The distributed permutation flow shop to minimise the total flowtime*

Victor Fernandez-Viagas[1]†, Paz Perez-Gonzalez[1], Jose M. Framinan[1]

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

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

June 7, 2018

Abstract

In the last years, researchers are paying special attention to scheduling in distributed environments due to the increasing benefits of multi-factory manufacture. In this paper, we address the distributed permutation flowshop scheduling problem to minimise the total flowtime. Since, to the best of our knowledge, this problem has not been addressed previously, we first analyse it and discuss several properties, theorems, assignment rules, representation of the solutions and speed-up procedures. Given that the problem is NP-hard, we focus on approximate procedures, and propose eighteen constructive heuristics to obtain high-quality solutions in reasonable CPU times. In addition, we propose an iterative improvement algorithm to further refine the so-obtained solutions. The extensive computational experience carried out shows that the proposed method outperforms several metaheuristics adapted from related scheduling problems.

Keywords: Scheduling, distributed, Flowshop, Heuristics, PFSP, total completion time, flowtime, evolutionary, permutation, NEH, genetic algorithm

*Preprint submitted to Computers & Industrial Engineering. https://doi.org/10.1016/j.cie.2018.03.014

†Corresponding author. Email: vfernandezviagas@us.es

1 Introduction

The Permutation Flow shop Scheduling Problem (PFSP) is one of the most studied optimisation problems in the Operations Research literature (see e.g. Fernandez-Viagas et al., 2017 for a recent review). In this problem, n jobs must be processed following the same route of m machines, where job passing is not allowed. In the PFSP all machines are placed in the same factory, i.e. following a mono-factory environment. However, the number of companies using a single factory environment is decreasing in the real-world (Moon et al., 2002), as they are turning into employing a distributed environment in order to reduce manufacturing and delivery costs, and risks, among other reasons (Kahn et al., 2004). As a consequence, research on distributed scheduling problems is growing in the last years (see e.g. Lin and Ying, 2016; Lin and Zhang, 2016; Deng et al., 2017; Wang et al., 2016; Ying et al., 2017). A specific distributed environment is the Distributed Permutation Flowshop Scheduling Problem, denoted as DPFSP (Naderi and Ruiz, 2010). In DPFSP, there are F identical factories where n jobs can be processed, being each factory a permutation flow shop with m machines. The decision problem consists on assigning the jobs to the factories and scheduling them according to a given performance measure.

Up to now, the DPFSP has been addressed almost exclusively to minimise the maximum completion time, or makespan. Nevertheless, the total flowtime (or equivalently the mean flowtime or total completion time if the release times of the jobs are zero) has been pointed out as a more relevant and meaningful objective for today's dynamic manufacturing environment (Liu and Reeves, 2001). Among other goals, the minimisation of the total flowtime aims to minimise the turnaround of jobs, to stabilise the use of resources, and to reduce the in-process inventory (Rajendran and Ziegler, 1997). These advantages are even higher in distributed environments where the imbalances among resources and work in progress may increase due to the existence of several factories. In this paper we address the DPFSP to minimise the total flowtime, denoted as $DF|prmu|\sum C_j$ in the usual notation for scheduling problems (see Graham et al., 1979; Naderi and Ruiz, 2010).

Since the problem under study is a generalisation of the PFSP for total completion time, $Fm|prmu|\sum C_j$, which is known to be strongly NP-hard for $m \geq 2$ (Garey et al., 1976), the

$DF|prmu|\sum C_j$ problem is also strongly NP-hard for $m \geq 2$. Therefore, we focus on obtaining approximate solutions for the problem in reasonable computation times. More specifically, we aim to make a fourfold contribution to the problem. Firstly, we present several properties of the problem in order to reduce the computational effort associated to exploring different types of neighborhood structures. Secondly, we propose two representations of the solutions and several assignment rules to perform the construction of the solutions, and develop simple speed up procedures for the solution representations. Thirdly, we propose eighteen constructive heuristics to efficiently solve the problem using the representations and properties described before, and by adapting existing heuristics from related problems. And fourthly, we propose a simple evolutionary algorithm to improve the obtained solutions from the best proposed constructive heuristic.

To present these contributions, the paper is structured as follows: in Section 2, we review the existing literature on related scheduling problems, and particularly the contributions on the DPFSP for different criteria and/or constraints. In Sections 3 and 4, we perform a detailed analysis of the problem under study. More specifically, we define the problem and present several properties in Section 3. Furthermore, we propose two different representations of the solutions in Subsection 4.1; several assignment rules in Subsection 4.2; and two speed up mechanisms in Subsection 4.3. In Section 5, the constructive heuristics are presented. A simple evolutionary algorithm is presented in Section 6. The computational experiments are carried out in Section 7, and the conclusions are discussed in Section 8.

