduling*. Springer, New York.

Kim, Y.-D. (1993). Heuristics for flowshop scheduling problems minimizing mean tardiness. *Journal of the Operational Research Society*, 44(1):19–28.

Lauff, V. and Werner, F. (2004). Scheduling with common due date, earliness and tardiness penalties for multimachine problems: A survey. *Mathematical and Computer Modelling*, 40(5-6):637–655.

Li, X., Wang, Q., and Wu, C. (2009). Efficient composite heuristics for total flowtime minimization in permutation flow shops. *OMEGA, The International Journal of Management Science*, 37(1):155–164.

Li, X. and Wu, C. (2005). An efficient constructive heuristic for permutation flow shops to minimize total flowtime. *Chinese Journal of Electronics*, 14(2):203–208.

Madhushini, N., Rajendran, C., and Deepa, Y. (2009). Branch-and-bound algorithms for scheduling in permutation flowshops to minimize the sum of weighted flowtime/sum of weighted tardiness/sum of weighted flowtime and weighted tardiness/sum of weighted flowtime, weighted

tardiness and weighted earliness of jobs. *Journal of the Operational Research Society*, 60(7):991–1004.

M'Hallah, R. (2014). An iterated local search variable neighborhood descent hybrid heuristic for the total earliness tardiness permutation flow shop. *International Journal of Production Research*, 52(13):3802–3819.

Moslehi, G., Mirzaee, M., Vasei, M., Modarres, M., and Azaron, A. (2009). Two-machine flow shop scheduling to minimize the sum of maximum earliness and tardiness. *International Journal of Production Economics*, 122(2):763–773.

Naderi, B. and Ruiz, R. (2010). The distributed permutation flowshop scheduling problem. *Computers & Operations Research*, 37(4):754–768.

Nawaz, M., Enscore Jr., E., and Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *OMEGA, The International Journal of Management Science*, 11(1):91–95.

Pan, Q.-K. and Ruiz, R. (2013). A comprehensive review and evaluation of permutation flowshop heuristics to minimize flowtime. *Computers & Operations Research*, 40(1):117–128.

Pan, Q.-K., Tasgetiren, M., and Liang, Y.-C. (2008). A discrete differential evolution algorithm for the permutation flowshop scheduling problem. *Computers and Industrial Engineering*, 55(4):795–816.

Pan, Q.-K. and Wang, L. (2012). Effective heuristics for the blocking flowshop scheduling problem with makespan minimization. *OMEGA, The International Journal of Management Science*, 40(2):218–229.

Pinedo, M. (1995). *Scheduling: Theory, Algorithms and Systems*. Prentice Hall.

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

Rad, S. F., Ruiz, R., and Boroojerdian, N. (2009). New high performing heuristics for minimizing makespan in permutation flowshops. *OMEGA, The International Journal of Management Science*, 37(2):331–345.

Rajendran, C. (1993). Heuristic algorithm for scheduling in a flowshop to minimize total flowtime. *International Journal of Production Economics*, 29(1):65–73.

Rajendran, C. and Ziegler, H. (1997). An efficient heuristic for scheduling in a flowshop to minimize total weighted flowtime of jobs. *European Journal of Operational Research*, 103:129–138.

Reza Hejazi, S. and Saghafian, S. (2005). Flowshop-scheduling problems with makespan criterion: A review. *International Journal of Production Research*, 43(14):2895–2929.

Rios-Mercado, R. and Bard, J. (1998). Heuristics for the flow line problem with setup costs. *European Journal of Operational Research*, 110(1):76–98.

Ruiz, R. and Maroto, C. (2005). A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research*, 165(2):479–494.

Ruiz, R. and Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177(3):2033–2049.

Schaller, J. and Valente, J. (2013). A comparison of metaheuristic procedures to schedule jobs

in a permutation flow shop to minimise total earliness and tardiness. *International Journal of Production Research*, 51(3):772–779.

Shabtay, D. and Steiner, G. (2012). Scheduling to maximize the number of just-in-time jobs: A survey. In Rios-Mercado, R. Z. and Rios-Solis, Y. A., editors, *Just-in-Time Systems*, Springer Optimization and Its Applications, pages 3–20. Springer New York.

Storer, R., Wu, S., and Vaccari, R. (1992). New search spaces for sequencing problems with application to job shop scheduling. *Management Science*, 38:1495–1509.

Taillard, E. (1990). Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research*, 47(1):65–74.

Vallada, E. and Ruiz, R. (2010). Genetic algorithms with path relinking for the minimum tardiness permutation flowshop problem. *OMEGA, The International Journal of Management Science*, 38(1-2):57–67.

Vallada, E., Ruiz, R., and Minella, G. (2008). Minimising total tardiness in the m-machine flowshop problem: A review and evaluation of heuristics and metaheuristics. *Computers & Operations Research*, 35(4):1350–1373.

Vollman, T. E., Berry, W. L., and Wybark, D. C. (1997). *Manufacturing Planning and Control Systems*. McGraw-Hill, New York.

Zegordi, S., Itoh, K., and Enkawa, T. (1995). A knowledgeable simulated annealing scheme for the early/tardy flow shop scheduling problem. *International Journal of Production Research*, 33(5):1449–1466.