An effective iterated greedy algorithm for the mixed no-idle permutation flowshop scheduling problem

Quan-Ke Pan[a], Rubén Ruiz[b,*]

[a] *State Key Laboratory of Synthetical Automation for Process Industries (Northeastern University), Shenyang, 110819, PR China. College of Computer Science, Liaocheng University, Liaocheng, 252059, PR China.*

[b] *Grupo de Sistemas de Optimización Aplicada, Instituto Tecnológico de Informática, Ciudad Politécnica de la Innovación, Edifico 8G, Acc. B. Universitat Politècnica de València, Camino de Vera s/n, 46021, València, Spain.*

Abstract

In the no-idle flowshop, machines cannot be idle after finishing one job and before starting the next one. Therefore, start times of jobs must be delayed to guarantee this constraint. In practice machines show this behavior as it might be technically unfeasible or uneconomical to stop a machine in between jobs. This has important ramifications in the modern industry including fiber glass processing, foundries, production of integrated circuits and the steel making industry, among others. However, to assume that all machines in the shop have this no-idle constraint is not realistic. To the best of our knowledge, this is the first paper to study the mixed no-idle extension where only some machines have the no-idle constraint. We present a mixed integer programming model for this new problem and the equations to calculate the makespan. We also propose a set of formulas to accelerate the calculation of insertions that is used both in heuristics as well as in the local search procedures. An effective iterated greedy (IG) algorithm is proposed. We use an NEH-based heuristic to construct a high quality initial solution. A local search using the proposed accelerations is employed to emphasize intensification and exploration in the IG. A new destruction and construction procedure is also shown. To evaluate the proposed algorithm, we present several adaptations of other well-known and recent metaheuristics for the problem and conduct a comprehensive set of

*Corresponding author. Tel: +34 96 387 70 07. Fax: +37 96 387 74 99

Email addresses: 2281393146@qq.com (Quan-Ke Pan), rruiz@eio.upv.es (Rubén Ruiz)

Preprint submitted to OMEGA *October 2, 2013*

computational and statistical experiments with a total of 1,750 instances. The results show that the proposed IG algorithm outperforms existing methods in the no-idle and in the mixed no-idle scenarios by a significant margin.

Keywords: flowshop, no-idle, heuristics, iterated greedy, local search

1. Introduction

It has been almost 60 years since the seminal work about the two machine flowshop problem with makespan minimization criterion by Johnson (1954). Actually, in the scheduling literature this paper has been regarded as the first in the field (with the possible exception of the paper by Salveson, 1952). In a flowshop problem we deal with a set N of n jobs, modeling client orders of different products to be manufactured, that have to be produced on a set M of m machines. The layout of the machines in the production shop is in series, i.e., we have first machine 1, then machine 2 and so on until machine m. All jobs must visit the machines in the same processing sequence. This sequence can be, without loss of generality, $\{1, \ldots, m\}$. Therefore, a job is composed of m tasks or operations. Each task j, $j = \{1, \ldots, n\}$ requires a known, deterministic and non-negative amount of time at each machine i, $i = \{1, \ldots, m\}$. This amount is referred to as processing time and denoted by p_{ij}. The objective is to find a processing sequence of all jobs at each machine so that a given criterion is optimized. There are as many possible sequences of jobs as permutations and this permutation can change from machine to machine which results in a search space of $(n!)^m$ non-delay schedules for the Flowshop Scheduling Problem (FSP). Given this huge search space, most of the time, the problem simplified by forbidding job passing, i.e., once a permutation of jobs is obtained for the first machine, it is maintained for all other machines, reducing the search space to $n!$ solutions. This somewhat simpler problem is referred to as the Permutation Flowshop Scheduling Problem or PFSP. Following the work of Johnson (1954), the most studied optimization criterion is the minimization of the maximal job completion time or makespan ($C_{\max}$) which corresponds to the time at which the last job in the sequence is finished at the last machine in the shop. The PFSP with makespan criterion is denoted as $F/prmu/C_{\max}$, following the accepted three field notation of Graham et al. (1979). Reviews about flowshop scheduling with this criterion are given by Framinan et al. (2004), Ruiz and Maroto (2005), Hejazi and Saghafian (2005) and Gupta and Stafford (2006). The literature about flowshop scheduling is huge. Not only does each studied objective span a relatively large sub-field in itself with hundreds of references, like total tardiness minimization (see Vallada et al., 2008), flowtime optimization (Pan and Ruiz, 2013) or multiobjective (Minella et al., 2008), but also problem extensions and variations abound. It is safe to say that the literature of flowshop scheduling and variants comprises thousands of papers.

One of the seldom studied extensions of the flowshop is the no-idle version. In the no-idle permutation flowshop (NPFSP), machines are not allowed to sit idle after they have started processing the first job in the sequence. The no-idle condition appears in production environments where setup times or operating costs of machines are so high that shutting down machines after the initial setup is not cost-effective. Idle times might also not be allowed on machines due to technological constraints. More specifically, in the no-idle scenario, a machine must process all jobs in the sequence without interruptions. Therefore, if needed, the start of some jobs is delayed so as to ensure the no-idle constraint. Examples of no-idle situations appear in the steppers used in the production of integrated circuits through photolithography. These fixtures are so expensive that idling is avoided at all costs. The production of ceramic frits is an example where idling is technologically impossible due to the usage of special fusing ovens (called kilns) that burn at extreme temperatures. These ovens need a continuous thermal mass and therefore, idling is not allowed. Some other examples are found in fiber glass processing (Kalczynski and Kamburowski, 2005), and foundries (Saadani et al., 2003) amongst others. Ruiz et al. (2009) and Goncharov and Sevastyanov (2009) published recent reviews about the NPFSP or $F/prmu, no-idle/C_{\max}$.

The current situation is that the no-idle constraint has been so far considered all or nothing in the flowshop literature, i.e., either we have a regular idle flowshop where idle times are allowed on all machines or all machines have the no-idle constraint in the NPFSP. Real life production shops are mixed and most machines permit idle times whereas some do not accept idle times. Surprisingly, this realistic mixed no-idle flowshop problem or MNPFSP has not been studied in the literature before to the best of our knowledge. We denote this problem by $F/prmu, mixed\ \ no-idle/C_{\max}$. In the previous examples of integrated circuits and ceramic frit production, not all machines in the shop are no-idle. In the case of ceramic frits, only the central fusing kiln has the no-idle constraint. Other examples arise in the steelmaking industry. When producing steel, the charges of molten iron enter converter stages to reduce impurities (carbon, sulfur, silicon) through oxygen burning. These charges undergo several other refining stages where impurities are further reduced, alloys are added and other operations are carried out. Only after this phase, is the molten steel is poured into a tundish for casting. The flow of molten steel goes to the crystallizer where it solidifies into slabs. Technological constraints force the continuous flow of charges with the same crystallizer and caster. This is where the no-idle constraint appears. All other stages do not have this no-idle constraint. There are many other examples in real-life factories. As a matter of fact, the authors are not aware of any real example in which all the machines in a flowshop have the no-idle constraint. Therefore, the MNPFSP is a more realistic problem which has not been studied before and is thus the motivation for this research. The PFSP is known to be $\mathcal{NP}$-Complete in the strong sense for more than two machines and makespan criterion (Garey et al., 1976). Similarly, the NPFSP was shown to belong to the same complexity class for three or more machines by Baptiste and Hguny (1997). As a result the new MNPFSP studied in this paper is also $\mathcal{NP}$-Hard in the strong sense. The rest of the paper is divided into five more sections. In the next section we review the literature mainly in the no-idle flowshop. Section 3 introduces the MNPFSP in more detail. We present a mixed integer programming model, the formulae to calculate the makespan and a speed-up method for the efficient calculation of the insertion neighborhood. Section 4 deals with the proposed Iterated Greedy method. In section 5 we present a comprehensive computational and statistical campaign to test the proposed methodology. Finally, section 6 concludes the paper and provides some avenues for further research.