2 Related literature review

To the best of our knowledge, the $DF|prmu|\sum C_j$ problem has not been previously studied in the literature. In this section we therefore review related contributions in order to analyse how similar problems have been addressed. The corresponding problem for the makespan objective – denoted as $DF|prmu|C_{max}$– was first proposed by Naderi and Ruiz (2010). Nevertheless, without considering the distributed environment, the DPFSP to minimise makespan is mathematically identical to a scheduling problem with several flow shops in parallel on a unique factory, a problem

first proposed by David et al. (1996). This last problem is traditionally denoted as Parallel Flowline (see e.g. Vairaktarakis and Elhafsi, 2000) or Parallel Flow Shop Scheduling Problem (see e.g. Ribas et al., 2017). Under these names, this problem has been mainly addressed only for two-stage flowshops (see e.g. Cao and Chen, 2003; Zhang and Van De Velde, 2012). Regarding the distributed environment, Naderi and Ruiz (2010) propose and compare six different Mixed Integer Linear Programming (MILP) models as well as four different constructive heuristics, denoted as NEH1, NEH2, VND(a) and VND(b), based on two different rules to assign the jobs to the factories. NEH1 and NEH2 are variations of the NEH, which is a core heuristic for many scheduling problems and was originally proposed for the PFSP to minimise makespan by Nawaz et al. (1983).

Several contributions propose iterative approximate algorithms for the $DF|prmu|C_{max}$ problem: Xu et al. (2010) present a Hybrid Immune Algorithm. Gao and Chen (2011) propose a Genetic Algorithm with insertion and interchange local search procedures, which is later outperformed by the Tabu Search by Gao et al. (2013). This latter algorithm improves the solutions by iteratively exchanging jobs between two different factories by applying different methods of interchange and insertion of jobs. Wang et al. (2013) propose an Estimation of Distribution Algorithm for the problem and implement a probability model to generate the new individuals of each iteration. Lin et al. (2013) propose an iterated-greedy-based algorithm, MIG in the following. The aforementioned proposals by Gao and Chen (2011), Gao et al. (2013), and Lin et al. (2013) have been outperformed by the Scatter Search, SS, proposed by Naderi and Ruiz (2014), comparing the algorithms under the same conditions. Fernandez-Viagas and Framinan (2015a) propose a Bounded-Search Iterated Greedy algorithm, BSIG, which also outperforms the proposals by Gao and Chen (2011), Gao et al. (2013), Lin et al. (2013), and Wang et al. (2013) under the same conditions. Since SS and BSIG were developed at the same time, there is no direct comparison between both proposals, so both stay as the state-of-the-art metaheuristics for the problem.

Regarding different objective functions, we are not aware of previous contributions with the exception of Deng et al. (2017). They address the multi-objective DPFSP with makespan and total tardiness criteria, using a similar representation of the solutions as in Naderi and Ruiz (2010),

by means of a Memetic Algorithm. The performance of their algorithm is tested by comparing it with a random algorithm and with the multi-objective optimization algorithm proposed by Deb et al. (2002). Finally, regarding the distributed layout under different constraints, recently Lin and Ying (2016) address the DPFSP for no-wait constraints and makespan minimisation, i.e. $DF|prmu, nwt|C_{max}$. They propose both a MILP and an iterated-greedy-based algorithm. Finally, Ribas et al. (2017) address the DPFSP with blocking constraints and makespan minimisation, i.e. $DF|blocking|C_{max}$. They propose a total of 13 constructive heuristics and two iterative improvement algorithms, which are compared in an extensive computational evaluation.

3 Problem statement

3.1 Problem description and notation

The problem under study can be defined as follows: There is a set $\mathcal{N}$ of n jobs to be processed in a set $\mathcal{F}$ of F factories. Each job is processed in a single factory, and all factories can process all jobs. The manufacturing layout in each factory is a flowshop of m machines. All factories are assumed to be identical. The processing time of job j on machine i in factory f is denoted by $p_{ijf} = p_{ij}$. Given a sequence Π_f of jobs assigned to factory f, let $C_{ij}(\Pi_f)$ (or C_{ij}, whenever it does not lead to confusion) be the completion time of job $j \in \Pi_f$ on machine i. Let C_j be the completion time of job $j \in \Pi_f$ on the last machine, i.e. $C_j = C_{mj}$, or simply the completion time of job j. Analogously, $[k]$ denotes the job in position k in Π_f.

