Computers & Operations Research 37 (2010) 754–768

Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/cor

The distributed permutation flowshop scheduling problem

B. Naderi[a,*], Rubén Ruiz[b]

[a]Department of Industrial Engineering, Amirkabir University of Technology, 424 Hafez Avenue, Tehran, Iran
[b]Grupo de Sistemas de Optimización Aplicada, Instituto Tecnológico de Informática (ITI), Ciudad Politécnica de la Innovación, Edificio 8G, Acceso B, Universidad Politécnica de Valencia, Camino de Vera s/n, 46022 Valencia, Spain

ARTICLE INFO

Available online 3 July 2009

Keywords:
Distributed scheduling
Permutation flowshop
Mixed integer linear programming
Variable neighborhood descent

ABSTRACT

This paper studies a new generalization of the regular permutation flowshop scheduling problem (PFSP) referred to as the distributed permutation flowshop scheduling problem or DPFSP. Under this generalization, we assume that there are a total of F identical factories or shops, each one with m machines disposed in series. A set of n available jobs have to be distributed among the F factories and then a processing sequence has to be derived for the jobs assigned to each factory. The optimization criterion is the minimization of the maximum completion time or makespan among the factories. This production setting is necessary in today's decentralized and globalized economy where several production centers might be available for a firm. We characterize the DPFSP and propose six different alternative mixed integer linear programming (MILP) models that are carefully and statistically analyzed for performance. We also propose two simple factory assignment rules together with 14 heuristics based on dispatching rules, effective constructive heuristics and variable neighborhood descent methods. A comprehensive computational and statistical analysis is conducted in order to analyze the performance of the proposed methods.

1. Introduction

The flowshop scheduling has been a very active and prolific research area since the seminal paper of Johnson [1]. In the flowshop problem (FSP) a set N of n unrelated jobs are to be processed on a set M of m machines. These machines are disposed in series and each job has to visit all of them in the same order. This order can be assumed, without loss of generality, to be $1,\dots,m$. Therefore, each job $j, j \in N$ is composed of m tasks. The processing times of the jobs at the machines are known in advance, non-negative and deterministic. They are denoted by $P_{j,i}$, $j \in N$, $i \in M$. The FSP entails a number of assumptions [2]:

- All jobs are independent and available for processing at time 0.
- Machines are continuously available (no breakdowns).
- Each machine can only process one job at a time.
- Each job can be processed only on one machine at a time.
- Once the processing of a given job has started on a given machine, it cannot be interrupted and processing continues until completion (no preemption).
- Setup times are sequence independent and are included in the processing times, or ignored.
- Infinite in-process storage is allowed.

The objective is to obtain a production sequence of the n jobs on the m machines so that a given criterion is optimized. A production sequence is normally defined for each machine. Since there are $n!$ possible job permutations per machine, the total number of possible sequences is $(n!)^m$. However, a common simplification in the literature is to assume that the same job permutation is maintained throughout all machines. This effectively prohibits job passing between machines and reduces the solution space to $n!$. With this simplification, the problem is referred to as the permutation flowshop scheduling problem or PFSP in short.

Most scheduling criteria use the completion times of the jobs at the machines, which are denoted as $C_{j,i}$, $j \in N$, $i \in M$. Once a production sequence or permutation of jobs has been determined (be this permutation π, and be the job occupying position $j, \pi_{(j)}$), the completion times are easily calculated in $O(nm)$ steps with the following recursive expression:

$$C_{\pi_{(j)},i} = \max\{C_{\pi_{(j)},i-1}, C_{\pi_{(j-1)},i}\} + P_{\pi_{(j)},i}$$

where $C_{\pi_{(j)},0} = 0$ and $C_{\pi_{(0)},i} = 0$, $\forall i \in M$, $\forall j \in N$. One of the most common optimization criteria is the minimization of the maximum

* Corresponding author.
E-mail address: bahman.naderi@aut.ac.ir (B. Naderi).

completion time or makespan, i.e., $C_{\max} = C_{\pi_{(n)},m}$. Under this optimization objective, the PFSP is denoted as $F/prmu/C_{\max}$ following the well known three field notation of Graham et al. [3]. This problem is known to be NP-Complete in the strong sense when $m \geq 3$ according to Garey et al. [4]. The literature of the PFSP is extensive, recent reviews are those of Framinan et al. [5], Hejazi and Saghafian [6], Ruiz and Maroto [7] and Gupta and Stafford [8].

In the aforementioned reviews, and as the next section will also point out, there is a common and universal assumption in the PFSP literature: there is only one production center, factory or shop. This means that all jobs are assumed to be processed in the same factory. Nowadays, single factory firms are less and less common, with multi-plant companies and supply chains taking a more important role in practice [9]. In the literature we find several studies where distributed manufacturing enables companies to achieve higher product quality, lower production costs and lower management risks (see [10,11] for some examples). However, such studies are more on the economic part and distributed finite capacity scheduling is seldom tackled. In comparison with traditional single-factory production scheduling problems, scheduling in distributed systems is more complicated. In single-factory problems, the problem is to schedule the jobs on a set of machines, while in the distributed problem an important additional decision arises: the allocation of the jobs to suitable factories. Therefore, two decisions have to be taken; job assignment to factories and job scheduling at each factory. Obviously, both problems are intimately related and cannot be solved sequentially if high performance is desired.

As a result, in this paper we study the distributed generalization of the PFSP, i.e., the distributed permutation flowshop scheduling problem or DPFSP. More formally, the DPFSP can be defined as follows: a set N of n jobs have to be processed on a set G of F factories, each one of them contains the same set M of m machines, just like in the regular PFSP. All factories are capable of processing all jobs. Once a job j, $j \in N$ is assigned to a factory f, $f \in G$ it cannot be transferred to another factory as it must be completed at the assigned plant. The processing time of job j on machine i at factory f is denoted as $P_{j,i,f}$. We are going to assume that processing times do not change from factory to factory and therefore, $P_{j,i,1} = P_{j,i,2} = \cdots = P_{j,i,f} = P_{j,i}$, $\forall i \in M$. We study the minimization of the maximum makespan among all factories and denote the DPFSP under this criterion as $DF/prmu/C_{\max}$.

The number of possible solutions for the DPFSP is a tricky number to calculate. Let us denote by f_h the number of jobs assigned to factory h, $h = \{1,2,\ldots,F\}$. The n jobs have to be partitioned into the F factories. For the moment, let us assume that this partition is given as $f_1, f_2, \ldots, f_F$. For the first factory, we have n available jobs and we have to assign f_1 so we have $\binom{n}{f_1}$ possible combinations. For each combination, we have $f_1!$ possible sequences. Therefore, for F factories, and given the partition of jobs $f_1, f_2, \ldots, f_F$, the total number of solutions is:

$$\binom{n}{f_1} f_1! \times \binom{n-f_2}{f_2} f_2! \times \cdots \times \binom{n-\sum_{h=1}^{F-1} f_h}{f_F} f_F!$$

This expression, with proper algebra, ends up being $n!$. However, this is just the number of solutions for a given known single partition of n jobs into F factories. In order to know the total number of solutions in the DFPSP, we need to know the number of ways of partitioning n jobs into F factories. One possibility is the Stirling number of the second kind or

$$(n,F) = \frac{1}{F!} \sum_{i=0}^{F-1} (-1)^i \binom{F}{i} (F-i)^n$$

However, for this number, two sets with identical number of jobs assigned are no different and in the DPFSP, each factory is a unique entity that needs to be considered separately. Therefore, we have to calculate the number of possible ways of partitioning n jobs into F distinct factories. Obviously, the sum of all the jobs assigned to all factories must be equal to n, i.e.:

$$f_1 + f_2 + \cdots + f_F = n, h = \{1,2,\ldots,F\}, 0 \leq f_h \leq n$$

The number of possible solutions for the previous equation is equal to the number possible partitions of the n identical jobs into F distinct factories. This gives a total number of partitions equal to $\binom{n+F-1}{F-1}$. With this result, we have to just multiply each partition by $n!$ to get the final number of solutions for the DPFSP with n jobs and F factories:

$$f_1(n,F) = \binom{n+F-1}{F-1} n!$$

To have an idea, we have that for $n = 5$ and $F = 3$, the total number of solutions is $\binom{7}{2} 5! = 21 \cdot 120 = 2520$ or 21 times more than the 120 possible solutions of the regular PFSP when $n = 5$. For $n = 10$ and $F = 4$, the total number of solutions in the DFPSP climbs up to 1,037,836,800, or 286 times more solutions than the regular 10! of the PFSP. $f_1(n,F)$ is exponential on both n and F. For example, f_1 (20,4) is almost 431 billion. Clearly, at least from the number of solutions, the DPFSP is more complex than the PFSP.

In the previous calculations, we are counting all solutions where one or more factories might be empty (i.e., $f_1(n,F)$ includes those cases in which at least one or more $f_h = 0$). As a matter of fact, in the DPFSP, nothing precludes us from leaving a factory without jobs assigned. However, and as regards the $C_{\max}$ optimization criterion, it makes no sense to leave any factory empty. In general, if $n \leq F$, we just need to assign each job to one factory in order to obtain the optimal solution. Actually there is nothing (no sequence) to optimize. As a result, in the DPFSP we have $n > F$ and $1 \leq f_h \leq n-F+1$, $h = \{1,2,\ldots,F\}$. The following theorem shows that it makes no sense to leave factories empty:

Theorem. *In at least one optimal solution of the DPFSP with makespan criterion, each factory has at least one job (if $n > F$).*

Proof. Consider an optimal solution s with

$$C_{\max} = \max\{C_1, C_2, \ldots, C_x, \ldots, C_y, \ldots, C_F\}$$

where C_h denotes the makespan of factory h. Suppose that $C_{\max} = C_y$ and $C_x = 0$ (i.e., factory x is empty). Therefore, since $n > F$, at least one factory has two or more jobs. From this point we have two cases:

Case 1: Suppose that factory y has one single job. Suppose also that another factory h, $h \neq x,y$ has more than one job (there should be at least one factory with more than one job). Clearly, the $C_{\max}$ does not increase if we assign one job from factory h to factory x.

Case 2: Suppose that factory y is the one with more than one job, we have that $C_y = C'_y + \delta_j$ where δ_j denotes the effect of job j on C_y. Therefore, $C'_y \leq C_y$ and $\delta_j \leq C_y$. If job j is assigned to factory x, we have that $C'_x \leq C_y$. Hence, after the change we have:

$$C'_{\max} = \max\{C_1, C_2, \ldots, C'_y, \ldots, C'_x, \ldots, C_F\}$$

$C'_{\max} \leq C_{\max}$ since $C'_y \leq C_y$ and $C'_x \leq C_y$ $\square$

Considering the above theorem, we can further reduce the search space from the previous results as follows:

$$f_1 + f_2 + \cdots + f_F = n \xrightarrow{f_h \geq 1,\ \ f'_h = f_h - 1} f'_1 + f'_2 + \cdots f'_F = n - F \quad f'_h \in N, \{0\}$$

Therefore, the total number of solutions becomes:

$$f_2(n,F) = \binom{n-F+F-1}{F-1} n! = \binom{n-1}{F-1} n!$$

This new result reduces the number of solutions tremendously, if we compare the ratio between $f_2(n,F)$ and $f_1(n,F)$ we have:

$$\frac{f_2(n,F)}{f_1(n,F)} = \frac{\binom{n-1}{F-1}}{\binom{n+F-1}{F-1}}$$

For example, in the case of $n=10$ and $F=4$, we reduce the number of solutions from approximately 1378 million to 305 million.