2. Literature review

As stated, the MNPFSP has not been studied before. As a result, we focus our summarized review in the no-idle flowshop where all machines have the no-idle constraint. The NPFSP was first studied by Adiri and Pohoryles (1982) where polynomial time algorithms were proposed for special cases of the NPFSP mainly with two machines and total completion time criterion. Some amendments to this paper were carried out by Čepek et al. (2000). The $C_{\max}$ objective in the NPFSP was studied for the first time by Vachajitpan (1982). The author presented mathematical models and branch and bound methods for small instances. Baptiste and Hguny (1997) also presented a branch and bound method for the m-machine NPFSP and makespan criterion whereas the three machine problem was studied by Narain and Bagga (2003)

also with mathematical models and exact approaches. To date, no effective exact approach has been proposed for the NPFSP and rarely do any published results solve problems with more than a handful of jobs. As a result of this, the focus has been on heuristics for the problem. Some of the early heuristic methods were presented by Woollam (1986) that took some existing heuristics and recalcuated their produced solutions eliminating idle times and doing some simple adjacent pairwise exchange moves on the results. The adaptation of the NEH heuristic of Nawaz et al. (1983) produced the best results. Saadani et al. (2001) presented a method based on heuristics for the traveling salesman problem denoted as SGM. This research was later published in paper form in Saadani et al. (2005). The three machine case was studied by Saadani et al. (2003) to be improved on later by Kamburowski (2004). Heuristics for special cases with dominating machines are studied by Narain and Bagga (2005a,b). The general m-machine NPFSP with makespan criterion has been approached with successful heuristics by several authors. For example, Kalczynski and Kamburowski (2005) presented a method based on Johnson's heuristic, denoted as KK that was shown to outperform an adaptation of the NEH heuristic to the no-idle setting and the method of Saadani et al. (2005). A local search insertion method proposed by Baraz and Mosheiov (2008) is also shown to outperform that of Saadani et al. (2005) and is denoted by GH_BM.
Ruiz et al. (2009) presented a comprehensive comparison of heuristic methods, along with adaptations of the NEH method and the best heuristics proposed for the PFSP by Rad et al. (2009). The authors also presented an improved GH_BM method. All methods were tested with and without the accelerations of the insertion neighborhood presented by Pan and Wang (2008a,b). The results of the comprehensive computational and statistical campaign with a set of 250 instances were clear: the adapted method FRB3 of Rad et al. (2009) and the improved GH_BM2 version, both with accelerations produced the best results.
As regards metaheuristics, the first papers are by Pan and Wang (2008a,b). In the first, the authors present a discrete particle swarm optimization method, referred to as HDPSO. In the second a discrete differential evolution method is presented (DDE). Both papers are heavily based on insertion local search and an important result is given: an acceleration of the calculation of the exploration of this neighborhood. Similar to what Taillard (1990) did, the authors explain a set of calculations to reduce the complexity of the calculation of a pass in the insertion neighborhood from $\mathcal{O}(n^3m)$ to $\mathcal{O}(n^2m)$ in the NPFSP. The authors hybridized their methods with the Iterated Greedy

algorithm of Ruiz and Stützle (2007) and demonstrated in computational tests, using the instances of Taillard (1990), a clear superiority over the algorithms presented in Saadani et al. (2005) and in Kalczynski and Kamburowski (2005). However, Ruiz et al. (2009) also tested HDPSO and DDE, along with a simple adaptation of the IG of Ruiz and Stützle (2007) and showed, in a more comprehensive benchmark of 250 instances and through detailed statistical tests, that the simple IG produces better results than the HDPSO and DDE hybrids.

More recently, Deng and Gu (2012) published a hybrid discrete differential evolution method (HDDE). This method has many similarities to those of Pan and Wang (2008b) and Ruiz and Stützle (2007). Basically, a different initialization based on an improvement of the NEH and a modified insertion local search is used. The 250 instances of Ruiz et al. (2009) are used. According to their reimplementations, the results show that the new presented HDDE is better than the IG adaptation of Ruiz et al. (2009) and also the HDPSO and DDE of Pan and Wang (2008a,b). Also recently, Fatih Tasgetiren et al. (2013a) have presented a variable iterated greedy and differential evolution hybrid. The algorithm presented is shown to outperform that of Deng and Gu (2012). A side paper is that of Fatih Tasgetiren et al. (2013b) where methods are presented but for the minimization of the total tardiness criterion.

As we can see, the mixed no-idle flowshop has not been studied yet, despite being a more realistic problem. Furthermore, most modern high-performing methods for the pure no-idle version are based on the accelerated insertion neighborhood and on variants of the Iterated Greedy of Ruiz and Stützle (2007). As a matter of fact, IG is being applied to many flowshop variants like setup times (Ruiz and Stützle, 2008), blocking (Ribas et al., 2011), no-wait (Pan et al., 2008b), non-permutation (Ying, 2008), tardiness criterion (Framinan and Leisten, 2008) and multiobjective (Minella et al., 2011) as well as in many other scheduling problems. Therefore, pursuing the IG avenue for the research of the new mixed no-idle flowshop, along with the accelerations of the insertion neighborhood is the most logical step.

3. The mixed no-idle permutation flowshop problem

The no-idle flowshop differs from the regular PFSP in that no idle time exists in between any two consecutive tasks at machines. Extending the previous notation of the PFSP we denote as o_{ij} the operation of the task i of job j, i.e., the processing of job j by machine i. Similarly, C_{ij} is the completion time of this task j at machine i. In general, we have a permutation π of the n jobs and $\pi_{(j)}$ denotes the job that occupies the $j-$th position in the permutation. In the regular PFSP the following condition holds for jobs occupying consecutive positions in the permutation: $C_{i,\pi_{(j)}} \geq C_{i,\pi_{(j-1)}} + p_{i,\pi_{(j)}}$. In the no-idle flowshop, this inequality is transformed into an equality: $C_{i,\pi_{(j)}} = C_{i,\pi_{(j-1)}} + p_{i,\pi_{(j)}}$. By joining these two properties we have the mixed no-idle flowshop or MNPFSP. We define the subset of no-idle machines as $M' \subseteq M$ with m' no-idle machines. All other machines not in M' are regular idle machines. Note that all other common flowshop assumptions apply (Baker, 1974): (1) All jobs are independent and available for processing at time 0. (2) Machines are continuously available and never break down. (3) Machines can only process one task at a time. (4) A job can only be processed by one machine at a time. (5) Tasks are processed without interruptions. (6) Setup times are either independent from the sequence and included in the processing times or simply ignored. (7) There is an infinite in-process storage capacity between machines.

With the previous definitions we propose the following mixed integer linear programming model.

3.1. A mixed linear integer program

The decision variables are the typical ones in a permutation problem (Naderi and Ruiz, 2010):

$$
\begin{aligned}
X_{j,k} &= \begin{cases} 1, & \text{if job } j \text{ occupies position } k \text{ of the sequence} \\ 0, & \text{otherwise} \end{cases} \\
C_{i,k} &= \text{Completion time of job in position } k \text{ on machine } i \\
C_{\max} &= \text{Maximum completion time or makespan}
\end{aligned}
$$

The objective function is the minimization of the makespan, which is equivalent to the time at which the job occupying the last position of the permutation finishes at the last machine:

$$\min C_{\max} = C_{m,n}$$

Subject to the following constraints:

$$\sum_{k=1}^{n} X_{j,k} = 1, \qquad j = 1, \ldots, n \tag{1}$$

$$\sum_{j=1}^{n} X_{j,k} = 1, \qquad k = 1, \ldots, n \tag{2}$$

$$C_{1,k} \geq \sum_{j=1}^{n} X_{j,1} \cdot p_{1,j} \qquad k = 1, \ldots, n \tag{3}$$

$$C_{i,k} \geq C_{i-1,k} + \sum_{j=1}^{n} X_{j,k} \cdot p_{i,j} \qquad k = 1, \ldots, n, i = 2, \ldots, m \tag{4}$$

$$\begin{cases} C_{i,k} = C_{i,k-1} + \sum_{j=1}^{n} X_{j,k} \cdot p_{i,j}, & \text{if } i \in M' \\ C_{i,k} \geq C_{i,k-1} + \sum_{j=1}^{n} X_{j,k} \cdot p_{i,j}, & \text{otherwise} \end{cases} \quad k = 2, \ldots, n, i = 1, \ldots, m \tag{5}$$

$$C_{i,k} \geq 0 \qquad k = 1, \ldots, n, i = 1, \ldots, m \tag{6}$$

$$X_{i,k} \in \{0, 1\} \qquad k = 1, \ldots, n, i = 1, \ldots, m \tag{7}$$

Constraints (1) and (2) ensure that each job occupies exactly one position in the permutation and that each position in the permutation is occupied by exactly one job. Constraint set (3) controls the completion time of the job placed in the first position of the sequence. Constraints (4) force the completion times of tasks on the second and subsequent machines to be larger than the completion times of the previous tasks on previous machines plus the processing time. The core of the MNPFSP is given in constraint set (5). Here we control the completion time of a job at an idle machine so that it is exactly equal to its processing time plus the completion time of the job in the preceding position in the permutation, i.e., no idle time is allowed. However, for regular machines, it suffices to ensure that the completion time of a job is just greater to or equal than that of the preceding job plus the processing time. Finally, constraints (6) and (7) define the domains and nature of the decision variables.

3.2. Makespan calculation

As shown in Ruiz et al. (2009) and in Pan and Wang (2008a,b), calculating the makespan for the NPFSP is far from straightforward. Here we extend such calculations for the mixed no-idle version. Obviously, being a generalization, the proposed formulas reduce to those of the regular flowshop if $M' = \varnothing$ and to the no-idle flowshop if $M' = M$.

Let us suppose a permutation $\pi = \{\pi_1, \pi_2, \dots, \pi_n\}$ where $\pi_l \in N$ for $l = 1, \dots, n$ represents the jobs in the permutation. Let $S_{i,[l]}$ and $C_{i,[l]}$ denote the earliest starting time and completion time of task $o_{i,[l]}$ or the task at machine i of the job occupying position l of the permutation, respectively. We use the simplified notation $[l]$ to represent the job in position l of the permutation, i.e., π_l or $\pi_{(l)}$.