Definition 3.1. *A semi-active schedule for DPFSP is given by an assignment of jobs to each factory and by a sequence Π_f within each factory f.*

Note that the results presented in this section consider only semi-active schedules (see e.g. Pinedo, 1995 for a definition of this class of schedules). The goal of the problem is to find the sequence that minimises the total completion time of the jobs, i.e. the minimisation of $\sum_{j\in\mathcal{N}} C_j$ (also denoted by C_{sum}). We denote $\sum_{j\in\Pi_f} C_j$, or equivalently $C_{sum,f}(\Pi_f)$, the total completion

time of jobs in factory f according to Π_f. Then,

$$C_{sum} = \sum_{j \in \mathcal{N}} C_j = \sum_{f \in \mathcal{F}} \sum_{j \in \Pi_f} C_j = \sum_{f \in \mathcal{F}} C_{sum,f}(\Pi_f)$$

Additionally, we denote $C_{max,f}$ the makespan (maximum completion time) on factory f.

3.2 Problem properties

The aforementioned good results found by Fernandez-Viagas and Framinan (2015a) for the $DF|prmu|C_{max}$ problem prove that discovering properties of the solutions can substantially improve the behaviour of the algorithms. Similar results have also been found for other scheduling problems in several studies (see e.g. Rios-Mercado and Bard, 1998; Fernandez-Viagas and Framinan, 2015b). In this paper, we also try to further understand the structure of solutions in the problem under consideration with the aim of reducing the computational effort of the proposed approaches.

Given a partial sequence $\Pi_f := (\pi_1, \ldots, \pi_{n_f})$ of n_f jobs assigned to factory f, let us denote by $\Pi_f^{\sigma k}$ the sequence resulting from inserting job σ in position k of Π_f, i.e. $\Pi_f^{\sigma k} := (\pi_1, \ldots, \pi_{k-1}, \sigma, \pi_k, \ldots, \pi_{n_f})$.

Lemma 3.1. *The completion time of each job in position $l \geq k$ of $\Pi_f^{\sigma k}$ increases at least $\min_{1 \leq i \leq m} p_{i\sigma}$, i.e. it verifies*

$$C_{[l]}(\Pi_f^{\sigma k}) \geq C_{[l-1]}(\Pi_f) + \min_{1 \leq i \leq m} p_{i\sigma}$$

Proof. As we consider semi-active schedules, there must be at least a machine without idle time between jobs π_{k-1} and π_k. So, the completion time of each job in position $l > k$ increases at least the minimal processing time of σ among these machines, which is necessarily greater than or equal to:

$$\min_{1 \leq i \leq m} p_{i\sigma}$$

□

Next, we propose three different theorems (Theorems 3.1, 3.2, and 3.3) to establish lower bounds on the total completion time due to the assignment of a job to a partial sequence. Each theorem proposes a different lower bound with a different complexity. As we will see in the following sections, the idea behind these theorems is that, when constructing solutions, a job is not inserted in a factory where the lower bound of the resulting total completion time is higher than the actual upper bound.

Theorem 3.1. *LB1. A lower bound of the total completion time of* $\Pi_f^{\sigma k}$ *(*$1 \leq k \leq n_f + 1$*) is given by*

$$LB1 = \sum_{j \in \Pi_f} C_j + \sum_{i=1}^{m} p_{i\sigma} + \min \left\{ \min_{1 \leq i \leq m} p_{i\sigma}; \sum_{j \in \Pi_f} p_{1j} \right\} \tag{1}$$

Proof. Note that, for the partial sequence Π_f, the completion time in the first machine of the last job π_{n_f} is $C_{1[n_f]}(\Pi_f) = \sum_{j \in \Pi_f} p_{1j}$. Additionally, $\sum_{i=1}^{m} p_{i\sigma}$ is the completion time of job σ when inserted in the first position, so it is a lower bound when it is inserted in any other position. As a consequence, we have the following lower bounds for the total completion time $\Pi_f^{\sigma k}$.