We have shown that the number of solutions of the DPFSP is significantly greater than those of the regular PFSP. In any case, since the DPFSP reduces to the regular PFSP if $F=1$, and this latter problem is NP-Complete, we can easily conclude that the DFPSP is also an NP-Complete problem (if $n>F$).

As we will show, this paper represents the first attempt at solving the DPFSP. Our first contribution is to develop and test a battery of six different alternative Mixed Integer Linear Programming (MILP) models. This allows for a precise characterization of the DPFSP. We carefully analyze the performance of the different MILP models as well as the influence of the number of factories, jobs, machines and some solver options over the results. Apart from the MILP modeling, we present two simple, yet very effective job factory assignment rules. With these, we propose several adaptations of existing PFSP dispatching rules and constructive heuristics to the DPFSP. Lastly, we also present two simple iterative methods based on Variable Neighborhood Descent (VND) from Hansen and Mladenovic [12]. In order to do so, we iteratively explore two neighborhoods, one that searches for better job sequences at each individual factory and a second that searches for better job assignments among factories.

The rest of the paper is organized as follows: Section 2 provides a short literature review of the limited existing literature on distributed scheduling. Section 3 deals with the proposed MILP models. In Section 4 we present the job factory assignment rules and the adaptations of dispatching rules and constructive heuristics. Section 5 introduces the proposed variable neighborhood descent algorithms. Section 6 gives a complete computational evaluation of the MILP models and algorithms by means of statistical tools. Finally, Section 7 concludes the paper and points several future research directions.

2. Literature review

As mentioned before, plenty of research has been carried out over the regular PFSP after the pioneering paper of Johnson [1]. The famous “Johnson's rule” is a fast $O(n \log n)$ method for obtaining the optimal solution for the $F2/prmu/C_{\max}$ (two machines) and for some special cases with three machines. Later, Palmer [13] presented a heuristic to solve the more general m-machine PFSP. This heuristic assigns an index to every job and then produces a sequence after sorting the jobs according to the calculated index. Campbell et al. [14] develop another heuristic which is basically an extension of Johnson's algorithm to the m machine case. The heuristic constructs $m-1$ schedules by grouping the m original machines into two virtual machines and solving the resulting two machine problems by repeatedly using Johnson's rule. The list of heuristics is almost endless. Ruiz and Maroto [7] provided a comprehensive evaluation of heuristic methods and concluded that the famous NEH heuristic of Nawaz et al. [15] provides the best performance for the $F/prmu/C_{\max}$ problem. As a matter of fact, the NEH is actively studied today as the recent papers of Framinan et al. [16], Dong et al. [17], Rad et al. [18] or Kalczynski and Kamburowski [19] show.

Regarding metaheuristics, there is also a vast literature of different proposals for the PFSP under different criteria. Noteworthy Tabu search methods are those of Nowicki and Smutnicki [20] or Grabowski and Wodecki [21]. Simulated annealing algorithms are proposed by Osman and Potts [22], while genetic algorithms are presented by Reeves [23] and in Ruiz et al. [24]. Other metaheuristics like Ant Colony Optimization, Scatter Search, Discrete Differential Evolution, Particle Swarm Optimization or Iterated Greedy are presented by Rajendran and Ziegler [25], Nowicki and Smutnicki [20], Onwubolu and Davendra [26], Tasgetiren et al. [27] and Ruiz and Stützle [28], respectively. Recent and high performing approaches include parallel computing methodologies, like the one presented in Vallada and Ruiz [29].

Apart from makespan minimization, the PFSP has been studied under many other criteria. For example, Vallada et al. [30] review 40 heuristics and metaheuristics for tardiness-related criteria. Similarly, a recent review of multi-objective approaches is given in Minella et al. [31].

Despite the enormous existing literature for the PFSP, we are not aware of any study involving the distributed version or DPFSP. Furthermore, the literature on other different distributed scheduling problems is scant and in its infancy. The distributed job shop problem under different criteria is first studied in two related papers by Jia et al. [32] and Jia et al. [33], where a genetic algorithm (GA) is proposed. However, the authors centre their study on the non-distributed problem as only a single 6-job, 3-factory problem, each one with four machines is solved. The proposed genetic algorithm is rather standard (apart from the solution representation, which includes job factory assignment) and the authors do not present any local search operator or advanced schemes. Later, a related short paper is that of Jia et al. [34], where the authors refine the previous GA. Distributed job shops with makespan criterion are also studied with the aid of GAs by Chan et al. [35]. This time, a more advanced GA is proposed and the authors test problems of up to 80 jobs with three factories, each one with four machines. In a similar paper [36], the same methodology is applied to the same problem, this time with maintenance decisions. A recent book about distributed manufacturing has been published [37] where one chapter also deals with the distributed job shop. However, the book is more centered on planning and manufacturing problems rather than production scheduling.

To the best of our knowledge, no further literature exists on distributed scheduling. Apart from the previous cited papers, that study only the job shop problem (and with some limitations), no studies could be found about the DPFSP.

3. Mixed integer linear programming models

Mathematical models are the natural starting point for detailed problem characterization. As a solution technique for scheduling problems, it is usually limited to small problem sizes. However, this has not deterred authors from studying different model proposals. The literature of PFSP is full of papers with MILP models. The initial works of Wagner [38], Manne [39], Stafford [40], Wilson [41] are just a few examples. One could think that MILP modeling is no longer of interest to researchers. However, recent works like those of Pan [42], Tseng et al. [43], Stafford et al. [44] or more recently, Tseng and Stafford [45], indicate that authors are still actively working in effective MILP modeling for flowshop related problems.

In the previously cited literature about MILP models for the PFSP, it is not completely clear which models are the best performers in all situations. Therefore, in this section we present six alternative models. Recall that the DPFSP includes the additional problem dimension of job factory assignment. In the following proposed models, such dimension is modeled with different techniques, ranging from a substantial additional number of binary variables (BVs) to more smart models where no additional variables are needed.

Before presenting each model, we introduce the parameters and indices employed. The common parameters and indices for all six proposed models are defined in Table 1.

Table 1
Parameters and indices used in the proposed models.

Parameter	Description
n	Number of jobs
m	Number of machines
g	Number of factories
j, k	Index for jobs (or job positions in a sequence); $j,k \in \{1,2,\ldots,n\}$
i, l	Index for machines; $i,l \in \{1,2,\ldots,m\}$
f	Index for factories; $f \in \{1,2,\ldots,F\}$
$O_{j,i}$	Operation of job j at machine i
$P_{j,i}$	Processing time of $O_{j,i}$
M	A sufficiently large positive number

The objective function for all models is makespan minimization:
Objective:

$$\mathrm{Min}\, C_{\max} \tag{1}$$

3.1. Model 1 (sequence-based distributed permutation flowshop scheduling)

The first model is sequence-based meaning that binary variables (BV) are used to model the relative sequence of the jobs. Note that we introduce a dummy job 0 which precedes the first job in the sequence. We need to state that index k denotes job positions and starts from 0 due to the starting dummy job 0. The variables used in this model are:

$X_{k,j,f}$ Binary variable that takes value 1 if job j is processed in factory f immediately after job k, and 0 otherwise, where $j \neq k$ (2)

$Y_{j,f}$ Binary variable that takes value 1 if job j is processed in factory f, and 0 otherwise. (3)

$C_{j,i}$ Continuous variable for the completion time of job j on machine i. (4)

Model 1 for the DPFSP continues as follows:
Objective function: Eq. (1)
Subject to:

$$\sum_{k=0,k\neq j}^{n}\sum_{f=1}^{F} X_{k,j,f} = 1 \quad \forall_j \tag{5}$$

$$\sum_{f=1}^{F} Y_{j,f} = 1 \quad \forall_j \tag{6}$$

$$\sum_{k=1,k\neq j}^{n} (X_{k,j,f} + X_{j,k,f}) \leq 2 \cdot Y_{j,f} \quad \forall_{j,f} \tag{7}$$

$$\sum_{j=1,j\neq k}^{n}\sum_{f=1}^{F} X_{k,j,f} \leq 1 \quad \forall_{k\in\{1,\ldots,n\}} \tag{8}$$

$$\sum_{j=1}^{n} X_{0,j,f} = 1 \quad \forall_f \tag{9}$$

$$\sum_{f=1}^{F} (X_{k,j,f} + X_{j,k,f}) \leq 1 \quad \forall_{k\in\{1,\ldots,n-1\},j>k} \tag{10}$$

$$C_{j,i} \geq C_{j,i-1} + P_{j,i} \quad \forall_{j,i} \tag{11}$$

$$C_{j,i} \geq C_{k,i} + P_{j,i} + \left(\left(\sum_{f=1}^{F} X_{k,j,f}\right) - 1\right) \cdot M \quad \forall_{k\in\{1,\ldots,n\},j,i,j\neq k} \tag{12}$$

$$C_{\max} \geq C_{j,m} \quad \forall_j \tag{13}$$

$$C_{j,i} \geq 0 \quad \forall_{j,i} \tag{14}$$

$$X_{k,j,f} \in \{0,1\} \quad \forall_{k,j,i,j\neq k} \tag{15}$$

$$y_{j,f} \in \{0,1\} \quad \forall_{j,f} \tag{16}$$

Note that $C_{j,0}=0$, $\forall j$. Constraint set (5) ensures that every job must be exactly at one position and only at one factory. Constraint set (6) assures that every job must be exactly assigned to one factory. Constraint set (7) states that every job can be either a successor or predecessor in the factory to which it is assigned. Constraint set (8) indicates that every job has at most one succeeding job whereas Constraint set (9) controls that dummy job 0 has exactly one successor at each factory. Constraint set (10) avoids the occurrence of cross-precedences, meaning that a job cannot be at same time both a predecessor and a successor of another job. Constraint set (11) forces that for every job j, $O_{j,i}$ cannot begin before $O_{j,i-1}$ completes. Similarity, Constraint set (12) specifies that if job j is scheduled immediately after job k its processing on each machine i cannot begin before the processing of job k on machine i finishes. It is necessary to point out that Constraint set (11) guarantees that each job j is processed by at most one machine at a time while Constraint set (12) enforces that a machine can process at most one job at a time. Constraint set (13) defines the makespan. Finally, Constraint sets (14)–(16) define the domain of the decision variables.

The number of BVs of Model 1 is quite large and amounts to $n^2F+nF-nF=n^2F$. The first term is the number of BVs generated by the jobs and machines. The second term is the number of BVs added by dummy jobs 0 while the third term is the number of BVs reduced by avoiding impossible assignments, for example, it is impossible that job j is processed immediately after the same job j. This model has nm continuous variables (CVs). The number of constraints required by Model 1 to represent a problem with n jobs, m machines and F factories is $4n+F+nF+n^2m+\frac{1}{2}n(n-1)$.