We also denote by a_i the right shift or delay in the start time of the operation l' preceding l in the permutation, i.e, the delay in $o_{i,[l']}$ where $l' = 1, 2, \dots, l-1$ in order to meet the no-idle constraint. The makespan calculation procedure consists of calculating the start and completion times of the job in the first position π_1, then π_2 and so on until job is tested in position n or π_n. The maximum completion time of the permutation, $C_{\max}(\pi)$ is obtained with the following expressions:

$$\begin{cases} S_{1,[1]} = 0 \\ C_{1,[1]} = S_{1,[1]} + p_{1,[1]} \end{cases} \tag{8}$$

$$\begin{cases} S_{i,[1]} = C_{i-1,[1]} \\ C_{i,[1]} = S_{i,[1]} + p_{i,[1]} \end{cases} \quad i = 2, \dots, m \tag{9}$$

$$\begin{cases} S_{1,[l]} = C_{1,[l-1]} \\ C_{1,[l]} = S_{1,[l]} + p_{1,[l]} \end{cases} \quad l = 2, \dots, n \tag{10}$$

$$\begin{cases} S_{2,[l]} = \max\left\{C_{2,[l-1]}, C_{1,[l]}\right\} \\ C_{2,[l]} = S_{2,[l]} + p_{2,[l]} \\ a_2 = \begin{cases} \max\left\{C_{1,[l]} - C_{2,[l-1]}, 0\right\} & \text{if machine } 2 \in M' \\ 0 & \text{otherwise} \end{cases} \end{cases} \quad l = 2, \dots, n \tag{11}$$

$$\begin{cases} S_{i,[l]} = \max\left\{C_{i,[l-1]} + a_{i-1}, C_{i-1,[l]}\right\} \\ C_{i,[l]} = S_{i,[l]} + p_{i,[l]} \\ a_i = a_{i-1} + \begin{cases} \max\left\{C_{i-1,[l]} - (C_{i,[l-1]} + a_{i-1}), 0\right\} & i \in M' \\ 0 & \text{otherwise} \end{cases} \\ \qquad i = 3, \dots, m, l = 2, \dots, n \end{cases} \tag{12}$$

$$C_{\max}(\pi) = C_{m,[n]} \tag{13}$$

From the previous formulas, (8) computes the start and completion time for operation $o_{1,[1]}$ whereas set (9) calculates the same for operations $o_{i,[1]}$,

$i = 2, \ldots, m$. Set (10) computes the start and completion times for operations $o_{1,[l]}$, $l = 2, \ldots, n$. In set (11) we calculate the start times for operations $o_{2,[l]}$ and $a_2 = \max\left\{C_{1,[l]} - C_{2,[l-1]}, 0\right\}$ is the right shift or delay in the start time of operation $o_{2,[l']}$, $l' = 1, \ldots, l-1$ to ensure that there is no idle time between the operations on machine 2 if it is a no-idle machine. On the contrary, $a_2 = 0$ if machine 2 is a regular idle machine. In set (12) a similar calculation is carried out for operations $o_{i,[l]}$, $i = 3, \ldots, m, l = 2, \ldots, n$. Note that $\max\left\{C_{i-1,[l]} - (C_{i,[l-1]} + a_{i-1}), 0\right\}$ is the right shift or delay generated by machine i (if it is a no-idle machine) and a_{i-1} is the right shift or delay generated by all upstream no-idle machines. Therefore, a_i is the total delay. Finally, equation (13) gives us the makespan value of permutation π.

Let us consider an example with four jobs and five machines, i.e., $N = \{1, 2, 3, 4\}$ and $M = \{1, 2, 3, 4, 5\}$. Machines two and four are idle machines, i.e., $M' = \{2, 4\}$. The processing times of the four jobs in the five machines are the following:

$$[p_{ij}]_{5\times4} = \begin{bmatrix} 3 & 6 & 6 & 5 \\ 4 & 5 & 6 & 5 \\ 4 & 5 & 4 & 6 \\ 3 & 4 & 5 & 4 \\ 5 & 5 & 4 & 5 \end{bmatrix}$$

Let us suppose that we have a FIFO schedule, i.e., $\pi = \{1, 2, 3, 4\}$. Using the previous formulas (8) and (9) we calculate the start and completion times for all operations of job $\pi_1 = \{1\}$ as follows: $S_{1,[1]} = 0$, $C_{1,[1]} = 3$, $S_{2,[1]} = 3$, $C_{2,[1]} = 7$, $S_{3,[1]} = 7$, $C_{3,[1]} = 11$, $S_{4,[1]} = 11$, $C_{4,[1]} = 14$, $S_{5,[1]} = 14$, $C_{5,[1]} = 19$. The next job in the sequence is $\pi_2 = \{2\}$ and the calculations of the start and completion times, using expressions (10), (11) and (12) are the following: $S_{1,[2]} = 3$, $C_{1,[2]} = 9$, $S_{2,[2]} = \max\{C_{1,[2]}, C_{2,[1]}\} = 9$, $C_{2,[2]} = 14$, $a_2 = \max\{C_{1,[2]} - C_{2,[1]}, 0\} = 2$, $S_{3,[2]} = \max\{C_{3,[1]} + a_2, C_{2,[2]}\} = 14$, $C_{3,[2]} = 19$, $a_3 = a_2 = 2$, $S_{4,[2]} = \max\{C_{4,[1]} + a_3, C_{3,[2]}\} = 19$, $C_{4,[2]} = 23$, $a_4 = a_3 + \max\{C_{3,[2]} - (C_{4,[1]} - a_3), 0\} = 5$, $S_{5,[2]} = \max\{C_{5,[1]} + a_4, C_{4,[2]}\} = 24$, $C_{5,[2]} = 29$. We can see these calculations in Figure 1.

Similarly, the start and completion times for jobs $\pi_3 = \{3\}$ and $\pi_4 = \{4\}$ are summarized as follows: $S_{1,[3]} = 9$, $C_{1,[3]} = 15$, $S_{2,[3]} = 15$, $C_{2,[3]} = 21$, $a_2 = 1$, $S_{3,[3]} = 21$, $C_{3,[3]} = 25$, $a_3 = a_2 = 1$, $S_{4,[3]} = 25$, $C_{4,[3]} = 30$, $a_4 = 2$, $S_{5,[3]} = 31$, $C_{5,[3]} = 35$. $S_{1,[4]} = 15$, $C_{1,[4]} = 20$, $S_{2,[4]} = 21$, $C_{2,[4]} = 25$, $a_2 = 0$, $S_{3,[4]} = 26$, $C_{3,[4]} = 32$, $a_3 = 0$, $S_{4,[4]} = 32$, $C_{4,[4]} = 36$, $a_4 = 2$, $S_{5,[4]} = 37$,

$C_{5,[4]} = 42$. Finally, the makespan for the permutation $\pi = \{1, 2, 3, 4\}$ is $C_{\max}(\pi) = C_{5,[4]} = 42$.

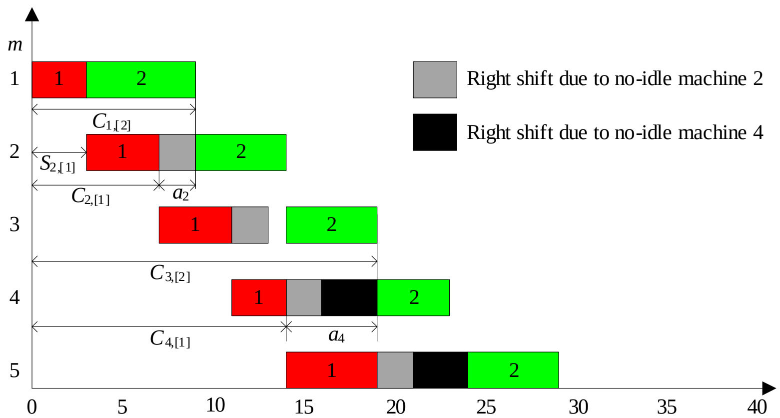

Figure 1: Makespan calculation for the first two jobs in the example.

3.3. A speed-up method for the insertion neighborhood

The insertion neighborhood is, by far, the most widely used neighborhood in the flowshop scheduling literature. Inspired by the early work of Nawaz et al. (1983), many authors have used this neighborhood with very good results. The papers of Osman and Potts (1989), Taillard (1990) or Nowicki and Smutnicki (1996) are some examples. Some of the state-of-the-art methods for the PFSP and variants employ this neighborhood (Vallada et al., 2008; Pan and Ruiz, 2013; Ruiz et al., 2009; Pan and Wang, 2008a,b; Ruiz and Stützle, 2007; Deng and Gu, 2012; Ruiz and Stützle, 2008; Ribas et al., 2011; Pan et al., 2008b; Minella et al., 2011 and many others).