Case $k = n_f + 1$ (insertion in the last position): $\Pi_f^{\sigma, n_f+1} = (\pi_1, \ldots, \pi_{n_f}, \sigma)$

In this case

$$\sum_{j \in \Pi_f^{\sigma, n_f+1}} C_j(\Pi_f^{\sigma, n_f+1}) \geq \sum_{j \in \Pi_f} C_j + C_{1[n_f]}(\Pi_f) + \sum_{i=1}^{m} p_{i\sigma} = \sum_{j \in \Pi_f} C_j + \sum_{j \in \Pi_f} p_{1j} + \sum_{i=1}^{m} p_{i\sigma}$$

Case $k < n_f + 1$ (insertion in the rest of positions): $\Pi_f^{\sigma k} = (\pi_1, \ldots, \pi_{k-1}, \sigma, \pi_k, \ldots, \pi_{n_f})$

In this case the completion time of each job sequenced after σ increases at least $\min_{1 \leq i \leq m} p_{i\sigma}$ according to Lemma 3.1. Therefore,

$$\sum_{j \in \Pi_f^{\sigma k}} C_j(\Pi_f^{\sigma k}) \geq \sum_{j \in \Pi_f} C_j + C_{1[k-1]}(\Pi_f) + \sum_{i=1}^{m} p_{i\sigma} + (n_f + 1 - k) \min_{1 \leq i \leq m} p_{i\sigma} \geq \sum_{j \in \Pi_f} C_j + \sum_{i=1}^{m} p_{i\sigma} + \min_{1 \leq i \leq m} p_{i\sigma}$$

□

The computation of $LB1$ in an iterative procedure has a complexity order of $O(1)$, as $\sum_{i=1}^{m} p_{i\sigma}$ and $\min_{1 \leq i \leq m}\{p_{i\sigma}\}$ can be computed in advance from the instance data (e.g. at the beginning of the procedure), and $\sum_{j \in \Pi_f} C_j$ and $\sum_{j \in \Pi_f} p_{1j}$ are values known from the existing sequence Π_f.

Theorem 3.2. *LB2. A lower bound of the total completion time of* $\Pi_f^{\sigma k}$ *(*$1 \leq k \leq n_f + 1$*) is given by*

$$LB2 = \sum_{j \in \Pi_f} C_j + \min \left\{ \sum_{i=1}^{m} p_{i\sigma} + n_f \min_{1 \leq i \leq m} \{p_{i\sigma}\}; \min_{2 \leq k \leq n_f+1} \left\{ \max\{\sum_{l=1}^{k-1} p_{1[l]}(\Pi_f) + \sum_{i=1}^{m} p_{i\sigma}; C_{[k-1]}(\Pi_f) + p_{m\sigma}\} + (n_f - k + 1) \min_{1 \leq i \leq m} p_{i\sigma} \right\} \right\} \tag{2}$$

Proof. Case $k = 1$ (insertion in the first position): $\Pi_f^{\sigma 1} := (\sigma, \pi_1, \ldots, \pi_{n_f})$

In this case

$$C_\sigma(\Pi_f^{\sigma 1}) = \sum_{i=1}^{m} p_{i\sigma}$$

and using Lemma 3.1 we have that $C_{[l]}(\Pi_f^{\sigma 1}) \geq C_{[l-1]}(\Pi_f) + \min_{1 \leq i \leq m} p_{i\sigma}$, $\forall l > 1$. Therefore,

$$\sum_{j \in \Pi_f^{\sigma 1}} C_j(\Pi_f^{\sigma 1}) \geq \sum_{j \in \Pi_f} C_j + \sum_{i=1}^{m} p_{i\sigma} + n_f \min_{1 \leq i \leq m} p_{i\sigma}$$

Case $k > 1$ (insertion in the rest of positions): $\Pi_f^{\sigma k} = (\pi_1, \ldots, \pi_{k-1}, \sigma, \pi_k, \ldots, \pi_{n_f}), k > 1$

First, we take into account that the completion times of the jobs before position k is the same for Π_f than $\Pi_f^{\sigma k}$, i.e. $\forall l < k$ $C_{[l]}(\Pi_f) = C_{[l]}(\Pi_f^{\sigma k})$.

Second, the completion time of the inserted job l (in position k) verifies

$$C_\sigma(\Pi_f^{\sigma k}) \geq C_{1[k-1]}(\Pi_f) + \sum_{i=1}^{m} p_{i\sigma} = \sum_{l=1}^{k-1} p_{1[l]}(\Pi_f) + \sum_{i=1}^{m} p_{i\sigma}$$

Additionally,

$$C_\sigma(\Pi_f^{\sigma k}) \geq C_{[k-1]}(\Pi_f) + p_{m\sigma}$$

Therefore,

$$C_\sigma(\Pi_f^{\sigma k}) \geq \max\{\sum_{l=1}^{k-1} p_{1[l]}(\Pi_f) + \sum_{i=1}^{m} p_{i\sigma}; C_{[k-1]}(\Pi_f) + p_{m\sigma}\}$$