3.2. Model 2 (position-Wagner's-based distributed permutation flowshop scheduling)

Wagner's [38] models have been recently recognized by Stafford, Tseng and Gupta [44] and by Tseng and Stafford [45] as some of the best performing models for the regular PFSP. Therefore, our second proposed model is position-based following the ideas of Wagner [38]. This means that BVs just model the position of a job in the sequence. A very interesting feature of Wagner's [38] models is that they do not need the big M and the resulting dichotomous constraints which are known to produce extremely poor linear relaxations. Of course, we modify the model to include the job factory assignment of the DPFSP. Model 2 consists of the following variables:

$X_{j,k,f}$ Binary variable that takes value 1 if job j occupies position k in factory f, and 0 otherwise. (17)

$I_{i,k,f}$ Continuous variable representing the time that machine i idles after finishing the execution of the job in position k of factory f and before starting processing of job in position $k+1$, where $k<n$. (18)

$W_{i,k,f}$ Continuous variable representing the time that job in position k of factory f waits after finishing its execution on machine i and before starting at machine $i+1$, where $i<m$. (19)

C_f Continuous variable for the makespan of factory f. (20)

Model 2 is then defined as:
Objective function: Eq. (1)
Subject to:

$$\sum_{k=1}^{n}\sum_{f=1}^{F} X_{j,k,f}=1 \quad \forall_j \tag{21}$$

$$\sum_{j=1}^{n}\sum_{f=1}^{F} X_{j,k,f}=1 \quad \forall_k \tag{22}$$

$$I_{i,k,f}+\sum_{j=1}^{n} X_{j,k+1,f}\cdot P_{j,i}+W_{i,k+1,f}-W_{i,k,f} - \sum_{j=1}^{n} X_{j,k,f}\cdot P_{j,i+1}-I_{i+1,k,f}=0 \quad \forall_{k<n,i<m,f} \tag{23}$$

$$C_f=\sum_{i=1}^{m-1}\sum_{j=1}^{n} X_{j,1,f}\cdot P_{j,i}+\sum_{k=1}^{n-1} I_{m,k,f}+\sum_{j=1}^{n}\sum_{k=1}^{n} X_{j,k,f}\cdot P_{j,m} \quad \forall_f \tag{24}$$

$$C_{\max}\ge C_f \quad \forall_f \tag{25}$$

$$I_{i,k,f}\ge 0 \quad \forall_{i,k<n,f} \tag{26}$$

$$W_{i,k,f}\ge 0 \quad \forall_{k,i<m,f} \tag{27}$$

$$X_{j,k,f}\in\{0,1\} \quad \forall_{j,k,f} \tag{28}$$

Note that $I_{1,k,f}=0$, $\forall_{k<n,f}$, and $W_{i,1,f}=0$, $\forall_{i<m,f}$. With Constraint set (21), we enforce that every job must exactly occupy exactly one position in the sequence. Constraint set (22) states that n positions among all the nF possible must be occupied. Constraint set (23) forces the construction of feasible schedules; in other words, it works similarly as Constraints sets (11) and (12) from Model 1. However, in this case, feasible schedules are constructed with the waiting and idle continuous variables as in the original model of Wagner [38]. Constraint set (24) calculates the makespan at each factory while Constraint set (25) computes the general makespan of the whole problem. Constraint sets (26)–(28) define the decision variables. The number of BVs of Model 2 is n^2F. Compared with Model 1, it has the same number of BVs but many more CVs. Model 2 has $F(1+2nm-n-m)$ CVs and $\{2n+(n-1)(m-1)F+2F\}$ constraints.

3.3. Model 3 (position-based distributed permutation flowshop scheduling)

This model is position-based like Model 2 but tries to reduce the number of continuous variables just like Model 1 but also without the dichotomous constraints like Model 2. To calculate the makespan, we employ constraints similar to those of Model 1 (Constraints sets (11) and (12)). We need, however, to change the definition of $C_{j,i}$ as:

$C_{k,i,f}$ Continuous variable representing the completion time of the job in position k on machine i at factory f. (29)

Thus, the variables of Model 3 are (17) and (29). The rest of Model 3 goes as follows:
Objective function: Eq. (1)
Subject to:

$$C_{k,i,f}\ge C_{k,i-1,f}+\sum_{j=1}^{n} X_{j,k,f}\cdot P_{j,i} \quad \forall_{k,i,f} \tag{30}$$

$$C_{k,i,f}\ge C_{k-1,i,f}+\sum_{j=1}^{n} X_{j,k,f}\cdot P_{j,i} \quad \forall_{k>1,i,f} \tag{31}$$

$$C_{\max}\ge C_{k,m,f} \quad \forall_{k,f} \tag{32}$$

$$C_{k,i,f}\ge 0 \quad \forall_{k,i,f} \tag{33}$$

Note that $C_{k,0,f}=0$. We also need Constraint sets (21), (22) and (28). Constraint set (30) controls that the processing of job in position k of factory f at each machine can only start when the processing of the same job on the previous machine is finished. Constraint set (31) ensures that each job can start only after the previous job assigned to the same machine at the same factory is completed. Notice that this previous job might not be exactly in the previous position but in any preceding position. Constraint set (32) formulates the makespan. Lastly, Constraint sets (28) and (33) define the decision variables. Model 3 has the same number of BVs as Model 2 does while it has much fewer CVs. Model 3 needs nmF CVs and $2n+F(2nm-m+n)$ constraints.

3.4. Model 4 (reduced position-based distributed permutation flowshop scheduling)

This model is still position-based like Models 2 and 3. In Models 2 and 3, the two decision dimensions of the DPFSP are jointly considered in a single variable with three indices. That is, the determination of the job assignment to factories and job sequence at each factory are put together in a set of BVs with three indexes ($X_{j,k,f}$). In Model 4, we aim at further reducing the number of both BVs and CVs at the expense of more constraints. To this end, we employ a separated strategy to tackle the two decisions. In other words, we define two different sets of BVs, each one of them dealing with the job sequencing and job factory assignment, respectively. Furthermore, we reduce the possible job positions from nF to n. The variables of Model 4 are as follows:

$X_{j,k}$ Binary variable that takes value 1 if job j occupies position k, and 0 otherwise. (34)

$Y_{k,f}$ Binary variable that takes value 1 if the job in position k is processed in factory f, and 0 otherwise. (35)

$C_{k,i}$ Continuous variable representing the completion time of the job in position k on machine i. (36)

We need to notice that the variables defined in (35) are the adaptation of the sequence-based variables defined in (3) to the case of position-based variables, i.e., in (3), we determine for every job j, the factory to which it is assigned while in (35), we determine the factory to which the job in position k of the sequence was assigned. The rest of Model 4 goes as follows:
Objective function: Eq. (1)
Subject to:

$$\sum_{k=1}^{n} X_{j,k}=1 \quad \forall_j \tag{37}$$

$$\sum_{j=1}^{n} X_{j,k} = 1 \quad \forall_k \tag{38}$$

$$\sum_{f=1}^{F} Y_{k,f} = 1 \quad \forall_k \tag{39}$$

$$C_{k,i} \geq C_{k,i-1} + \sum_{j=1}^{n} X_{j,k} \cdot P_{j,i} \quad \forall_{k,i} \tag{40}$$

$$C_{k,i} \geq C_{l,i} + \sum_{j=1}^{n} X_{j,k} \cdot P_{j,i} - M(1 - Y_{k,f}) - M(1 - Y_{l,f}) \quad \forall_{k>1,l<k,i,f} \tag{41}$$

$$C_{\max} \geq C_{k,m} \quad \forall_k \tag{42}$$

$$C_{k,i} \geq 0 \quad \forall_{k,i} \tag{43}$$

$$X_{j,k} \in \{0,1\} \quad \forall_{j,k} \tag{44}$$

$$Y_{k,f} \in \{0,1\} \quad \forall_{k,f} \tag{45}$$

where $C_{k,0} = 0$. Constraint set (37) ensures that every job must occupy exactly one position. Constraint set (38) forces that each position must be assigned exactly once. Constraint set (39) controls that each job is assigned to exactly one factory. Constraint sets (40) and (41) serve the same purpose of the previous Constraint sets (30) and (31). Similar to Constraint set (32), Constraint set (42) calculates makespan. Constraint sets (43)–(45) define the decision variables. Interestingly, the number of BVs and CVs required for Model 4 are reduced to n^2+nF and nm, respectively. At the expense of this reduction in BVs and CVs, we need a larger number of $4n+nm+\frac{1}{2}n(n-1)mF$ constraints.

3.5. Model 5 (minimal sequence-based distributed permutation flowshop scheduling)

So, far, the only sequence-based model is Model 1. With Model 5 we intend to add further refinement. Model 5 can solve the DPFSP without actually indexing the factories. Since we use sequence-based variables, we need to define dummy jobs 0. Besides the variables defined in (4), Model 5 has the following additional variables:

$X_{k,j}$ Binary variable that takes value 1 if job j is processed immediately after job k; and 0 otherwise. (46)

In this model, we exploit the existence of this dummy job 0 in such a way that it partitions a complete sequence into F parts each of which corresponds to a factory. This is done through F repetitions of dummy job 0 in the sequence along with the other jobs. Therefore, this model searches the space with a sequence including n+F positions. One of these repetitions takes place in the first position of the sequence. All the subsequent jobs until the second repetition of dummy job 0 are scheduled in factory 1 with their current relative order. The jobs between the second and third repetitions of dummy job 0 are those allocated to factory 2. This repeats for all the subsequent repetitions of dummy job 0. The jobs after the F-th repetition of dummy job 0 until the last job in the sequence are those assigned to factory F. For example, consider a problem with $n = 6$ and $F = 2$, one of the possible solutions is $X_{0,1} = X_{1,3} = X_{3,6} = X_{6,0} = X_{0,4} = X_{4,2} = X_{2,1} = X_{1,5} = 1$; that is, {0, 3, 6, 0, 4, 2, 1, 5}. In this example, jobs 3 and 6 are allocated to factory 1 with this order {3, 6} while the other jobs are assigned to factory 2 with the permutation or sequence {4, 2, 1, 5}. The rest of Model 5 is now detailed:

Objective function: Eq. (1)

Subject to:

$$\sum_{k=0,k\neq j}^{n} X_{k,j} = 1 \quad \forall_j \tag{47}$$

$$\sum_{j=0,k\neq j}^{n} X_{k,j} \leq 1 \quad \forall_k \tag{48}$$

$$\sum_{j=1}^{n} X_{0,j} = F \tag{49}$$

$$\sum_{k=1}^{n} X_{k,0} = F - 1 \tag{50}$$

$$X_{k,j} + X_{j,k} \leq 1 \quad \forall_{k\in\{1,2,\ldots,n-1\},j>k} \tag{51}$$

$$C_{j,i} \geq C_{j,i-1} + p_{j,i} \quad \forall_{j,i} \tag{52}$$

$$C_{j,i} \geq C_{k,i} + p_{j,i} + (X_{k,i} - 1) \cdot M \quad \forall_{j,k,i,k\neq j} \tag{53}$$

$$X_{k,j} \in \{0,1\} \quad \forall_{k,j,j\neq k} \tag{54}$$

Note that $C_{k,0} = C_{0,l} = 0$, and that we also need Constraint sets (13) and (14). Constraint sets (47), (48), (51)–(53) work in a similar way as Constraint sets (5), (8), (10)–(12), respectively. Constraint set (49) enforces that dummy job 0 appears F times in the sequence as a predecessor where Constraint set (50) assures dummy job 0 must be a successor F–1 times. Constraint set (54), accompanied by Constraint sets (13) and (14), define the decision variables. Model 5 has a number of $(n+1)^2 - (n+1) = n(n+1)$ BVs, and nm CVs while it needs $3n + 2 + nm(1+n) + \frac{1}{2}n(n-1)$ constraints. Notice that dichotomous constraints are needed in this case. Later, when evaluating the performance of the models, we will see if this original and compact model is actually more efficient.