The insertion neighborhood of a given permutation π of n jobs is the result of the consideration of all pairs of positions $j, k \in \{1, \ldots, n\}$ of π, $j \neq k$ where the job in position j is removed from π and inserted in position k. The resulting sequence after such a movement is

$$\pi' = \{\pi_{(1)}, \ldots, \pi_{(j-1)}, \pi_{(j+1)}, \ldots, \pi_{(k)}, \pi_{(j)}, \pi_{(k+1)}, \ldots, \pi_{(n)}\}$$

if $j < k$, or

$$\pi' = \{\pi_{(1)}, \ldots, \pi_{(k-1)}, \pi_{(j)}, \pi_{(k)}, \ldots, \pi_{(j-1)}, \pi_{(j+1)}, \ldots, \pi_{(n)}\}$$

if $j > k$. The set of insertion moves I is defined as

$$I = \{(j, k) : j \neq k,\ 1 \leq j, k \leq n \wedge j \neq k-1,\ 1 \leq j \leq n, 2 \leq k \leq n\}$$

and the insertion neighborhood of π is defined as $V(I, \pi) = \{\pi_v : v \in I\}$. The cardinality of the insertion neighborhood is $(n-1)^2$.

Since calculating the makespan for PFSP problems usually involves $\mathcal{O}(nm)$ operations, the complexity of examining the insertion neighborhood (a single pass) is $\mathcal{O}(n^3m)$. This can be computationally costly for moderate to large values of n. However, Taillard (1990) proposed the famous so called "accelerations" to reduce the complexity of the insertion neighborhood to $\mathcal{O}(n^2m)$. As a matter of fact, the accelerations were proposed for the NEH heuristic and as explained in Rad et al. (2009), the largest instances of Taillard (1993) with 500 jobs and 20 machines (500×20) require up to 30 seconds of CPU time without accelerations and as little as 77 milliseconds with accelerations on a Pentium IV computer running at 3.2 GHz. As we can see, the impact of the accelerations is huge, as the accelerated NEH requires almost 400 times less CPU time. From the results of Taillard (1990), accelerations for the calculation of the insertion neighborhood with makespan criterion have been profusely proposed for many flowshop variants. As commented, the closest references are the accelerations proposed by Pan and Wang (2008a,b) for the NPFSP.

Given the calculation of the makespan in the mixed no-idle PFSP with $\mathcal{O}(nm)$ steps of section 3.2, we now propose accelerations for the insertion neighborhood so as to reduce its complexity to $\mathcal{O}(n^2m)$.

It is well known that flowshop problems have a reversibility property (Ribas et al., 2010, among others). Under this property, the makespan of a permutation π can be calculated traversing the permutation from the first to the last job or in reverse order, i.e., from the last job in the sequence to the first. Therefore, we can divide permutation π into two partial sequences, $\pi^1 = \{\pi_{(1)}, \pi_{(2)}, \ldots, \pi_{(k)}\}$ and $\pi^2 = \{\pi_{(k+1)}, \pi_{(k+2)}, \ldots, \pi_{(n)}\}$. The forward calculation pass involves π^1 and the backward pass π^2. We denote by $S'_{i,[l]}(C'_{i,[l]})$ the starting (completion) time of operation $o_{i,[l]}$, $l = k+1, k+2, \ldots, n$ in the reverse sequence. With this, the makespan $C_{\max}(\pi)$ can be calculated as follows:

$$L_1 = C_{1,[k]} + C'_{1,[k+1]} \tag{14}$$

$$\begin{cases} L_2 = C_{2,[k]} + C'_{2,[k+1]} \\ L = \max\{L_1, L_2\} \\ a_2 = \begin{cases} \max\{L - L_2, 0\} & \text{if machine } 2 \in M' \\ 0 & \text{otherwise} \end{cases} \end{cases} \tag{15}$$

$$\begin{cases} L_i = C_{i,[k]} + a_{i-1} + C'_{i,[k+1]} \\ L = \max\{L, L_i\} \\ a_i = a_{i-1} + \begin{cases} \max\{L - L_i, 0\} & \text{if machine } i \in M' \\ 0 & \text{otherwise} \end{cases} \end{cases} \quad i = 3, \ldots, m \tag{16}$$

$$C_{\max}(\pi) = L \tag{17}$$

Let us apply the acceleration formulas to the previous example with $\pi^1 = \{1, 2\}$ and $\pi^2 = \{3, 4\}$. After calculating job 1, we calculate job 2 with the forward pass and job 3 with the reverse (also after having calculated job 4): $C_{1,[2]} = 9$, $C_{2,[2]} = 14$, $C_{3,[2]} = 19$, $C_{4,[2]} = 23$, $C_{5,[2]} = 29$ and $C'_{1,[3]} = 32$, $C'_{2,[3]} = 26$, $C'_{3,[3]} = 19$, $C'_{4,[3]} = 14$, $C'_{5,[3]} = 9$. Then the makespan is as follows:

$L_1 = C_{1,[2]} + C'_{1,[3]} = 41;$

$L_2 = C_{2,[2]} + C'_{2,[3]} = 40,\ L = \max\{L_1, L_2\} = 41,\ a_2 = \max\{L - L_2, 0\} = 1;$

$L_3 = C_{3,[2]} + a_2 + C'_{3,[3]} = 39,\ L = \max\{L, L_3\} = 41,\ a_3 = a_2 = 1;$

$L_4 = C_{4,[2]} + a_3 + C'_{4,[3]} = 38,\ L = \max\{L, L_4\} = 41,\ a_4 = a_3 + \max\{L - L_4\} = 4;$

$L_5 = C_{5,[2]} + a_4 + C'_{5,[3]} = 42,\ L = \max\{L, L_5\} = 42,\ a_5 = a_4 = 4;$

$C_{\max}(\pi) = L = 42.$

A graphical depiction of the process is given in Figure 2.

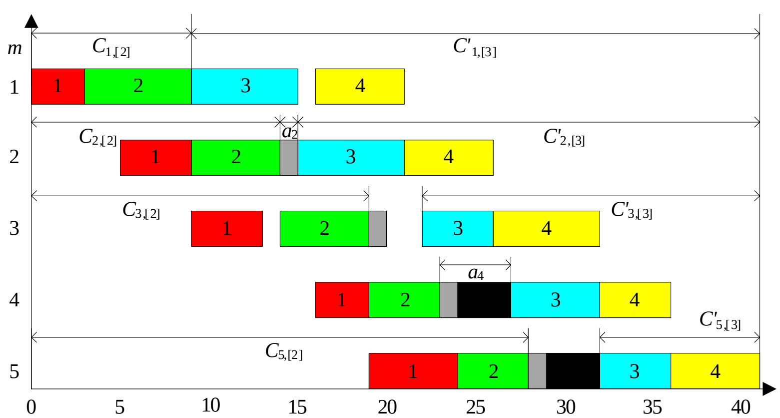

Figure 2: Calculations of sequences $\pi^1 = \{1, 2\}$ and $\pi^2 = \{3, 4\}$ for the example.

The speed-up method then consists of evaluating all permutations generated by the insertion of a single job in all possible positions of a sequence. Let $\pi = \{\pi_{(1)}, \pi_{(2)}, \ldots, \pi_{(n-1)}\}$ be a partial sequence of $n-1$ jobs. We want to insert job j_k into all possible n positions of π, generating n complete permutations. Using the formulas of Section 3.2 this would require $\mathcal{O}(n^2m)$ steps for one job or $\mathcal{O}(n^3m)$ for all jobs. With the previous formulas and the following procedure, this complexity is reduced to $\mathcal{O}(nm)$ for a single job:

Step 1. Compute $S_{i,[l]}$ and $C_{i,[l]}$ for $i = 1, 2, \ldots, m$ and $l = 1, 2, \ldots, n-1$ with the forward pass and $S'_{i,[l]}$ and $C'_{i,[l]}$ for $i = m, m-1, \ldots, 1$ and $l = n-1, n, \ldots, 1$ with the backward pass.

Step 2 For $l = 1, \ldots, n$ do the following steps:

Step 2.1 Insert job j_k into the l^{th} position of the partial sequence π and generate a full permutation $\omega = \{\pi_{(1)}, \pi_{(2)}, \ldots, \pi_{(l-1)}, j_k, \pi_{(l)}, \ldots, \pi_{(n-1)}\}$.

Step 2.2 Divide ω into two partial sequences: $\omega^1 = \{\pi_{(1)}, \pi_{(2)}, \ldots, \pi_{(l-1)}, j_k\}$ and $\omega^2 = \{\pi_{(l)}, \pi_{(l+1)}, \ldots, \pi_{(n-1)}\}$. Note that $\omega^1 = \varnothing$ if $l = 1$ and $\omega^2 = \varnothing$ if $l = n$.

Step 2.3 Calculate the starting and completion time for the last job j_k of ω^1 after obtaining $S_{i,[l-1]}$ and $C_{i,[l-1]}$ in Step 1 with formulas (8) to (13).

Step 2.4 Calculate the makespan of ω using equations (14) to (17).