Third, the completion times of jobs after σ (in total $n_f - k + 1$ jobs) are increased at least by $\min_{1 \leq i \leq m} p_{i\sigma}$ according to Lemma 3.1. Then, for $l > k$, $C_{[l]}(\Pi_f^{\sigma k}) \geq C_{[l-1]}(\Pi_f) + \min_{1 \leq i \leq m} p_{i\sigma}$. As consequence,

$$\sum_{j\in\Pi_f^{\sigma k}} C_j(\Pi_f^{\sigma k}) \geq \sum_{j\in\Pi_f} C_j + \max\{\sum_{l=1}^{k-1} p_{1[l]}(\Pi_f) + \sum_{i=1}^{m} p_{i\sigma}; C_{[k-1]}(\Pi_f) + p_{m\sigma}\} + (n_f + 1 - k) \min_{1\leq i\leq m} p_{i\sigma} \geq$$

$$\geq \min_{2\leq k\leq n_f+1} \left\{ \sum_{j\in\Pi_f} C_j + \max\{\sum_{l=1}^{k-1} p_{1[l]}(\Pi_f) + \sum_{i=1}^{m} p_{i\sigma}; C_{[k-1]}(\Pi_f) + p_{m\sigma}\} + (n_f + 1 - k) \min_{1\leq i\leq m} p_{i\sigma} \right\} =$$

$$= \sum_{j\in\Pi_f} C_j + \min_{2\leq k\leq n_f+1} \left\{ \max\{\sum_{l=1}^{k-1} p_{1[l]}(\Pi_f) + \sum_{i=1}^{m} p_{i\sigma}; C_{[k-1]}(\Pi_f) + p_{m\sigma}\} + (n_f + 1 - k) \min_{1\leq i\leq m} p_{i\sigma} \right\}$$

□

The complexity of this lower bound is $O(n)$, which is determined by the computation of the second term in the minimum. Although its complexity is higher than $LB1$, its value is tighter –i.e. $LB1 \leq LB2$– as it is shown in the next corollary. So there is a trade-off between complexity and quality between both lower bounds.

Corollary 3.1. *Let $LB1$ be the lower bound of Theorem 3.1, and $LB2$ the lower bound of Theorem 3.2. Then, $LB1 \leq LB2$.*

Proof. From Equation 1, we have that

$$LB1 \leq \sum_{j\in\Pi_f} C_j + \sum_{i=1}^{m} p_{i\sigma} + \min_{1\leq i\leq m} p_{i\sigma}$$

Let A be the right hand of the given inequality.

From Equation 2, we have that

$$LB2 \geq \sum_{j\in\Pi_f} C_j + \min \left\{ \sum_{i=1}^{m} p_{i\sigma} + n_f \min_{1\leq i\leq m} p_{i\sigma}; \sum_{i=1}^{m} p_{i\sigma} + \min_{2\leq k\leq n_f+1} \{\sum_{l=1}^{k-1} p_{1[l]}(\Pi_f) + (n_f - k + 1) \min_{1\leq i\leq m} p_{i\sigma}\} \right\} =$$

$$= \sum_{j\in\Pi_f} C_j + \sum_{i=1}^{m} p_{i\sigma} + \min\{n_f \min_{1\leq i\leq m} p_{i\sigma}; \min_{2\leq k\leq n_f+1} \{\sum_{l=1}^{k-1} p_{1[l]}(\Pi_f) + (n_f - k + 1) \min_{1\leq i\leq m} p_{i\sigma}\}$$

Let B be the right hand of the given inequality.

We have that

$$LB1 \leq A \leq B \leq LB2 \Leftrightarrow A \leq B \Leftrightarrow$$

$$\Leftrightarrow \min_{1\leq i\leq m} p_{i\sigma} \leq \min\{n_f \min_{1\leq i\leq m} p_{i\sigma}; \min_{2\leq k\leq n_f+1}\{\sum_{l=1}^{k-1} p_{1[l]}(\Pi_f) + (n_f-k+1)\min_{1\leq i\leq m} p_{i\sigma}\} \Leftrightarrow$$

$$\Leftrightarrow \begin{cases} \min_{1\leq i\leq m} p_{i\sigma} \leq n_f \min_{1\leq i\leq m} p_{i\sigma} \\ \min_{1\leq i\leq m} p_{i\sigma} \leq \min_{2\leq k\leq n_f+1}\{\sum_{l=1}^{k-1} p_{1[l]}(\Pi_f) + (n_f-k+1)\min_{1\leq i\leq m} p_{i\sigma}\} \end{cases}$$

Both inequalities are trivial for $n_f > 0$.