3.6. Model 6 (sequence-Manne's-based distributed permutation flowshop scheduling)

The last model is sequence-based again. To modelize scheduling problems, Manne [39] proposed another definition of BVs in such a way that they just report whether job is processed after job (not necessarily immediately) in the sequence or not. Through this definition, the number of necessary BVs is reduced. Another advantage of this definition is that dummy jobs 0 are not needed. Model 6 incorporates all these ideas and applies them to the DPFSP. Besides Constraint sets (3) and (4), Model 6 has the following variables:

$X_{k,j}$ Binary variable that value 1 if job j is processed after job k, and 0 otherwise. $k = \{1, 2, \ldots, n-1\}$, $k < j$ (55)

The remaining Model 6 is now detailed:

Objective function: Eq. (1)

Subject to:

$$C_{k,i} \geq C_{j,i} + P_{k,i} - M \cdot X_{k,j} - M(1 - Y_{k,f}) - M(1 - Y_{j,f}) \quad \forall_{i,k\in\{1,2,\ldots,n-1\},j>k,f} \tag{56}$$

$$C_{j,i} \geq C_{k,i} + P_{j,i} - M(1 - X_{k,j}) - M(1 - Y_{k,f}) - M(1 - Y_{j,f}) \quad \forall_{i,k\in\{1,2,\ldots,n-1\},j>k,f} \tag{57}$$

$$X_{k,j} \in \{0,1\} \quad \forall_{k\in\{1,2,\ldots,n-1\},j>k} \tag{58}$$

where $C_{j,0}$ = 0. Constraint sets (6), (11), (13), (14) and (16) are needed as well. Constraint sets (56) and (57) are the dichotomous pairs of

Table 2
Processing times for the example.

Machines	Jobs				
	1	2	3	4	5
1	10	6	8	9	3
2	5	7	4	6	11

constraints relating each possible job pair. Constraint set (58) defines the decision variables. As a result of the precious definitions, Models 5 and 6 have the lowest number of BVs among the proposed models. Model 6 has $\frac{1}{2}n(n-1)+nF$ BVs, nm CVs and $2n+nm+nmF(n-1)$ constraints.

4. Heuristic methods

Mathematical programming, although invaluable to precisely define a given problem, is limited to small instances as we will show later in the experimental sections. In order to cope with this limitation, we develop 12 heuristics based on six well-known existing heuristics from the regular PFSP literature. These are: shortest processing time (SPT) and largest processing time (LPT) [2], Johnson's rule from Johnson [1], the index heuristic of Palmer [13], the well-known CDS method of Campbell, Dudek and Smith [14] and the most effective heuristic known for the PFSP, the NEH algorithm of Nawaz, Enscore and Ham [15].

Recall that the DPFSP requires dealing with two inter-dependent decisions: (1) allocation of jobs to factories and (2) determination of the production sequence at each factory. Obviously, the previously cited six existing heuristics from the regular PFSP literature only deal with the sequencing decision. The allocation dimension of the DPFSP can be easily added to the six heuristics by including a factory assignment rule that is applied after sequencing. We tested several possible rules, from simple to complicated ones. After some experimentation, the following rules provided the best results while still being simple to implement:

(1) Assign job j to the factory with the lowest current $C_{\max}$, not including job j.
(2) Assign job j to the factory which completes it at the earliest time, i.e., the factory with the lowest $C_{\max}$, after including job j.

Although the previous rules are fairly simple, we still need a better illustration in order to reduce the ambiguity. Suppose a DPFSS problem with $F=2$, $n=5$ and $m=2$. Table 2 presents the processing times of the jobs on each machine (let us remind that factories are identical). Let us also suppose that we have the following job sequence provided by a heuristic: $\theta=\{3, 5, 1, 4, 2\}$ for which we want to find the makespan value. We first show the application of the rule 1 and then rule 2.

Rule 1—When processing job 3, the first in the sequence, there is no difference in allocating it to factories 1 or 2. Therefore, factory 1 is arbitrarily selected. After scheduling job 3, the next job to process is job 5. Since no job has been so far allocated to factory 2, it has the completion time of 0, while this amount for factory 1 is 12 time units. Therefore, job 5 is allocated to factory 2. The next job in sequence θ is job 1. The completion times of factories 1 and 2, prior to scheduling job 1, are 12 and 14 time units, respectively. So, job 1 is assigned to factory 1 because it has the lowest partial makespan. The procedure repeats similarly for all unscheduled jobs. Fig. 1 shows the resulting Gantt chart for the solution obtained by applying Rule 1 to the sequence θ.

Rule 2—Job 3 is arbitrarily allocated to factory 1. Now, to schedule job 5, we must allocate it to the factory which finishes job 5 in

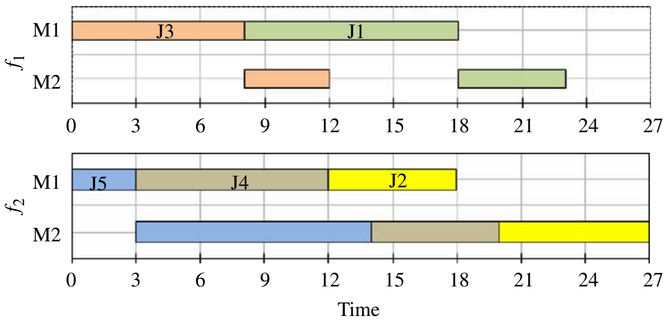

Fig. 1. The Gantt chart of the solution produced by Rule 1.

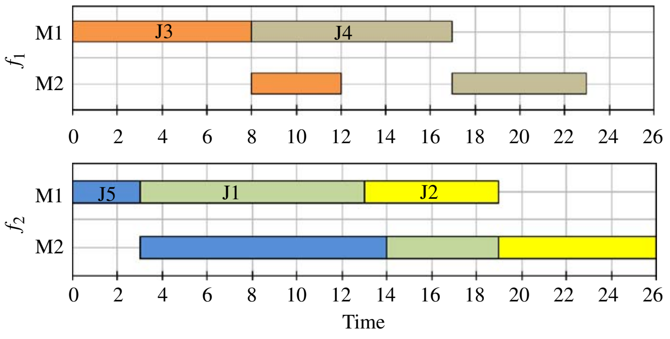

Fig. 2. The Gantt chart of the solution produced by Rule 2.

earliest time. If we assign job 5 to factory 1 (factory 2), its completion time becomes 23 (14) time units; consequently, factory 2 is chosen. The next job (1) is tested in factory 1 and results in a completion time of 23 units. However, assigning job 1 to factory 2 results in 19 units completion time. Therefore, job 1 is assigned to factory 2. This process is repeated for remaining jobs. Fig. 2 shows the Gantt chart of the solution obtained by Rule 2.

It is easy to see that the first rule has no added computational cost and the second one requires to sequence all m tasks of job j at all factories, i.e., $O(mF)$. We take the six original PFSP heuristics and add the two previous job factory assignment rules, thus resulting in 12 heuristics. We add the suffix 1 (respectively 2) to the heuristics that use the job factory assignment Rule 1 (respectively, 2). Note that in all heuristics (with the exception of NEH), the job sequence is first determined by the heuristic algorithm itself and then jobs are assigned to factories by applying the corresponding rules. For more details on each heuristic, the reader is referred to the original papers. Additionally, in [7], a comprehensive computational evaluation of these heuristics and other methods for the regular PFSP was provided.

Lastly, as regards the NEH method for the DPFSP, more details are needed. The NEH can be divided into three simple steps: (1) the total processing times for the jobs on all machines are calculated: $p_j=\sum_{i=1}^{m} p_{ij}, j=1,2,\ldots,n$. (2) Jobs are sorted in descending order of p_j. (3) If factory assignment Rule 1 is to be applied, then take job j, $j=(1,\ldots n)$ and assign it to the factory with the lowest partial makespan. Similarly to the original NEH, once assigned, the job is inserted in all possible positions of the sequence of jobs that are already assigned to that factory. With the second factory assignment rule, each job is inserted in all possible positions of all existing factories. The job is finally placed in the position of the factory resulting in the lowest makespan value. Notice that for both versions of the NEH heuristic, we employ the accelerations of Taillard [46].

5. Variable neighborhood descent algorithms

Variable neighborhood search (VNS) was proposed by Mladenovic and Hansen [47] and is a simple but effective local search method based on the systematic change of the neighborhood during the search process. Local search methods have been profusely employed in the regular PFSP literature with very good results, like the iterated local search of Stützle [48] and the iterated greedy method of Ruiz and Stützle [28]. However, all these methods are based on the exploration of a single neighborhood structure (and more precisely, the insertion neighborhood). VNS is based on two important facts: (1) a local optimum with respect to one type of neighborhood is not necessarily so with respect to another type, and (2) a global optimum is a local optimum with respect to all types of neighborhoods.

Among the existing VNS methods, reviewed again recently by Hansen and Mladenovic [12], the most simple one is the so called variable neighborhood descent (VND). Since in this paper the DPFSP is approached for the first time, we are interested in evaluating the performance of simple methods, and hence we chose the simple VND scheme with only two neighborhood structures. In VND, typically the different neighborhoods are ordered from the smallest and fastest to evaluate, to the slower one. Then the process iterates over each neighborhood while improvements are found, doing local search until local optima at each neighborhood. Only strictly better solutions are accepted after each neighborhood search. Our proposed VND algorithms are deterministic and incorporate one local search to improve the job sequence at each factory, and another one to explore job assignments to factories. In the following sections we further detail the proposed VND methods.

5.1. Solution representation and VND initialization

Solution representation is a key issue for every metaheuristic method. In the regular PFSP, a job permutation is the most widely used representation. Recall that the DPFSP adds the factory assignment dimension and therefore, such representation does not suffice. We propose a representation composed by a set of F lists, one per factory. Each list contains a partial permutation of jobs, indicating the order in which they have to be processed at a given factory. Notice that this representation is complete and indirect. It is complete because all possible solutions for the DPFSP can be represented and it is indirect since we need to decode the solution in order to calculate the makespan.

Starting VND from a random solution quickly proved to be a poor decision since the method rapidly converges to a local optimum of poor quality. Furthermore, effective heuristic initialization of algorithms is a very common approach in the scheduling literature. As a result, we initialize the proposed VND from the NEH2 method explained before.

5.2. Neighborhoods, local search and acceptance criteria

As mentioned above, we employ two different neighborhoods. The first one is aimed at improving the job sequence inside each factory. As usual in the PFSP literature, the insertion neighborhood, in which each job in the permutation is extracted and reinserted in all possible positions, has given the best results when compared to swap or interexchange methods. Furthermore, as shown by Ruiz and Stützle [28], among many others, a single evaluation of this neighborhood can be done very efficiently if the accelerations of Taillard [46] are employed. We refer to LS_1 as the local search procedure used in the proposed VND for improving job sequences at each factory. LS_1 works as follows: all sequences obtained by removing one single job and by inserting it into all possible positions inside the factory processing that job are tested. The sequence resulting in the lowest $C_{\max}$ is accepted and the job is relocated. If the $C_{\max}$ for that given factory is improved after the reinsertion, all jobs are reinserted again (local search until local optima). Otherwise, the search continues with the next job. Fig. 3 shows the procedure of LS_1.

```
Procedure: LS_1

Improvement := true
while improvement do
    improvement := false
    for $i = 1$ to $f_f$ do                    %$f_f$ is the number of jobs in $f$-th factory
        Remove job $k$ in the $i$-th position of the $f$-th factory
        Test job $k$ in all the $f_f$ positions of the $f$-th factory
        Place job $k$ in the position resulting in the lowest $C_{max}$
        if $C_{\max}$ is strictly improved then
            improvement := true
            break                              % so that all jobs are examined again
        endif
    endfor
endwhile
```

Fig. 3. Outline of local search based on the insertion neighborhood (LS_1).