Step 1 has a computational complexity of $\mathcal{O}(nm)$. Step 2 contains a loop of n steps where each step has a complexity of $\mathcal{O}(m)$. Therefore, Step 2 has a $\mathcal{O}(nm)$ complexity as a whole. This means that testing a job in all possible n positions of a sequence has a computational complexity of $\mathcal{O}(nm)$. Since there are n jobs to test, the full examination of the insertion neighborhood needs $\mathcal{O}(n^2m)$ steps.

4. Iterated Greedy approach

The first application of the Iterated Greedy for flowshop problems was given by Ruiz and Stützle (2007) and as commented in Section 2, IG methods have been applied to all sorts of scheduling problems since then. The main feature of the IG is its simplicity which is contrary to sophisticated algorithms that embed problem specific knowledge and that usually have many control parameters. In contrast, IG has very few parameters. Despite its simplicity, IG has shown state-of-the-art results under different flowshop variants and objectives.

An IG algorithm consists basically of a few steps. First, a starting solution is built, usually by means of a high performing constructive heuristic. Then the main loop is run until a termination criterion is reached. Inside this loop, two operators are iteratively applied. The first operator is a random destruction, where some elements of the solution are removed. The second operator is a greedy reconstruction method which reinserts the removed elements in order to form a new complete solution. The reconstruction also uses a high performing heuristic. After a new complete solution is obtained, an acceptance criterion is applied in order to decide if the new solution substitutes the incumbent. Optionally, a local search procedure can be applied, typically after the initial solution construction and before the acceptance criterion at each pass of the main loop. All these steps are explained in the following sections.

4.1. Initialization

By far, the NEH algorithm of Nawaz et al. (1983) is the heuristic of choice for the initialization of metaheuristics in the flowshop literature. The NEH is a greedy constructive heuristic. Jobs are initially sorted according to total processing times and then the two possible permutations containing the first two sorted jobs are calculated. The best among the two is kept for the third step. In the third step, the third sorted job is inserted in the first, second and third possible positions of the partial sequence. The job is finally placed in the position resulting in the best makespan value. The process continues with the fourth job and completes when all jobs have been inserted. Most state-of-the-art methods for the PFSP and many variants employ the NEH. Ruiz and Maroto (2005) demonstrated the NEH to be the best heuristic, better even than more modern heuristics. Some authors, like Dong et al. (2008) or Kalczynski and Kamburowski (2007) have shown some methods that improve on the NEH performance. However, the outperformance is relatively small as these methods focus on the ties that occur in the insertion steps of the NEH. Clearly better heuristics are presented in Rad et al. (2009) where the authors proposed five methods, referred to as FRB1-FRB5 and demonstrated a significant advantage over the NEH. This outperformance comes at an additional computational cost as the methods are based on reinsertions of already inserted jobs. The authors also demonstrated that initializing competitive metaheuristics with some of their proposed methods instead of with the NEH produced better end results. Following these results, we also employ an improved heuristic instead of NEH. More precisely, we present an improvement of the $FRB4_k$ method of Rad et al. (2009). $FRB4_k$ produces good results while at the same time the additional CPU time needed is small. The idea behind the $FRB4_k$ is simple: after a job has been inserted in position p of the sequence in a given step of the NEH, k jobs around position p are reinserted in all positions looking for a better fit. The higher the k, the more jobs are reinserted and therefore the better results but also at a cost of more CPU time as the computational complexity is $\mathcal{O}(kn^2m)$. Our proposed improvement over the FRB4$_k$ is based on the recent work of Pan and Wang (2012). In this paper, it was observed that the initial LPT ordering of the NEH is being broken during the insertions. The authors proposed a modification in which a partial LPT sequence of jobs is kept and the NEH process starts after a number of jobs λ have been assigned in the initial sequence. The side benefit of this modification is that less steps are needed in the main loop and the FRB4$_k$ gains speed. Furthermore, to speed up the process, we fix k at the lowest possible value of one. A pseudo-algorithm for this improved method, referred to as FRB4$_1^*$, is given in Figure 3.

```
procedure FRB4*_1(λ)
 Calculate P_j = Σ_{i=1}^{m} p_ij, ∀j ∈ N      % (LPT order)
 Sort jobs according to decreasing order of P_j obtaining β = {β_(1), ..., β_(n)}
 π := {β_(1), β_(2), ..., β_(λ−1)}   % Initial partial LPT sequence
 for l := λ to n do
  Take job β_(l) and test it in all positions of π
  Insert job β_(l) in position p of π resulting in the best C_max
  for m := max(1, p − 1) to min(l, p + 1) do
   Extract job π_(m) from position m of π and test it in all positions of π
   Insert job π_(m) at the position resulting in the best C_max
  endfor
 endfor
end
```

Figure 3: Improved constructive heuristic FRB4$_1^*$.

Note that the initial solution obtained after applying the FRB4$_1^*$ method is further improved with the local search algorithm detailed in Section 4.2. The proposed FRB4$_1^*$ method has a working parameter λ indicating when the NEH insertions start. This parameter will be calibrated in Section 4.4.

4.2. Local search

Similar to the NEH, which is an insertion constructive heuristic, most competitive local search methods for the PFSP and variants are based on the insertion neighborhood. Good results with the insertion neighborhood are obtained in many papers, most notably Ruiz and Stützle (2007), Framinan and Leisten (2008) and Vallada and Ruiz (2009), to cite just a few. In the insertion neighborhood, a job is extracted from its position and inserted in all other $n-1$ possible positions of the sequence (excluding the original one). If a better $C_{\max}$ value is found in a different position, the job is relocated and the process is repeated for another job. The process terminates when all jobs have been placed in all possible positions without improvements. Note that the accelerations given in Section 3.3 fit perfectly into this scheme, allowing us to reap the speed benefits. This local search was used for the IG by Ruiz and Stützle (2007), Ruiz and Stützle (2008) and Vallada and Ruiz (2009) for problems other than the no-idle flowshop and by Ruiz et al. (2009) for the no-idle version. In this local search, jobs to be inserted are selected randomly, without repetition, until local optimality is reached. However, quite recently, Pan et al. (2008a) and Pan and Ruiz (2012) have used a similar but better performing version, referred to as referenced local search (RLS). In this version, jobs are not extracted randomly but in the order given by a referenced permutation. Recently, Deng and Gu (2012) also applied RLS to the no-idle flowshop. Let $\pi^{\text{ref}} = \{\pi^{\text{ref}}_{(1)}, \pi^{\text{ref}}_{(2)}, \ldots, \pi^{\text{ref}}_{(n)}\}$ be the referenced sequence, which, in this paper, is the best found solution so far. The RLS is detailed in Figure 4. Both the regular local search of Ruiz and Stützle (2007) and the presented RLS will be tested in the proposed IG.

procedure RLS(π,π^{ref})
  $i := 1$; $counter := 0$
  repeat
    Locate and extract job $\pi^{\text{ref}}_{(i)}$ from π
    Take job $\pi^{\text{ref}}_{(i)}$ and test it in all positions of π
    $\pi^* :=$ Insert job $\pi^{\text{ref}}_{(i)}$ at the position resulting in the best $C_{\max}$
    if $C_{\max}(\pi^*) < C_{\max}(\pi)$ **then**
      $\pi := \pi^*$; $counter := 1$
    elseif
      $counter := counter + 1$
     endif
    $i := \text{mod}(i+1, n)$
  until $counter = n$
  return π
end

Figure 4: Referenced Local Search (RLS) in the insertion neighborhood.

4.3. Destruction, reconstruction and acceptance criterion

In the destruction phase of the Iterated Greedy, and according to Ruiz and Stützle (2007), d jobs are randomly extracted from the incumbent permutation π and inserted into a list of removed jobs π_R. Then, in the reconstruction phase, all jobs in π_R are reinserted, one by one, back into π using the NEH insertion procedure. This is referred to as the DC operator (Destruction-reConstruction). We propose a minor but, as we will see, important modification as regards the final performance of the proposed method. After reinserting one job, the jobs occupying the previous and posterior positions are also reinserted in all positions of π. This is, in essence, the application of the FRB4_1^* ideas presented previously. This improved DC operator is referred to as eDC. The local search operator is applied after the solution has been fully reconstructed.

Note that the choice of d in the destruction procedure is key. A small d value will result in difficulties for IG in escaping strong local optima whereas a large d value is no different from a randomized NEH procedure. Similar to what Ruiz and Stützle (2007) did, we will calibrate the d value using strong statistical techniques.

At each iteration, after the destruction, reconstruction and local search steps we have a new solution. It has to be decided if this solution replaces the current incumbent one. We adopt the same acceptance criterion as Ruiz and Stützle (2007) and Ruiz and Stützle (2008) which in turn is based on the constant temperature Simulated Annealing-like criterion of Osman and Potts (1989). Basically, a constant temperature is calculated as $Temp = T \cdot \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} p_{ij}}{n \cdot m \cdot 10}$, where T is another value to calibrate. Ruiz and Stützle (2007) demonstrated this value to be very robust and basically any other value except 0 is acceptable. The final proposed IG method, including some alternative operators, is given in Figure 5.