□

Theorem 3.3. *LB3. A lower bound of the total completion time of $\Pi_f^{\sigma k}(1 \leq k \leq n_f+1)$ is given by*

$$LB3 = \min\{LB3_1; LB3_2; LB3_3\} \tag{3}$$

with $LB3_1$, $LB3_2$ and $LB3_3$ given by the following expresions:

$$LB3_1 = (n_f+1)\sum_{i=1}^{m} p_{i\sigma} + \sum_{l=1}^{n_f}(n_f+1-l)\min_{1\leq i\leq m} p_{i[l]}(\Pi_f) \tag{4}$$

$$LB3_2 = \min_{2\leq k\leq n_f}\{\sum_{l=1}^{k-1} C_{[l]}(\Pi_f) + (n_f-k+2)C_\sigma(\Pi_f^{\sigma k}) + \sum_{l=k}^{n_f}(n_f+1-l)\min_{1\leq i\leq m} p_{i[l]}(\Pi_f)\} \tag{5}$$

$$LB3_3 = \sum_{j\in\Pi_f} C_j(\Pi_f) + C_\sigma(\Pi_f^{\sigma,n_f+1}) \tag{6}$$

Proof. Case $k=1$ (insertion in the first position): $\Pi_f^{\sigma 1} = (\sigma, \pi_1, \ldots, \pi_{n_f}) = (\pi_1^{\sigma 1}, \ldots, \pi_{n_f+1}^{\sigma 1})$

In this case

$$C_\sigma(\Pi_f^{\sigma 1}) = \sum_{i=1}^{m} p_{i\sigma}$$

and taking into account Lemma 3.1, we have that

$$C_{[l]} \geq C_\sigma(\Pi_f^{\sigma 1}) + \sum_{l'=2}^{l} \min_{1\leq i\leq m} p_{i[l']}(\Pi_f^{\sigma 1}), \forall l > k$$

Therefore

$$\sum_{j\in\Pi_f^{\sigma 1}} C_j \geq \sum_{i=1}^{m} p_{i\sigma} + \sum_{2\leq l\leq n_f+1}\left(\sum_{i=1}^{m} p_{i\sigma} + \sum_{l'=2}^{l}\min_{1\leq i\leq m} p_{i[l']}(\Pi_f^{\sigma 1})\right) =$$

$$= (n_f+1)\sum_{i=1}^{m} p_{i\sigma} + \sum_{l=2}^{n_f+1}(n_f-l+2)\min_{1\leq i\leq m} p_{i[l]}(\Pi_f^{\sigma 1}) = (n_f+1)\sum_{i=1}^{m} p_{i\sigma} + \sum_{l=1}^{n_f}(n_f+1-l)\min_{1\leq i\leq m} p_{i[l]}(\Pi_f)$$

Case $1 < k < n_f + 1$ (insertion in the rest of positions, except the last position):

$$\Pi_f^{\sigma k} = (\pi_1, \ldots, \pi_{k-1}, \sigma, \pi_k, \ldots, \pi_{n_f}) = (\pi_1^{\sigma k}, \ldots, \pi_{n_f+1}^{\sigma k})$$

First, and in the same way that the proof of Theorem 3.2, note that the completion time of jobs in positions previous to k are the same for Π_f than $\Pi_f^{\sigma k}$, i.e. $\forall l < k\ C_{[l]}(\Pi_f) = C_{[l]}(\Pi_f^{\sigma k})$.

Second, note that the completion time of job σ can be easily computed since the values of $C_{i[k-1]}$ are known $\forall i$ for the sequence Π_f.

Third, and taking into account Lemma 3.1, we have that the completion times of jobs after σ (in total $n_f - k + 1$ jobs), i.e. for $l > k$, verifies that

$$C_{[l]}(\Pi_f^{\sigma k}) \geq C_\sigma(\Pi_f^{\sigma k}) + \sum_{l'=k+1}^{l} \min_{1\leq i\leq m} p_{i[l']}(\Pi_f^{\sigma k})$$