The second proposed neighborhood and associated local search procedure for the VND is referred to as LS_2 and works over the job factory assignment dimension. Reassignment of all jobs to all factories is a lengthy procedure. Furthermore, in many cases, this will not improve the overall maximum $C_{\max}$. Notice that at least one factory will have a maximum completion time equal to $C_{\max}$. Let us refer to this factory generating the overall maximum $C_{\max}$ as y. Normally, in order to improve the $C_{\max}$ we need to reassign one job from y to another factory x, $x = \{1, 2, \ldots, F\}$, $x \neq y$. Therefore, in LS_2, all solutions obtained by the reallocation and insertion of all jobs assigned to y into all positions of all other factories are tested. For example, suppose we have $n = 8$ and $F = 3$, where $f_1 = 3, f_2 = 2$ and $f_3 = 3$. Let us also suppose that $y = 1$, i.e. factory 1 generates the overall maximum $C_{\max}$. Therefore, factory y has three jobs, and each one of them can be reinserted into 7 positions: 3 positions in factory 2 and 4 positions in factory 3. As a result the proposed LS_2 procedure evaluates 21 candidate solutions. Only one out of the candidate solutions is accepted if the acceptance criterion is satisfied.

The acceptance of movements in LS_2 is not straightforward. After all jobs are extracted from y and inserted in all positions of all other factories we end up with two new makespan values: the new best makespan at factory y, which is referred to as $C'_{\max}$ and the lowest makespan obtained after relocating all jobs to another factory x, $x = \{1, 2, \ldots, F\}, x \neq y$. Let us call this second makespan value C'_x. In order to decide the acceptance of the local search movement, $C'_{\max}$ and C'_x should be compared to those values prior to the movement, i.e., $C_{\max}$ and C_x, respectively. With this in mind, we have two possible acceptance criteria:

(1) Accept the movement if the general makespan is improved, i.e., if $C'_{\max} < C_{\max} \wedge C'_x < C_{\max}$.
(2) Accept the movement if there is a "net gain" between the two factories, i.e., if $(C_{\max} - C'_{\max}) + (C_x - C'_x) > 0$.

A priori, it is not clear which acceptance criterion will result in the best performance. Therefore, we propose two VND algorithms, referred to as VND(a) and VND(b), employing acceptance criteria (1) and (2) above, respectively. Fig. 4 shows the outline of LS_2.

Furthermore, each job reassignment of LS_2 affects two factories. If the incumbent solution is changed in LS_2, LS_1 is performed again according to the VND scheme. Obviously, only factories that had their jobs reassigned during LS_2 must undergo LS_1. This is controlled in both local search methods by means of some flags. In the first iteration of the VND, all factories undergo LS_1. Subsequent iterations require the application of LS_1 only to the factories affected by LS_2. Our proposed VND ends when no movement is accepted at LS_2. Fig. 5 shows the general framework of the proposed VND.

6. Experimental evaluation

In this section, we evaluate the proposed MILP models and the proposed algorithms on two sets of instances. The first set includes small-sized instances in order to investigate the effectiveness of the developed mathematical models and also the general performance of the proposed algorithms. In the second set, we further evaluate the performances of the algorithms against a larger benchmark which is based on Taillard [49] instances. We test the MILP models using CPLEX 11.1 commercial solver on an Intel Core 2 Duo E6600 running at 2.4 GHz with 2GB of RAM memory. All other methods have been coded in C++ and are run under Windows XP in the same computer.

```
Procedure: LS_2

for $i = 1$ to $f_y$ do
    Remove job $k$ in the $i$-th position of factory $y$
    for $f = 1$ to $F$ do
        if $f \neq y$ do
            Test job $k$ in all the $f_f$ positions of the $f$-th factory
        endif
    endfor
endfor
if acceptance criterion is satisfied then
    reassign the job from $y$ to another factory that resulted in the best movement
endif
```

Fig. 4. Outline of local search based on the factory job assignment neighborhood (LS_2).

6.1. Evaluation of the MILP models and heuristics on small-sized instances

The general performance of the MILP models and the proposed algorithms is evaluated with a set of small-sized instances which are randomly generated as follows: the number of jobs n is tested with the following values $n = \{4, 6, 8, 10, 12, 14, 16\}$ whereas $m = \{2, 3, 4, 5\}$ and $F = \{2, 3, 4\}$. There are 84 combinations of n, m, and F. We generate five instances for each combination for a total of 420 small instances. The processing times are uniformly distributed over the range $(1, 99)$ as usual in the scheduling literature. All instances, along with the best known solutions are available at http://soa.iti.es. We generate one different LP file for each instance and for each one of the six proposed models. Thus we analyze 2520 different MILP models. Each model is run with two different time limits: 300 and 1800 s. Lastly, starting with version 11, ILOG CPLEX allows for parallel branch and bound. Given that the Intel Core 2 Duo computer used in the experiments has two CPU cores available, we also run all models once with a single thread and another time with two threads (CPLEX SET THREADS 2 option). As a result, each one of the 2520 models is run four times (serial 300, serial 1800, parallel 300 and parallel 1800) for a total number of results of 10,080. We first compare the performance of the models, and then we evaluate the general performance of the heuristic algorithms.

Table 3 presents a summary of the six proposed models as regards the number of binary and continuous variables (BV and CV, respectively), number of constraints (NC) and whether the models

```
Procedure: Variable_Neiborhood_Descent_for_the_DPFSP

NEH2 Initialization
improvement := true
flag[$F$] = true;                         % flag for each factory
while improvement do
    for $f = 1$ to $F$ do
        if flag[$f$] do
            Perform LS_1 on $f$
            flag[$f$] := false
        endif
    endfor
    Find factory $y$                                  % $y$ is the factory with the highest $C_{\max}$
    Perform LS_2 on $y$
    if acceptance criterion is satisfied do        % solution changed
        flag[$y$] := true
        flag[$x$] := true             % factory $x$ is the one that received a job from $y$ during LS_2
    elseif
        improvement := false
    endif
endwhile
```

Fig. 5. Framework of the proposed VND algorithm for the DPFSP.

Table 3
Comparison of the different proposed models.

Model	Number of binary variables (BV)	Number of continuous variables (CV)	Number of constraints (NC)	Dichotomous constraints
1	n^2F	nm	$4n + F + nF + n^2m + \frac{1}{2}n(n-1)$	Yes
2	n^2F	$F(1{+}2nm{-}n{-}m)$	$2n + (n-1)(m-1)F + 2F$	No
3	n^2F	nmF	$2n + F(2nm - m + n)$	No
4	$n^2 + nF$	nm	$4n + nm + \frac{1}{2}n(n-1)mF$	Yes
5	$n(n{+}1)$	nm	$3n + 2 + nm(1 + n) + \frac{1}{2}n(n-1)$	Yes
6	$\frac{1}{2}n(n+1) + nF$	nm	$2n + nm + n(n-1)mF$	Yes

Table 4
Performance results of the different models for the small instances.

Model	Time limit (s)				Average
	300		1800		
	Serial	Parallel	Serial	Parallel	
1					
% opt	44.52	46.43	48.33	49.52	47.20
GAP%	15.75	15.69	14.60	14.52	15.14
Time	175.45	169.28	973.67	948.07	566.62
2					
% opt	44.05	46.67	50.24	54.05	48.75
GAP%	9.04	8.05	6.33	5.41	7.21
Time	179.82	169.76	957.84	899.96	551.84
3					
% opt	51.19	55.00	57.86	61.67	56.43
GAP%	7.12	6.20	4.82	4.57	5.68
Time	157.06	148.98	824.20	759.78	472.51
4					
% opt	47.86	50.00	53.81	56.67	52.08
GAP%	14.73	13.06	11.48	9.66	12.23
Time	165.10	158.44	890.00	847.88	515.35
5					
% opt	55.48	58.10	61.43	63.33	59.58
GAP%	12.69	12.20	11.06	10.67	11.66
Time	140.21	135.46	753.89	709.07	434.66
6					
% opt	53.10	57.14	58.10	62.62	57.74
GAP%	9.51	7.86	7.40	6.79	7.89
Time	147.31	138.27	796.67	733.39	453.91
Average					
% opt	49.37	52.22	54.96	57.98	53.63
GAP%	11.47	10.51	9.28	8.60	9.97
Time	160.82	153.36	866.04	816.36	499.15

contain dichotomous constraints which are known to have poor linear relaxations.

6.1.1. *Models' performance*

No model was able to give the optimum solution for all instances in the allowed CPU time. As a result, for each run we record a categorical variable which we refer to as "type of solution" with two possible values: 0 and 1. 0 means that an optimum solution was reached. The optimum $C_{\max}$ value and the time needed for obtaining it are recorded. 1 means that the CPU time limit was reached and at the end a feasible integer solution was found but could not be proved to be optimal. The gap is recorded in such a case. In all cases the models where able to obtain at least a feasible integer solution.

Overall, from the 420 instances, we were able to obtain the optimum solution in 301 (71.67%) cases after collecting the solutions of all models. From the 10,080 runs, in almost 54% of the cases the optimal solution was found. Table 4 presents, for each model, and for all runs, the percentage of optimum solutions found (% opt) the average gap in percentage for the cases where the optimum solution could not be found (GAP%) and the average time needed in all cases.

As expected, more CPU time results in a higher number of instances being solved to optimality. The same applies with the parallel version of CPLEX. In any case, neither of these two factors produce significant improvements. The total time needed for the 10,080 CPLEX runs was 58.24 days of CPU time. The type of model employed, however, seems to have a larger impact on results. On average, the lowest percentage of solved instances corresponds to Model 1 with 47.20%. Comparatively, Model 5 almost reaches 60%. I is also interesting to note that the lowest gap for unsolved instances are obtained with Models 2 and 3, which are the ones that do not employ dichotomous constraints. However, together with Model 1, these three models have the largest number of binary variables and hence their relative worse performance.

We now carry out a more comprehensive statistical testing in order to draw strong conclusions for the averages observed in the previous table. We use a not so common advanced statistical technique called automatic interaction detection (AID). AID bisects experimental data recursively according to one factor into mutually exclusive and exhaustive sets that explain the variability of a given response variable in the best statistically significant way. AID was proposed by Morgan and Sonquist [50] and was later improved by Biggs et al. [51] which developed an improved version which is known as exhaustive CHAID. AID techniques are a common tool in the fields of education, population studies, market research, psychology, etc. CHAID was recently used by Ruiz et al. [52] for the analysis of a MILP model for a complex non-distributed scheduling problem. We use the exhaustive CHAID method for the analysis of the MILP results presented before. The factors Model, time limit, serial or parallel run, n, m and F are studied. The response variable is the type of solution reached by CPLEX with two possible values (0 and 1). The process begins with all data in a single root node. All controlled factors are studied and the best multi-way split of the root node according to the levels of each factor is calculated. A statistical significance test is carried out to rank the factors on the splitting capability. The procedure is applied recursively to all nodes until no more significant partitions can be found. A decision tree is constructed from the splitting procedure. This tree allows a visual and comprehensive study of the effect of the different factors on the response variable. We use SPSS DecisionTree 3.0 software. We set a high confidence level for splitting of 99.9% and a Bonferroni adjustment for multi-way splits that compensates the statistical bias in multi-way paired tests. The first five levels of the decision tree are shown in Fig. 6. Each node represents the total percentage of instances solved to optimality (type 0) and the total number of instances in that node. A node ending in a dash "–" indicates that no further statistically significant splits were found. All node splits are strongly statistically significant with all p-values very close to 0 even for the last level splits.