4.4. Calibration of the $FRB4_1^*$ heuristic and proposed IG

In this section we calibrate the λ parameter of the FRB4_1^* heuristic and also test the two local search schemes, construction and reconstruction operators of the IG method, along with the temperature T and number of jobs to destruct in the destruction phase (d). In order to calibrate these methods we need some test instances.

In this paper we propose a comprehensive benchmark. Since there is no known benchmark for the MNPFSP, we base our instances on those for

```
procedure IG(d, T)
 π :=FRB4*_1
 π :=LS(π) or RLS(π)     % Choice of local search
 π_b := π
 while (termination criterion not satisfied) do
  π' := π
  for i := 1 to d do        % Destruction phase
   π' := remove one job at random from π' and insert it in π'_R
  endfor
  for i := 1 to d do        % Reconstruction phase
   π' := Insert job π'_R(i) in position p resulting in the best C_max
   % Improved eDC operator
   π' := Reinsert jobs π'_(p±1) in positions resulting in the best C_max
  endfor
  π'' :=LS(π') or RLS(π')    % Choice of local search
  if C_max(π'') < C_max(π) then     % Acceptance Criterion
   π := π''
   if C_max(π) < C_max(π_b) then    % New best solution
    π_b := π
   endif
  elseif (random ≤ e^{-(C_max(π'')-C_max(π))/Temp}) then
   π := π''
  endif
 endwhile
end
```

Figure 5: Proposed Iterated Greedy (IG) method.

the no-idle PFSP of Ruiz et al. (2009). The basic benchmark contains 250 instances. All combinations of the following n and m values are used: $n = \{50, 100, 150, 200, 250, 300, 350, 400, 450, 500\}$ and $m = \{10, 20, 30, 40, 50\}$. For each one of the $10 \times 5 = 50$ combinations, five replicates are obtained which results in 250 instances. Furthermore, in order to test different mixed no-idle scenarios, we generate seven different groups as follows: Group 1: The first 50% of the machines have the no-idle constraint. The remaining 50% are regular idle machines. Group 2: The second 50% of the machines have the no-idle constraint. Group 3: The machines alternate, in order, between regular and no-idle constraints. Group 4: A random 25% of the machines are no-idle. Group 5: 50% random no-idle machines. Group 6: 75% random no-idle machines. Group 7: This group contains the 250 original no-idle instances of Ruiz et al. (2009), i.e., in this group all machines have the no-idle constraint. Since there are 250 instances in each group, the grand total of instances in the benchmark is 1,750. The processing times for all instances are generated

following a uniform distribution in the range $U[1, 99]$ as it is common in the scheduling literature. Note that this is a comprehensive benchmark that will allow us to use detailed results in the computational comparison.
Calibrating algorithms with the same instances that will later be used for computational results and comparisons constitutes poor practice. If an algorithm is calibrated on the same instances that will be later tested we risk having biased or over fitted results. In order to remedy this problem we also generate a calibration benchmark of 100 random instances. To generate each instance, a random n, m and group are selected and the instance is generated. All instances, both the test and the calibration benchmarks are available for download at `http://soa.iti.es`.

A first quick experiment was carried out to calibrate the λ parameter of the FRB4$_1^*$. We use the 100 calibration instances and test λ from 0 to 100%. This percentage relates to the number of jobs n so a $\lambda = 50\%$ means that 50% of the initial sequence is maintained as LPT before starting the NEH insertion procedure. We use a step equal to 5% which means that we test 21 different values for λ. We solve the 100 calibration instances with these 21 versions of the FRB4$_1^*$. The response variable to measure is the Relative Percentage Deviation from the best known solution denoted as $RPD = \frac{Some_{sol} - Best_{sol}}{Best_{sol}} \cdot 100$. $Some_{sol}$ is the solution obtained by one of the versions on a given instance and $Best_{sol}$ is the lowest makespan known for that instance. All best known solutions for the test instances are also available at `http://soa.iti.es`.
All tests are carried out in a cluster with 30 blades, each one containing two Intel XEON E5420 processors with a core clock of 2.5 GHz. and 4 cores each (8 in total per blade) and 16 GBytes of RAM memory (480 GBytes in total). To analyze the results we carry out a full factorial design of experiments with one factor (λ) at 21 levels on 100 instances which gives 2,100 treatments. The results of the experiment are analyzed by means of the Analysis of Variance (ANOVA) technique. ANOVA is a parametric statistical technique and three main hypotheses must be met. In order of importance these are the independence of the residuals, homoscesdasticity of the different levels and variants of the factors studied (homogeneity of variance) and normality of the residuals. No significant deviations were found in the fulfillment of the hypotheses. The detailed results of this short initial experiment are omitted due to space constraints but suffice to say that the statistically best result is obtained when $\lambda = 50\% n$.

A much larger Design of Experiments (see Montgomery (2012), among many others) is carried out to calibrate the proposed IG. We test the following factors: 1) type of destruction-reconstruction operator, tested at two variants: regular DC and improved eDC. 2) type of local search, tested at two variants: regular LS and referenced local search RLS. 3) Destruction size d tested at six levels: 8-13. 4) T tested at five levels: 0.4-0.8. Apart from these controlled factors, each IG configuration is run five different times on each instance (we call this the replicate witness factor which should not be statistically significant). Note that the IG needs a termination criterion, which we set at a given elapsed CPU time equal to $t = 5nm$ milliseconds. Setting the CPU time depending on the instance size (number of jobs n and number of machines m) is good practice in order to better observe the effect of the factors. With a fixed CPU time, smaller instances end up with large CPU times and become "easy" whereas large instances might not have enough CPU time and might be wrongly portrayed as "hard". To sum up, we have a multi-factor full factorial experimental design with $100 \cdot 5 \cdot 2 \cdot 2 \cdot 6 \cdot 5 = 60,000$ treatments. With such a large and powerful experiment, we will be able to fully calibrate the proposed IG with a high degree of accuracy. The same computer is used for the experiments and the RPD response variable is analyzed in a multi-factor ANOVA. We do not show here the ANOVA table with interactions of second order due to space limitations. Instead, we reproduce the means plots with confidence intervals of the most important and statistically significant factors. The most significant factors are the type of local search and d, followed by the type of destruction-reconstruction factors. The means plots of these factors, together with 95% Tukey's Honest Significant Difference (HSD) confidence intervals are given in Figure 6. Recall that overlapping confidence intervals means that the observed difference in the response variable (RPD) of the two overlapped means is statistically insignificant.

As can be seen, the improved eDC destruction-reconstruction operator is statistically better than the Ruiz and Stützle (2007) regular operator. The same can be said about the referenced local search RLS. While in Figure 6 it might seem that the differences are small, combined, the usage of eDC in conjunction with RLS results in significant improvements over the regular LS and DC operators. As regards d, the differences are small for central values and we settle for $d = 10$. Finally, the factor T is not statistically significant, which coincides with the results of Ruiz and Stützle (2007). We select the central value of $T = 0.6$. Detailed ANOVA tables and all results of the experiments are available upon request from the authors.

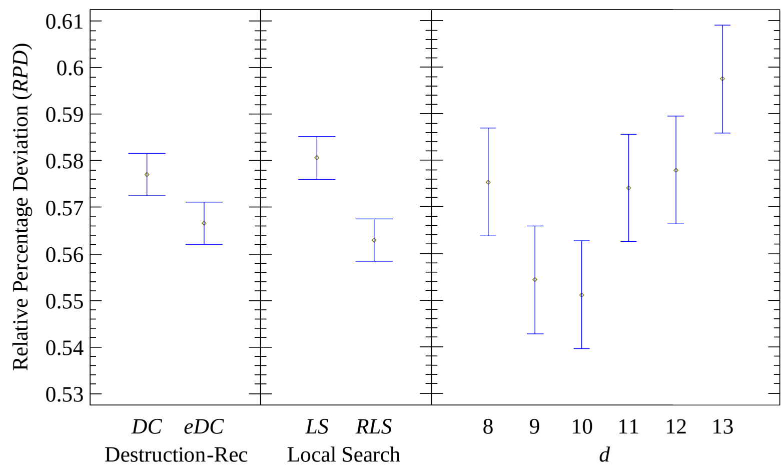

Figure 6: Means plot for the type of destruction-reconstruction operator, type of local search and d factors for the IG ANOVA calibration experiment. All means have Tukey's Honest Significant Difference (HSD) 95% confidence intervals.