As a consequence, the total completion time of $\Pi_f^{'}$ verifies that

$$\sum_{j\in\Pi_f^{\sigma k}} C_j \geq \sum_{l=1}^{k-1} C_{[l]}(\Pi_f) + C_\sigma(\Pi_f^{\sigma k}) + \sum_{l=k+1}^{n_f+1}(C_\sigma(\Pi_f^{\sigma k}) + \sum_{l'=k+1}^{l} \min_{1\leq i\leq m} p_{i[l']}(\Pi_f^{\sigma k})) =$$

$$= \sum_{l=1}^{k-1} C_{[l]}(\Pi_f) + (n_f - k + 2)C_\sigma(\Pi_f^{\sigma k}) + \sum_{l=k+1}^{n_f+1}(n_f - l + 2)\min_{1\leq i\leq m} p_{i[l]}(\Pi_f^{\sigma k}) \geq$$

$$\geq \min_{2\leq k\leq n_f}\{\sum_{l=1}^{k-1} C_{[l]}(\Pi_f) + (n_f - k + 2)C_\sigma(\Pi_f^{\sigma k}) \sum_{l=k+1}^{n_f+1}(n_f - l + 2)\min_{1\leq i\leq m} p_{i[l]}(\Pi_f^{\sigma k})\} =$$

$$= \min_{2\leq k\leq n_f}\{\sum_{l=1}^{k-1} C_{[l]}(\Pi_f) + (n_f - k + 2)C_\sigma(\Pi_f^{\sigma k}) + \sum_{l=k}^{n_f}(n_f + 1 - l)\min_{1\leq i\leq m} p_{i[l]}(\Pi_f)\}$$

Case $k = n_f + 1$ (insertion in the last position): $\Pi_f^{\sigma, n_f+1} = (\pi_1, \ldots, \pi_{n_f}, \sigma)$

Obviously,

$$\sum_{j \in \Pi_f^{\sigma,n_f+1}} C_j \geq \sum_{j \in \Pi_f} C_j + C_\sigma(\Pi_f^{\sigma,n_f+1})$$

□

The complexity of $LB3$ is bounded by the calculation of $\min_{2 \leq k \leq n_f} \{\sum_{l=1}^{k-1} C_{[l]}(\Pi_f) + (n_f - k + 2) C_\sigma(\Pi_f^{\sigma k}) + \sum_{l=k}^{n_f} (n_f + 1 - l) \min_{1 \leq i \leq m} p_{i[l]}(\Pi_f)\}$, which is $O(\max\{n^2, nm\})$. Note that this complexity is higher than both previous lower bounds. However, in this case we are not aware of the any relationship between the value of $LB3$ and the other bounds, as the procedure to obtain it is different.

Note that the proposed theorems can be used to reduce the number of solutions explored by the algorithms, as stated in the following corollary (see Section 4.3 for more details).

Corollary 3.2. *Let $C_{sum,f}(\Pi_f)$ be the total completion time in each factory f ($\forall f \in \mathcal{F}$). Let $LB1$, $LB2$, and $LB3$ be the lower bounds when a new job σ is inserted in Factory f' ($\Pi_{f'}^{\sigma k}$), corresponding to Theorems 3.1, 3.2, and 3.3, respectively. In addition, let UB denote the best value of the objective function (C_{sum}) after the insertion of σ in f'' (with $f'' \neq f'$). Then, the value of UB cannot be improved by inserting σ in f' if $UB \leq \sum_{\forall f \neq f'} C_{sum,f}(\Pi_f) + LBi$, $i \in \{1, 2, 3\}$.*

Proof. The proof of the property is obvious using Theorems 3.1, 3.2, and 3.3. □

4 Scheme of solutions

4.1 Representation of the solutions

The representation of the solutions is a key factor of the efficiency of the algorithms. Each representation corresponds to a value of the objective function. The procedure to obtain this value from the representation of the solutions is labelled as decoding. So, both a representation of the solutions and a decoding procedure should be defined in an algorithm. Regarding the representation of the solutions, firstly, it influences the total number of solutions to be computed. Thus, a representation can cover from every feasible schedule to only a small percentage of all

feasible schedules. Secondly, it influences the size of the different neighbourhoods of local search methods and therefore influences their CPU requirements. Thirdly, along with the decoding, it influences the quality of the computed solutions. In this way, a well-designed representation and decoding must try to reach good regions of solutions among the total feasible solutions. The following different representations of the solutions can be employed for our problem:

- $\mathcal{R}_1$: This representation consists of a single sequence of jobs for the system (set of factories). The size of the sequence is n. Note that using this representation the assignment of jobs to factories is not known, i.e. jobs have to be assigned to factories using any of the factory assignment rules discussed in Subsection 4.2 to decode the solution.