As can be seen, n is, by far, the most influential factor. As a matter of fact, no model can solve instances with 14 or 16 jobs except in a very few cases, even with 30 min of CPU time and the most modern CPLEX parallel version 11.1 (at the time of the writing of this paper). After n, the model (represented in the tree as "Mod") is the second most influential factor in most situations. For $n = 10$ or 12 we see that the model that is able to solve the highest number of instances is Model 5. The number of factories F is also an influential factor. However, its effect depends on the type of model. For example, a larger number of factories is detrimental for Models 2 and 3 but beneficial to Model 5. This behavior is clear since the number of binary variables in Model 5 does not depend on F. Apart from some isolated cases, the effect of the CPLEX parallel version (two threads) is not statistically significant over the number of solved instances response variable. However, the time limit is obviously significant, but only at the last levels of the tree and for instances up to $n = 12$ where the gaps were small enough so that more CPU time could be adequately employed.

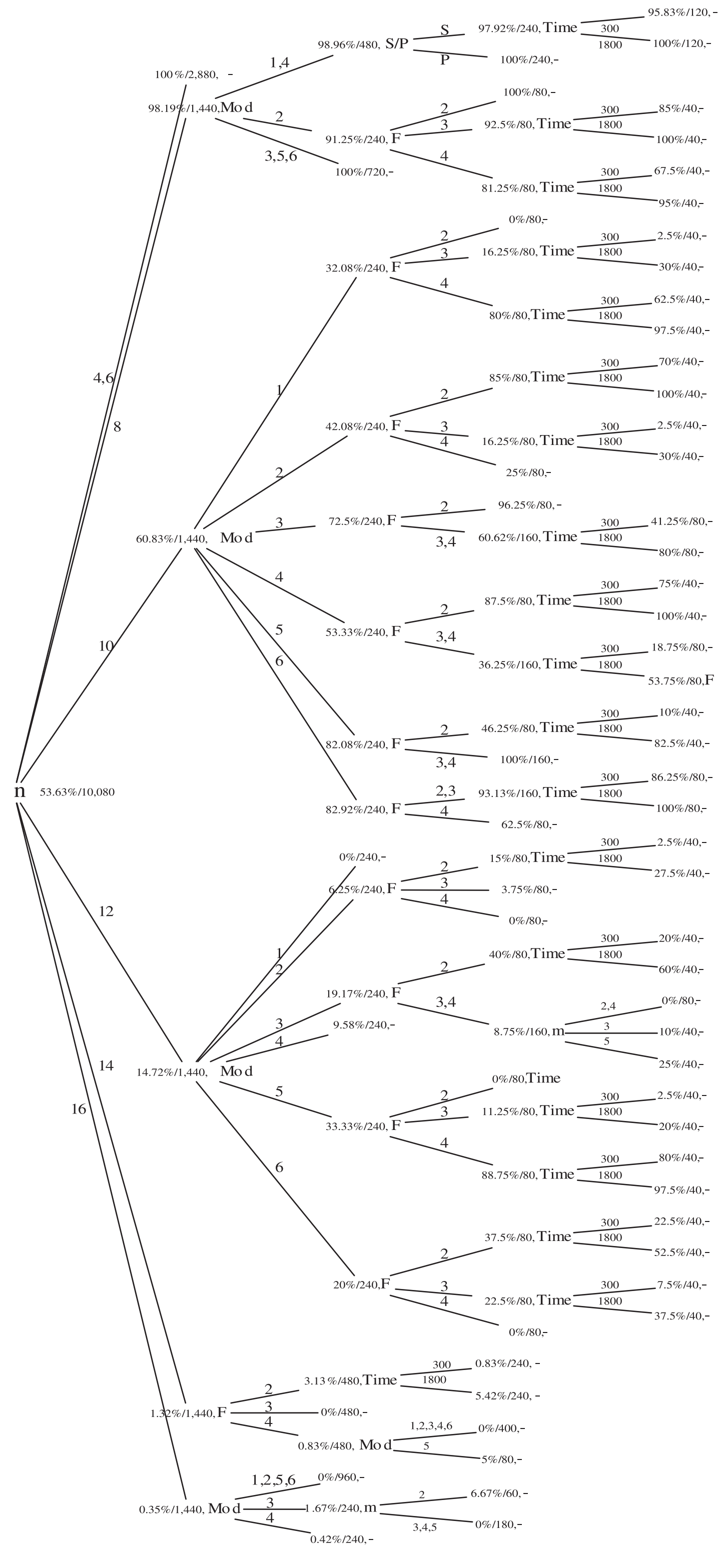

Fig. 6. Decision tree for the MILP models evaluation.

It is clear that the DPFSP is a formidable problem as regards MILP modeling. The limit for our proposed models seems to lie around $n = 12$ and $F \geq 3$.

6.1.2. Algorithms' performance

The proposed methods are heuristics and are not expected to find the optimum solution. Therefore, we measure a relative percentage deviation (*RPD*) from a reference solution as follows:

$$RPD = \frac{Alg_{sol} - OPT_{sol}}{OPT_{sol}} \times 100$$

where OPT_{sol} is the optimal solution found by any of the models (or best known MIP bound) and Alg_{sol} corresponds to the makespan obtained by a given algorithm for a given instance. The best solutions

Table 5
Relative percentage deviation (*RPD*) of the heuristic algorithms.

Instance $F{\times}n$	Algorithms						
	SPT1	LPT1	Johnson1	CDS1	Palmer1	NEH1	VND(a)
2×4	12.95	19.31	6.13	2.69	7.75	2.61	0.00
2×6	13.30	29.53	10.34	7.56	7.57	4.42	1.26
2×8	17.31	35.43	9.75	12.33	9.97	4.43	2.10
2×10	21.03	36.93	8.38	9.72	10.18	4.28	3.22
2×12	20.40	39.36	10.14	11.81	10.77	6.95	3.38
2×14	17.73	40.00	10.27	6.59	10.57	6.05	2.70
2×16	17.11	42.42	12.33	10.53	10.97	6.66	2.92
3×4	8.87	6.42	5.31	5.01	4.82	0.43	0.00
3×6	21.24	27.63	6.91	8.50	9.40	4.04	0.68
3×8	17.71	26.87	10.14	9.17	11.84	5.08	1.86
3×10	24.48	37.98	12.67	13.56	13.97	8.42	2.59
3×12	25.92	42.12	14.90	14.66	14.39	7.66	3.66
3×14	23.44	41.00	14.62	16.45	17.97	10.54	4.48
3×16	25.31	41.71	14.39	15.55	15.68	8.18	3.50
4×4	0.00	0.00	0.00	0.00	0.00	0.00	0.00
4×6	13.34	14.52	7.65	4.99	6.02	3.10	0.00
4×8	15.21	16.86	9.20	9.23	10.17	4.19	0.77
4×10	22.65	34.47	13.19	13.90	16.21	7.07	1.57
4×12	25.51	38.72	13.61	18.13	15.49	8.58	4.23
4×14	26.61	42.89	13.50	15.47	16.53	10.94	4.25
4×16	28.70	44.01	19.12	17.62	19.34	9.79	5.08
Average	**18.99**	**34.34**	**10.60**	**10.64**	**11.41**	**5.88**	**2.30**
	SPT2	LPT2	Johnson2	CDS2	Palmer2	NEH2	VND(b)
2×4	10.02	17.80	3.76	0.72	5.22	0.15	0.15
2×6	10.38	28.37	8.08	4.60	4.83	1.44	1.44
2×8	16.16	33.09	7.98	10.33	8.64	2.53	2.52
2×10	17.49	37.12	7.52	8.38	9.25	3.27	2.89
2×12	17.23	37.42	8.29	10.49	10.32	4.13	3.90
2×14	17.34	38.36	9.21	5.24	9.24	3.16	2.82
2×16	16.95	41.63	10.99	8.26	9.00	3.84	3.30
3×4	5.36	5.41	1.99	2.55	1.50	0.43	0.00
3×6	11.93	25.26	5.28	6.99	3.40	1.39	1.39
3×8	17.93	25.14	8.30	8.67	6.98	2.43	2.43
3×10	16.23	35.43	10.07	10.81	11.67	3.79	3.63
3×12	19.55	40.14	10.12	11.00	11.36	5.08	5.08
3×14	19.95	38.56	14.04	13.47	14.05	4.90	4.46
3×16	22.83	39.39	10.41	11.60	11.06	3.98	3.81
4×4	0.00	0.00	0.00	0.00	0.00	0.00	0.00
4×6	11.31	12.52	5.62	3.39	2.98	0.47	0.00
4×8	13.89	16.44	5.56	8.61	7.04	0.77	0.77
4×10	16.84	30.64	8.93	9.30	7.23	2.30	2.22
4×12	20.54	34.06	8.68	14.49	12.39	4.97	4.68
4×14	22.31	39.79	8.51	11.23	12.25	4.54	4.46
4×16	24.09	40.67	13.96	13.69	15.84	5.59	5.55
Average	**15.63**	**29.39**	**7.97**	**8.28**	**8.30**	**2.82**	**2.64**

Small instances.

for the instances are available from http://soa.iti.es. Table 5 summarizes the results grouped by each combination of *n* and *F* (20 data per average).

VND(a) provides solutions that deviate 2.30% over best known ones. VND(b) is a bit worse at 2.64%. Although these are just the results for the small instances, we see that the second acceptance criterion gives slightly worse results. We can also see how the much more comprehensive NEH2 gives much better results than NEH1. As a matter of fact, all heuristics with the second job factory assignment rule (recall that the second rule assigns each job to the factory that can complete it at the earliest possible time, instead of at the first available factory) produce better results in mostly all cases. Therefore we can conclude that the second job factory assignment rule is more effective.

Once again, we need to check if the differences observed in the previous table are statistically significant. We carry out a multifactor ANOVA over the results of all heuristics and all small instances (except LPT1 and LPT2 since they are clearly worse than all other methods). The ANOVA has 12,420 = 5040 data and all three hypotheses of the parametric ANOVA (normality, homocedasticity and independence of the residuals) are checked and satisfied. The response variable is the *RPD* and the controlled factors are the algorithm, *n*, *m* and *F*. We omit the presentation of the ANOVA table for reasons of space but all previous factors result in strong statistically significant differences in the *RPD* response variable with *p*-values very close to zero. On contrary than with the previous experiment with MILP models, in this case, the most significant factor (with an *F*-Ratio of almost 250) is the type of algorithm. Fig. 7 shows the corresponding means plot and honest significant difference (HSD) intervals at 99% confidence. Recall that overlapping intervals indicate that there is no statistically significant difference among the overlapped means.

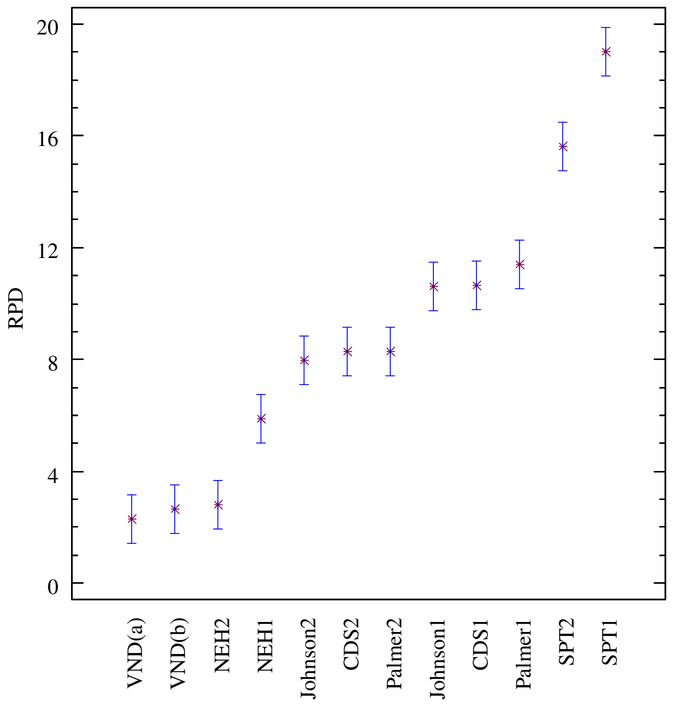

Fig. 7. Means plot and 99% confidence level HSD intervals for the algorithms and small instances.