5. Computational comparisons and statistical analysis

After calibrating the proposed IG method we compare it with the state-of-the-art algorithms from the literature. Since there are no MNPFSP methods proposed so far, we take the best competing algorithms from the no-idle PFSP literature. We will use the 1,750 test instances detailed in Section 4.4 for the computational comparisons. Note that the 7th group in those instances are no-idle problems so the proposed IG will be tested against existing methods on no-idle instances as well. The following methods have been fully reimplemented: 1) The hybrid Genetic Algorithm of Ruiz et al. (2006) (hGA). 2) The hybrid discrete PSO of Pan and Wang (2008a) (hDPSO). 3) The hybrid discrete differential evolution algorithm of Pan and Wang (2008b) (hDDE$_P$). 4,5) The IG method of Ruiz et al. (2009) tested with $d = 4$ and $d = 8$ (IG$_{R_4}$ and IG$_{R_8}$, respectively). 6) The hybrid discrete differential evolution algorithm of Deng and Gu (2012) (hDDE$_D$). 7) The recent variable IG hybridized with differential evolution of Fatih Tasgetiren et al. (2013a) (IG$_T$) and finally the eighth method is the IG algorithm proposed in this paper (IG). Note that all methods have been reimplemented and use the proposed accelerations of the insertion neighborhood. Makespan calculation functions are also shared. All methods have been coded in Visual C++ 6.0 and have been run on the same computers. Therefore, the results are fully and completely comparable.
All algorithms have a natural stopping criterion which we set at a predefined

elapsed CPU time following the expression $t = n \times (m/2) \times \rho$ milliseconds where ρ has been tested at values 10, 20, 30, 60, 90. Our objective is to analyze the performance of all the methods from short to very long CPU times. Note that for $\rho = 90$ the largest instances of 500 jobs and 50 machines are run for almost 19 minutes. Given the 8 algorithms tested, 1,750 instances, 5 different stopping times and 5 replicates we have a total of $1,750 \times 5 \times 5 \times 8 = 350,000$ results. This is an extremely rich dataset which will allow us to draw strong conclusions. Note that the total CPU time needed for all experiments was 1.92 years (the real time was much shorter as all tests were divided among the 30 blade clusters). The average relative percentage deviation, grouped only by instance group (250 instances $\times$ 5 replicates $\times$ 5 different stopping times = $6,250$ values averaged at each cell) are given in Table 1.

Instance group	hDDE$_D$	hDDE$_P$	hDPSO	hGA	IG	IG$_{R_4}$	IG$_{R_8}$	IG$_T$
1	0.42	0.41	0.42	0.65	**0.33**	0.61	0.42	0.95
2	0.42	0.41	0.44	0.66	**0.35**	0.63	0.42	0.96
3	0.42	0.42	0.43	0.66	**0.31**	0.65	0.43	0.95
4	0.47	0.45	0.47	0.74	**0.37**	0.64	0.46	1.14
5	0.44	0.42	0.45	0.68	**0.31**	0.66	0.44	0.98
6	0.42	0.40	0.41	0.62	**0.26**	0.62	0.40	0.81
7	0.39	0.37	0.40	0.56	**0.23**	0.61	0.37	0.71
Average	0.42	0.41	0.43	0.65	**0.31**	0.63	0.42	0.93

Table 1: Average Relative Percentage Deviation for all the 8 tested algorithms and the 1,750 test instances. Results grouped by type of instance.

As can be seen, the proposed IG produces the best results in all instance groups. While for groups 1-6 this is somewhat expected, as these are the mixed no-idle groups and the other methods were not designed for this setting, the differences are also large for group 7, which is the full no-idle case. For group 7, the Average *RPD* of the proposed IG is 0.23 whereas the second best method is hDDE$_P$ (tied with IG$_{R_8}$), which have an Average *RPD* of 0.37%. This means that the IG produces solutions that are, on average, almost 61% better for the no-idle flowshop. Clearly, IG presents itself as the new state-of-the-art for the no-idle flowshop problem. On average, the best algorithm is the IG with an overall *RPD* for the 1,750 instances of 0.31%. The second best overall method is hDDE$_P$ with an Average *RPD* of 0.41%, again, with a large outperformance of more than 33%. It is also of interest to examine the results of Table 1 but broken down according to the allowed CPU time ρ. This is given in Table 2.

ρ	hDDE$_D$	hDDE$_P$	hDPSO	hGA	IG	IG$_{R_4}$	IG$_{R_8}$	IG$_T$
10	0.47	0.49	0.49	0.72	**0.36**	0.73	0.48	1.06
20	0.44	0.44	0.45	0.67	**0.32**	0.67	0.44	1.01
30	0.42	0.41	0.43	0.64	**0.31**	0.63	0.42	0.97
60	0.40	0.37	0.40	0.62	**0.28**	0.58	0.39	0.85
90	0.39	0.35	0.39	0.61	**0.27**	0.55	0.37	0.76
Average	0.42	0.41	0.43	0.65	**0.31**	0.63	0.42	0.93

Table 2: Average Relative Percentage Deviation for all the 8 tested algorithms and the 1,750 test instances. Results grouped by allowed CPU time ρ.

Once again, the superiority of the proposed IG method is clear. While we were expecting that for larger values of ρ the differences between methods would diminish, we have found this not to be the case. The IG method has a lead of more than 30% in Average *RPD* regardless of ρ value.

While the differences between IG and competing methods depicted in Tables 1 and 2 are quite large, it is still mandatory to run some statistical tests on the results in order to ascertain if the observed differences in the Average *RPD* values are indeed statistically significant. We have conducted a multi-factor ANOVA where n, m, instance group, ρ, replica (witness factor) and algorithm are all controlled factors. Single factor effects as well as two way interactions are studied. As expected with such a large dataset, most factors are statistically significant (after all, with an infinite sample size, all differences in the means, even if they tend to zero, are statistically significant). We are most interested in the interaction between the algorithm and ρ, shown in Figure 7.

As can be seen, there are four groups of algorithms with no statistically significant differences in the Average *RPD* within each group. The first group is composed of algorithm IG$_T$, which, despite being a very recent proposal for the no-idle flowshop, it no better than the rest. However, and as we can see, it is the algorithm that benefits most from the added CPU time. The second group is made up of hGA and IG$_{R_4}$. These results are expected since, and according to the results of Ruiz and Stützle (2007), the basic IG performs very similar to that of the GA of Ruiz et al. (2006). A tight third group is formed by IG$_{R_8}$, hDDE$_D$, hDDE$_P$ and hDPSO. With the exception of IG$_{R_8}$, which was not tested with $d = 8$ by the original authors (Ruiz et al., 2009),

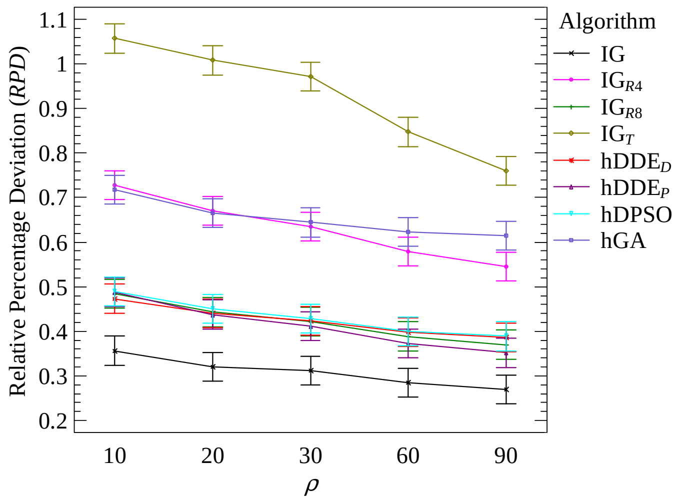

Figure 7: Means plot for the interaction between the algorithm and elapsed CPU time stopping criterion (ρ). All means have Tukey's Honest Significant Difference (HSD) 95% confidence intervals.

the other three algorithms are very similar and therefore it is expected that their performance is comparable with one another. The last group is formed of the proposed method IG. We can see that for all CPU times (values of ρ) its Tukey's Honest Significant Difference (HSD) 95% confidence intervals do not overlap with the intervals of any of the other methods. This means that for all tested values of ρ, the proposed IG is statistically better than all other methods and by a significant margin. A full table with the breakdown of n and m, as well as all results, best detailed solutions, excel files and statistical tests are available upon request from the authors.

6. Conclusions and future research

This paper proposes for the first time a generalization of both the regular permutation flowshop and no-idle permutation flowshop scheduling problem resulting from the consideration of both regular as well as no-idle machines in the shop. The result is referred to as the mixed no-idle problem or MNPFSP. It has many practical applications in the ceramic tile industry, the production of ceramic frits, the steelmaking industry and the manufacturing of integrated circuits among many others.