As we can see, there is a large variance on results and the differences observed for VND(a), VND(b) and NEH2 are not statistically significant. However, this is the overall grand picture. For some specific instances we can find greater differences. However, and as shown, NEH1 is already statistically worse than NEH2.

Note that we are not reporting the CPU times employed by the heuristic algorithms for the small instances since they are extremely short and below our capability of measuring them.

6.2. Evaluation of the heuristics on large-sized instances

We now evaluate the performance of the 12 proposed heuristics and the two proposed VND methods over a set or larger instances. In the scheduling area, and more specifically, in the PFSP literature, it is very common to use the benchmark of Taillard [49] that is composed of 12 combinations of $n{\times}m$: {(20, 50, 100)×(5, 10, 20)}, {200×(10, 20)} and (500×20). Each combination is composed of 10 different replicates so there are 120 instances in total. We augment these 120 instances with seven values of $F = \{2, 3, 4, 5, 6, 7\}$. This results in

Table 6
Relative percentage deviation (*RPD*) of the heuristic algorithms. Large instances.

Instance (*F*)	Algorithms						
	SPT1	LPT1	Johnson1	CDS1	Palmer1	NEH1	VND(a)
2	18.71	33.70	12.68	9.29	10.44	2.92	0.10
3	19.95	33.05	13.40	10.83	11.72	3.42	0.10
4	20.07	33.30	13.48	11.19	11.95	4.21	0.06
5	20.04	32.89	13.18	11.29	12.24	4.28	0.11
6	20.32	32.58	13.57	11.50	12.70	4.73	0.11
7	21.04	32.02	13.61	11.49	12.53	4.86	0.10
Average	**20.02**	**32.92**	**13.35**	**10.93**	**11.93**	**4.07**	**0.10**
	SPT2	LPT2	Johnson2	CDS2	Palmer2	NEH2	VND(b)
2	17.71	32.36	11.58	8.52	9.34	1.21	0.32
3	17.58	31.57	11.18	8.92	9.43	1.15	0.35
4	17.23	30.42	10.67	8.54	8.90	1.11	0.46
5	16.73	29.57	10.24	8.07	8.76	0.92	0.46
6	16.07	28.55	9.94	7.83	8.45	0.95	0.51
7	15.42	27.14	9.71	7.31	8.21	0.81	0.45
Average	**16.79**	**29.93**	**10.55**	**8.20**	**8.85**	**1.03**	**0.43**

a total of 720 large instances. They are available for download at http://soa.iti.es.

We now evaluate the *RPD* of the heuristic methods. Note that since no optimal solutions or good MIP bounds are known, the *RPD* is calculated with respect to the best solution known (the best obtained among all heuristics). Table 6 presents the results of the experiments, averaged for each value of *F* (120 data per average).

As can be seen, VND(a) is the best performing algorithm with an *RPD* of just 0.10%. NEH1 and NEH2 provide the best results among the simple constructive heuristics with *RPD* values of 1.03% and 4.07%, respectively. Clearly, NEH2 obtains much lower *RPD* than NEH1.

Similarly to the analysis carried out with the small instances, we analyze the results of the ANOVA now with the larger instances. Fig. 8 shows the means plot for the algorithm factor.

As we can see, the confidence intervals are much narrower now. This is because there are many more samples (12 tested algorithms, each one on 720 instances for a total of 8640 data) and less variance in the results. VND(a) is almost statistically better than VND(b) and there are large differences among all other algorithms. In order to closely study the differences and performance trends among the proposed heuristics, we need to study interactions with the other factors. Sadly, in Taillard's [49] benchmark, the values of n and m are not orthogonal, since the combinations 200×5, 500×10 and 500×20 are missing. Therefore, it is not possible to study, in the same ANOVA experiment, all these factors with interactions. To cope with this, we study, separately, in three different ANOVA experiments, the effect of n, m and F in the algorithms. To make the graphs readable, we only consider the first four best performing algorithms, i.e., VND(a), VND(b), NEH2 and NEH1 since all others are clearly worse and by a significant margin. Fig. 9 (left) shows the interaction between the algorithms and n and Fig. 9 (right) between algorithms and m.

We can see that VND(a) and VND(b) are most of the time statistically equivalent as regards n and m. However, VND(a)'s average is consistently lower than that of VND(b) so it is expected that with more samples, VND(a) will eventually become statistically better than VND(b). For some cases, NEH2 is statistically equivalent to VND(b), although clearly better than NEH1. Interestingly, NEH1's performance improves with higher number of jobs and/or machines.

Lastly, Fig. 10 shows the interaction between the algorithms and the number of factories F. The situation is more or less similar to previous ANOVAs, with the clear difference that NEH1 deteriorates

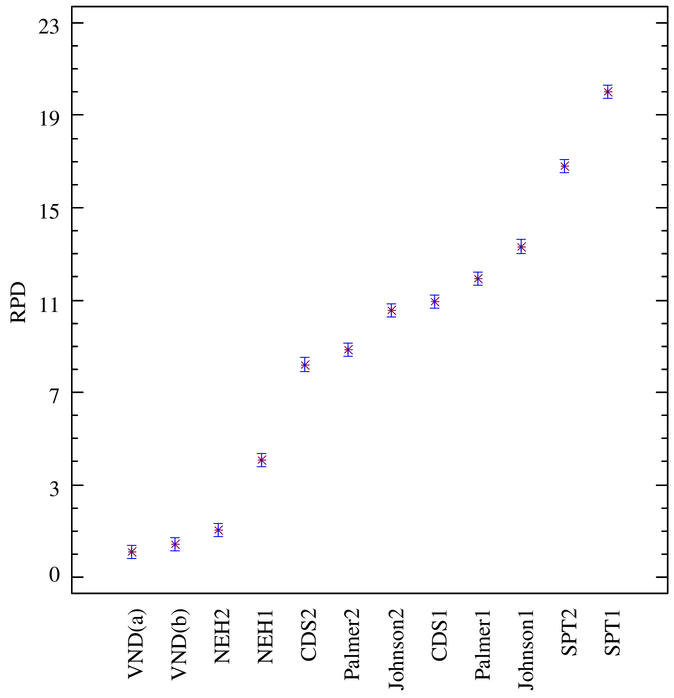

Fig. 8. Means plot and 99% confidence level HSD intervals for the algorithms and large instances.

with higher values of F. This is an expected result since NEH1 uses the first job factory assignment rule, which assigns jobs to the first available factory, instead of examining all possible factories. NEH1 is clearly faster than NEH2, but suffers when the instance contains a larger number of factories. Note that the plots of Figs. 9 and 10 are different experiments carried out with the objective of analyzing the effect of n, m and F over results. Fig. 8 shows the overall picture and from that picture, it is clear that VND(a) and VND(b) are superior than all other proposed methods.

Last but not least we comment on the CPU time employed by the best performing heuristics (the CPU time used by all other proposed heuristics is also negligible (less than 0.01 s), even for the largest instances of seven factories, 500 jobs and 20 machines.

Table 7 summarizes the CPU times of VND(a), VND(b), NEH2 and NEH1.

On average, we see that the CPU times are extremely short. The most expensive method is VND(a) and yet it needs, on average, less than 0.15 s. We can also see how NEH1 is almost instantaneous. We have grouped the results, for the sake of brevity, by F. However, it is well known in the PFSP literature that the most influential factor in CPU times is n. We have recorded the longest time each one of the methods needed to solve any instance. These times are 3.884, 1.154, 0.219 and 0.063 s for VND(a), VND(b), NEH2 and NEH1, respectively. Therefore, the proposed heuristics are very efficient in most of the cases. Obviously, VND(a) and VND(b) are slower than NEH2 and NEH1 (they should be, since they use them as a seed solution) but we can see that the added CPU time, although significant is, in practice, irrelevant.

7. Conclusions and future research

In this paper we have studied for the first time, to the best of our knowledge, a generalization of the well-known permutation flowshop scheduling problem or PFSP. This generalization comes after considering several production centers or shops. We have referred to this setting as the distributed permutation flowshop scheduling

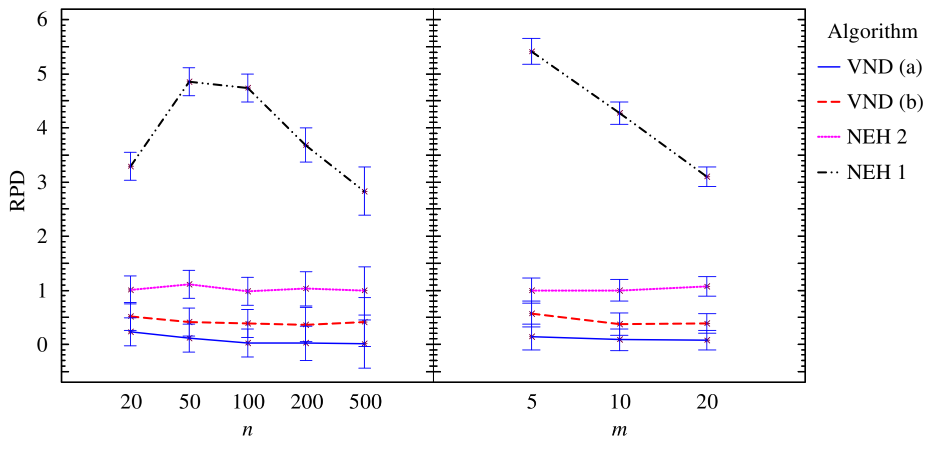

Fig. 9. Means plots and 99% confidence level HSD intervals for the interaction between the algorithms and n (left) and between algorithms and m (right). Large instances.

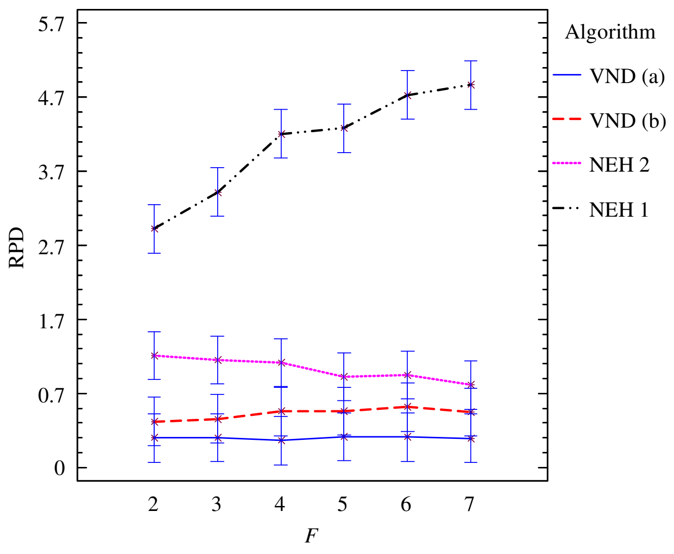

Fig. 10. Means plot and 99% confidence level HSD intervals for the interaction between the algorithms and F. Large instances.

Table 7
CPU time required by VND and NEH in both variants (s).