We have reviewed the existing literature and have proved the novelty of the MNPFSP setting, for which we have presented a mixed linear integer programming model. We have shown how to calculate the makespan value and have demonstrated that it is far from trivial. The insertion neighborhood is frequently employed by heuristics and metaheuristics in the flowshop literature and we have also presented in this paper a method for calculating all insertions of a job in a sequence in $\mathcal{O}(nm)$ steps, reducing the computational complexity and allowing for fast methods. We have presented an improved Iterated Greedy (IG) method that builds on the successful algorithms of Ruiz and Stützle (2007). We have extended the method with a more comprehensive initialization, an improved destruction-reconstruction operator and referenced local search. After careful calibration, we have tested our proposed IG against 7 other state-of-the-art methods mainly proposed for the no-idle flowshop. In a comprehensive benchmark of 1,750 instances and after an accumulated CPU time of almost two years we have demonstrated that the proposed IG is not only statistically better than all other methods in the mixed no-idle settings but also in the full no-idle environment and by a wide and significant margin. The outcome of the experimentation is also interesting since the proposed IG is much simpler than the competing hybrid discrete differential evolution and hybrid discrete particle swarm optimization methods. Our experiments include 350,000 different results which, along with the powerful statistical analyses allow us to conclude that the proposed IG is the new state-of-the-art both for the no-idle flowshop as well as for the new mixed no-idle flowshop. Future research will include the consideration of other optimization objectives and sequence dependent setup times, possibly for the regular idling machines as these configurations are common within industry. Hybrid no-idle or hybrid mixed no-idle flowshops pose another interesting avenue for future research.

Acknowledgements

Quan-Ke Pan is partially supported by the National Science Foundation of China (No. 61174187), Basic Scientific Research Foundation of Northeast University (No. N110208001), and by the Starting Foundation of Northeast University (No. 29321006). Rubén Ruiz is partially supported by the Spanish Ministry of Economy and Competitiveness, under the project "RESULT - Realistic Extended Scheduling Using Light Techniques" with reference DPI2012-36243-C02-01 co-financed by the European Union and FEDER funds and by the Universitat Politècnica de València, for the project MRPIV with reference PAID/2012/202.

References

Adiri, I. and Pohoryles, D. (1982). Flowshop no-idle or no-wait scheduling to minimize the sum of completion times. *Naval Research Logistics*, 29(3):495–504.

Baker, K. R. (1974). *Introduction to Sequencing and Scheduling*. John Wiley & Sons, New York.

Baptiste, P. and Hguny, L. K. (1997). A branch and bound algorithm for the $F/no-idle/C_{max}$. In *Proceedings of the International Conference on Industrial Engineering and Production Management, IEPM'97*, volume 1, pages 429–438, Lyon, France.

Baraz, D. and Mosheiov, G. (2008). A note on a greedy heuristic for flowshop makespan minimization with no machine idle-time. *European Journal of Operational Research*, 184(2):810–813.

Deng, G. and Gu, X. (2012). A hybrid discrete differential evolution algorithm for the no-idle permutation flow shop scheduling problem with makespan criterion. *Computers & Operations Research*, 39(9):2152–2160.

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

Fatih Tasgetiren, M., Pan, Q.-K., Suganthan, P. N., and Buyukdagli, O. (2013a). A variable iterated greedy algorithm with differential evolution for the no-idle permutation flowshop scheduling problem. *Computers & Operations Research*, 40(7):1729–1743.

Fatih Tasgetiren, M., Pan, Q.-K., Suganthan, P. N., and Oner, A. (2013b). A discrete artificial bee colony algorithm for the no-idle permutation flowshop scheduling problem with the total tardiness criterion. *Applied Mathematical Modelling*, 37(10-11):6758–6779.

Framinan, J. M., Gupta, J. N. D., 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(1):1243–1255.

Framinan, J. M. 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.

Garey, M. R., Johnson, D. S., and Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2):117–129.

Goncharov, Y. and Sevastyanov, S. (2009). The flow shop problem with no-idle constraints: A review and approximation. *European Journal of Operational Research*, 196(2):450–456.

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

Gupta, J. N. D. and Stafford, Jr, E. F. (2006). Flowshop scheduling research after five decades. *European Journal of Operational Research*, 169(3):699–711.

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

Johnson, S. M. (1954). Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1(1):61–68.

Kalczynski, P. and Kamburowski, J. (2007). On the neh heuristic for minimizing the makespan in permutation flow shops. *OMEGA, the International Journal of Management Science*, 35(1):53–60.

Kalczynski, P. J. and Kamburowski, J. (2005). A heuristic for minimizing the makespan in no-idle permutation flow shops. *Computers & Industrial Engineering*, 49(1):146–154.

Kamburowski, J. (2004). More on three-machine no-idle flow shops. *Computers & Industrial Engineering*, 46(3):461–466.

Minella, G., Ruiz, R., and Ciavotta, M. (2008). A review and evaluation of multi-objective algorithms for the flowshop scheduling problem. *INFORMS Journal on Computing*, 20(3):451–471.

Minella, G., Ruiz, R., and Ciavotta, M. (2011). Restarted iterated pareto greedy algorithm for multi-objective flowshop scheduling problems. *Computers & Operations Research*, 38(11):1521–1533.

Montgomery, D. C. (2012). *Design and Analysis of Experiments*. Wiley, eight edition.

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

Narain, L. and Bagga, P. C. (2003). Minimizing total elapsed time subject to zero total idle time of machines in $n \times 3$ flowshop problem. *Indian Journal of Pure & Applied Mathematics*, 34(2):219–228.

Narain, L. and Bagga, P. C. (2005a). Flowshop/no-idle scheduling to minimise the mean flowtime. *Anziam Journal*, 47:265–275.

Narain, L. and Bagga, P. C. (2005b). Flowshop/no-idle scheduling to minimize total elapsed time. *Journal of Global Optimization*, 33(3):349–367.

Nawaz, M., Enscore, Jr, E. 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.

Nowicki, E. and Smutnicki, C. (1996). A fast tabu search algorithm for the permutation flow-shop problem. *European Journal of Operational Research*, 91(1):160–175.

Osman, I. and Potts, C. (1989). Simulated annealing for permutation flow-shop scheduling. *OMEGA, The International Journal of Management Science*, 17(6):551–557.

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

Pan, Q.-K. and Ruiz, R. (2012). An estimation of distribution algorithm for lot-streaming flow shop problems with setup times. *OMEGA, the International Journal of Management Science*, 40(2):166–180.

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. and Wang, L. (2008a). No-idle permutation flow shop scheduling based on a hybrid discrete particle swarm optimization algorithm. *International Journal of Advanced Manufacturing Technology*, 39(7-8):796–807.

Pan, Q.-K. and Wang, L. (2008b). A novel differential evolution algorithm for no-idle permutation flow-shop scheduling problems. *European Journal of Industrial Engineering*, 2(3):279–297.

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.

Pan, Q.-K., Wang, L., and Zhao, B. H. (2008b). An improved iterated greedy algorithm for the no-wait flow shop scheduling problem with makespan criterion. *International Journal of Advanced Manufacturing Technology*, 38(7-8):778–786.

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.

Ribas, I., Companys, R., and Tort-Martorell, X. (2010). Comparing three-step heuristics for the permutation flow shop problem. *Computers & Operations Research*, 37(12):2062–2070.

Ribas, I., Companys, R., and Tort-Martorell, X. (2011). An iterated greedy algorithm for the flowshop scheduling problem with blocking. *OMEGA, The International Journal of Management Science*, 39(3):293–301.

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., Maroto, C., and Alcaraz, J. (2006). Two new robust genetic algorithms for the flowshop scheduling problem. *OMEGA, the International Journal of Management Science*, 34:461–476.

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.

Ruiz, R. and Stützle, T. (2008). An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. *European Journal of Operational Research*, 187(3):1143–1159.

Ruiz, R., Vallada, E., and Fernández-Martínez, C. (2009). Scheduling in flowshops with no-idle machines. In Chakraborty, U., editor, *Computational Intelligence in Flow Shop and Job Shop Scheduling*, chapter 2, pages 21–51. Springer, New York.

Saadani, N. E. H., Guinet, A., and Moalla, M. (2001). A travelling salesman approach to solve the $F/no-idle/C_{max}$ problem. In *Proceedings of the International Conference on Industrial Engineering and Production Management, IEPM'01*, volume 2, pages 880–888, Quebec, Canada.

Saadani, N. E. H., Guinet, A., and Moalla, M. (2003). Three stage no-idle flow-shops. *Computers & Industrial Engineering*, 44(3):425–434.

Saadani, N. E. H., Guinet, A., and Moalla, M. (2005). A travelling salesman approach to solve the $F/no-idle/C_{max}$ problem. *European Journal of Operational Research*, 161(1):11–20.

Salveson, M. E. (1952). On a quantitative method in production planning and scheduling. *Econometrica*, 20(4):554–590.

Taillard, E. (1990). Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research*, 47(1):67–74.

Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285.

Vachajitpan, P. (1982). Job sequencing with continuous machine operation. *Computers & Industrial Engineering*, 6(3):255–259.

Vallada, E. and Ruiz, R. (2009). Cooperative metaheuristics for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 193(2):365–376.

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.

Čepek, O., Okada, M., and Vlach, M. (2000). Note: On the two-machine no-idle flowshop problem. *Naval Research Logistics*, 47(4):353–358.

Woollam, C. R. (1986). Flowshop with no idle machine time allowed. *Computers & Industrial Engineering*, 10(1):69–76.

Ying, K. C. (2008). Solving non-permutation flowshop scheduling problems by an effective iterated greedy heuristic. *International Journal of Advanced Manufacturing Technology*, 38(3-4):348–354.