Instance (F)	Algorithms			
	VND(a)	VND(b)	NEH2	NEH1
2	0.246	0.120	0.017	0.009
3	0.182	0.105	0.020	0.007
4	0.129	0.104	0.024	0.007
5	0.114	0.086	0.028	0.006
6	0.120	0.083	0.031	0.005
7	0.091	0.076	0.035	0.006
Average	**0.147**	**0.096**	**0.026**	**0.007**

problem or DPFSP. In this problem two interrelated decisions are to be taken: assignment of jobs to factories and sequencing of the jobs assigned to each factory. We have characterized this problem by analyzing the total number of feasible solutions. We have also demonstrated that for maximum makespan minization, no factory should be left empty. We have also proposed six different alternative mixed integer linear programming models (MILP) and have carefully analyzed their performance. Among the proposed MILP models, an original sequence-based modelization has produced the best results, as shown by the careful statistical analyses. We have also proposed 14 heuristic methods, from adaptations of simple dispatching rules to effective constructive heuristics and fast local search methods based on variable neighborhood descent (VND). In order to cope with the job assignment to factories, we have presented two simple rules. The first is based on assigning a job to the first available factory and the second checks every factory to minimize the job completion time. We have shown that the second rule offers better results at a very small computational penalty. Among the proposed methods, the VND heuristics are both really fast (under 4 s in the worst case) and much more effective than all other proposed methods. All experiments have been carefully confirmed by sound statistical analyses. Note that the proposed VND heuristics are slower than all other proposed methods. However, they are under 4 s in the worst tested case and under 0.15 s, on average. Therefore, the added CPU time has little practical relevance.

Of course, there is lot of room for improvement. First of all, the DPFSP considered in this paper assumes that all factories are identical (the processing times are identical from factory to factory) which is a strong assumption that could be generalized to distinct factories in further studies. Second, the VND methods proposed, albeit very fast and efficient, can be improved with more neighborhoods. At least, one could iterate over VND to construct a VNS or and iterated greedy method as shown, among others by Ruiz and Stützle [28]. Genetic algorithms or more elaborated metaheuristics are surely to perform better than our proposed VND. It is also interesting to study the particular properties that the DPFSP might have, and to develop solution methods accordingly. Last but not least, this research is also applicable to the distributed extensions of other scheduling problems. Examples are other flowshop problem variants [53], parallel machine problems [54] or even hybrid flowshop problems [55]. We hope that this paper sparks a new an interesting venue of research in distributed finite capacity scheduling.

Acknowledgments

The authors would like to express their most sincere thanks to the two anonymous refereees who have helped in improving an earlier version of this paper.

Rubén Ruiz is partially funded by the Spanish Ministry of Science and Innovation, under the project “SMPA-Advanced Parallel Multiobjective Sequencing: Practical and Theoretical Advances” with reference DPI2008-03511/DPI.

References

[1] Johnson SM. Optimal two- and three-stage production schedules with setup times included. Naval Research Logistics Quarterly 1954;1(1):61–8.

[2] Baker KR. Introduction to sequencing and scheduling. New York: Wiley; 1974.

[3] Graham RL, Lawler EL, Lenstra JK, Rinnooy Kan AHG. Optimization and approximation in deterministic sequencing and scheduling: a survey. Annals of Discrete Mathematics 1979;5:287–326.

[4] Garey MR, Johnson DS, Sethi R. The complexity of flowshop and jobshop scheduling. Mathematics of Operations Research 1976;1(2):117–29.

[5] Framinan JM, Gupta JND, Leisten R. A review and classification of heuristics for permutation flow-shop scheduling with makespan objective. Journal of the Operational Research Society 2004;55(12):1243–55.

[6] Hejazi SR, Saghafian S. Flowshop-scheduling problems with makespan criterion: a review. International Journal of Production Research 2005;43(14): 2895–929.

[7] Ruiz R, Maroto C. A comprehensive review and evaluation of permutation flowshop heuristics. European Journal of Operational Research 2005;165(2): 479–94.

[8] Gupta JND, Stafford EF. Flowshop scheduling research after five decades. European Journal of Operational Research 2006;169(3):699–711.

[9] Moon C, Kim J, Hur S. Integrated process planning and scheduling with minimizing total tardiness in multi-plants supply chain. Computers & Industrial Engineering 2002;43(1–2):331–49.

[10] Wang B. Integrated product, process and enterprise design. 1st ed., London: Chapman & Hall; 1997.

[11] Kahn KB, Castellion GA, Griffin A. The PDMA handbook of new product development. 2nd ed., New York: Wiley; 2004.

[12] Hansen P, Mladenovic N. Variable neighborhood search: principles and applications. European Journal of Operational Research 2001;130(3):449–67.

[13] Palmer DS. Sequencing jobs through a multi-stage process in the minimum total time: a quick method of obtaining a near optimum. Operational Research Quarterly 1965;16(1):101–7.

[14] Campbell HG, Dudek RA, Smith ML. Heuristic algorithm for *N*-job, *M*-machine sequencing problem. Management Science Series B—Application 1970;16(10):B630–7.

[15] Nawaz M, Enscore Jr. EE, Ham I. A heuristic algorithm for the *m* machine, *n* job flowshop sequencing problem. Omega—International Journal of Management Science 1983;11(1):91–5.

[16] Framinan JM, Leisten R, Rajendran C. Different initial sequences for the heuristic of Nawaz, Enscore and Ham to minimize makespan, idletime or flowtime in the static permutation flowshop sequencing problem. International Journal of Production Research 2003;41(1):121–48.

[17] Dong XY, Huang HK, Chen P. An improved NEH-based heuristic for the permutation flowshop problem. Computers & Operations Research 2008;35(12):3962–8.

[18] Rad SF, Ruiz R, Boroojerdian N. New high performing heuristics for minimizing makespan in permutation flowshops. Omega—International Journal of Management Science 2009;37(2):331–45.

[19] Kalczynski PJ, Kamburowski J. An empirical analysis of the optimality rate of flow shop heuristics. European Journal of Operational Research 2009;198(1): 93–101.

[20] Nowicki E, Smutnicki C. A fast tabu search algorithm for the permutation flow-shop problem. European Journal of Operational Research 1996;91(1): 160–75.

[21] Grabowski J, Wodecki M. A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion. Computers & Operations Research 2004;31(11):1891–909.

[22] Osman IH, Potts CN. Simulated annealing for permutation flowshop scheduling. Omega—International Journal of Management Science 1989;17(6):551–7.

[23] Reeves CR. A genetic algorithm for flowshop sequencing. Computers & Operations Research 1995;22(1):5–13.

[24] Ruiz R, Maroto C, Alcaraz J. Two new robust genetic algorithms for the flowshop scheduling problem. Omega—International Journal of Management Science 2006;34(5):461–76.

[25] Rajendran C, Ziegler H. Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. European Journal of Operational Research 2004;155(2):426–38.

[26] Onwubolu GC, Davendra D. Scheduling flow shops using differential evolution algorithm. European Journal of Operational Research 2006;171(2):674–92.

[27] Tasgetiren MF, Liang YC, Sevkli M, Gencyilmaz G. A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. European Journal of Operational Research 2007;177(3):1930–47.

[28] Ruiz R, Stützle T. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. European Journal of Operational Research 2007;177(3):2033–49.

[29] Vallada E, Ruiz R. Cooperative metaheuristics for the permutation flowshop scheduling problem. European Journal of Operational Research 2009;193(2): 365–76.

[30] Vallada E, Ruiz R, Minella G. Minimising total tardiness in the *m*-machine flowshop problem: a review and evaluation of heuristics and metaheuristics. Computers & Operations Research 2008;35(4):1350–73.

[31] Minella G, Ruiz R, Ciavotta M. A review and evaluation of multiobjective algorithms for the flowshop scheduling problem. INFORMS Journal on Computing 2008;20(3):451–71.

[32] Jia HZ, Fuh JYH, Nee AYC, Zhang YF. Web-based multi-functional scheduling system for a distributed manufacturing environment. Concurrent Engineering—Research and Applications 2002;10(1):27–39.

[33] Jia HZ, Nee AYC, Fuh JYH, Zhang YF. A modified genetic algorithm for distributed scheduling problems. Journal of Intelligent Manufacturing 2003;14(3–4): 351–62.

[34] Jia HZ, Fuh JYH, Nee AYC, Zhang YF. Integration of genetic algorithm and Gantt chart for job shop scheduling in distributed manufacturing systems. Computers & Industrial Engineering 2007;53(2):313–20.

[35] Chan FTS, Chung SH, Chan PLY. An adaptive genetic algorithm with dominated genes for distributed scheduling problems. Expert Systems with Applications 2005;29(2):364–71.

[36] Chan FTS, Chung SH, Chan LY, Finke G, Tiwari MK. Solving distributed FMS scheduling problems subject to maintenance: genetic algorithms approach. Robotics and Computer-Integrated Manufacturing 2006;22(5–6):493–504.

[37] Wang L, Shen W. Process planning and scheduling for distributed manufacturing. London: Springer; 2007.

[38] Wagner HM. An integer linear-programming model for machine scheduling. Naval Research Logistics Quarterly 1959;6(2):131–40.

[39] Manne AS. On the job-shop scheduling problem. Operations Research 1960;8(2):219–23.

[40] Stafford EF. On the development of a mixed-integer linear-programming model for the flowshop sequencing problem. Journal of the Operational Research Society 1988;39(12):1163–74.

[41] Wilson JM. Alternative formulations of a flowshop scheduling problem. Journal of the Operational Research Society 1989;40(4):395–9.

[42] Pan CH. A study of integer programming formulations for scheduling problems. International Journal of Systems Science 1997;28(1):33–41.

[43] Tseng FT, Stafford EF, Gupta JND. An empirical analysis of integer programming formulations for the permutation flowshop. Omega—International Journal of Management Science 2004;32(4):285–93.

[44] Stafford EF, Tseng FT, Gupta JND. Comparative evaluation of MILP flowshop models. Journal of the Operational Research Society 2005;56(1):88–101.

[45] Tseng FT, Stafford EF. New MILP models for the permutation flowshop problem. Journal of the Operational Research Society 2008;59(10):1373–86.

[46] Taillard E. Some efficient heuristic methods for the flow-shop sequencing problem. European Journal of Operational Research 1990;47(1):65–74.

[47] Mladenovic N, Hansen P. Variable neighborhood search. Computers & Operations Research 1997;24(11):1097–100.

[48] Stützle T. Applying iterated local search to the permutation flow shop problem. AIDA-98-04, FG Intellektik, TU Darmstadt; 1998.

[49] Taillard E. Benchmarks for basic scheduling problems. European Journal of Operational Research 1993;64(2):278–85.

[50] Morgan JN, Sonquist JA. Problems in analysis of survey data, and a proposal. Journal of the American Statistical Association 1963;58(302):415–34.

[51] Biggs D, De Ville B, Suen E. A method of choosing multiway partitions for classification and decision trees. Journal of Applied Statistics 1991;18(1):49–62.

[52] Ruiz R, Serifoglu FS, Urlings T. Modeling realistic hybrid flexible flowshop scheduling problems. Computers & Operations Research 2008;35(4):1151–75.

[53] Pan QK, Wang L. A novel differential evolution algorithm for no-idle permutation flow-shop scheduling problems. European Journal of Industrial Engineering 2008;2(3):279–97.

[54] Pessan C, Bouquard J-L, Néron E. An unrelated parallel machines model for an industrial production resetting problem. European Journal of Industrial Engineering 2008;2(2):153–71.

[55] Hmida AB, Huguet MJ, Lopez P, Haouari M. Climbing depth-bounded discrepancy search for solving hybrid flow shop problems. European Journal of Industrial Engineering 2007;1(2):223–40.