- $\mathcal{R}_2$: This representation considers a sequence of job, denoted as $\Pi_f := (\pi_{1,f}, \ldots, \pi_{n_f,f})$ for each factory f, with $f \in \{1, \ldots, F\}$, i,e., a solution is represented by $\Pi := (\Pi_1, \ldots, \Pi_f, \ldots, \Pi_F)$. The size of each sequence Π_f (i.e. the number of jobs assigned to factory f) is n_f, where $n = \sum_{f \in \mathcal{F}} n_f$. Note that each representation of the solutions according to $\mathcal{R}_2$ completely defines a schedule (see Definition 3.1) and then corresponds to a unique objective function when semi-active schedules are considered.

4.2 Factory assignment rules

In this section we propose several factory assignment rules. These rules are procedures to decide the factories where a new job must be assigned. The use of the factory assignment rules depends on the representation of the solutions. On the one hand, using $\mathcal{R}_1$ as explained above, each job is inserted in the last position of the factory determined by the chosen assignment rule. On the other hand, the assignment rules are part of the insertion procedures of the algorithms and are used to determine the factory where a job must be inserted when using $\mathcal{R}_2$. In this case, the assignment rules could be also considered as different types of neighbourhoods with different search spaces. In addition, each assignment rule can use a speed up procedure as explained in Subsection 4.3.

Let us denote by $\Pi^{\sigma,k,f'} := (\Pi_1^{\sigma,k,f'}, \ldots, \Pi_f^{\sigma,k,f'}, \ldots, \Pi_F^{\sigma,k,f'})$ the solution obtained after inserting job σ in $\Pi := (\Pi_1, \ldots, \Pi_f, \ldots, \Pi_F)$, in position k of factory f'. Obviously, $\Pi_f^{\sigma,k,f'} = \Pi_f$,

$\forall f \in \mathcal{F} - \{f^{'}\}$ and therefore $\Pi^{\sigma,k,f^{'}} := (\Pi_1, \ldots, \Pi_f, \ldots, \Pi^{\sigma,k}_{f^{'}}, \ldots, \Pi_F)$. Then, the proposed assignment rules are explained as follows:

1. $\mathcal{A}_1$: Assign job σ to factory f^* with the current lowest makespan according to Π_f (without considering job σ), i.e. $f^* := \arg\min_{f \in \mathcal{F}}\{C_{max,f}\}$. This assignment rule was first proposed for the $DF|prmu|C_{max}$ problem by Naderi and Ruiz (2010).

2. $\mathcal{A}_2$: Assign job σ to factory f^* with the lowest local total flowtime according to Π_f (without considering job σ), $f^* := \arg\min_{f \in \mathcal{F}}\{\sum_{j \in \Pi_f} C_j\}$.

3. $\mathcal{A}_3$: Assign job σ to factory f^* yielding the lowest local total flowtime after inserting job σ in the best position k^*, $(k^*, f^*) := \arg\min_{k,f'}\{\sum_{j \in \Pi^{\sigma,k}_{f'}} C_j\}$.

4. $\mathcal{A}_4$: Assign job σ to factory f^* yielding the lowest global total flowtime after inserting job σ in the best position k^*, $(k^*, f^*) := \arg\min_{k,f'}\{\sum_{f \in \mathcal{F}} \sum_{j \in \Pi^{\sigma,k,f'}_f} C_j\} = \arg\min_{k,f'}\{\sum_{j \in \Pi^{\sigma,k,f'}} C_j\}$.

5. $\mathcal{A}_5$: Assign job σ to factory f^* ($f^* \in \mathcal{F} - \{f_{max}\}$), where $f_{max} := \arg\max_f\{\sum_{j \in \Pi_f} C_j\})$, yielding the lowest global total flowtime after inserting job σ in the best position k^*, $(k^*, f^*) := \arg\min_{k,f' \neq f_{max}}\{\sum_{j \in \Pi^{\sigma,k,f'}} C_j\}$.

6. $\mathcal{A}_6$: Similarly to rule $\mathcal{A}_5$, job σ is tested in a subset $\mathcal{F}^{'}$ of $F/2$ factories, with $\mathcal{F}^{'} \in \mathcal{F}$. Subset $\mathcal{F}^{'}$ is formed by the $F/2$ factories with the lowest total flowtime. Job σ is assigned to factory f^* (among the first $F/2$ factories with the lowest local total flowtime) yielding the lowest global total flowtime after inserting job σ in the best position k^*, $(k^*, f^*) := \arg\min_{k,f' \in \mathcal{F}'}\{\sum_{j \in \Pi^{\sigma,k,f'}} C_j\}$.

4.3 Speed up mechanisms

Since the accelerations proposed by Taillard (1990) for the PFSP to minimise makespan, several speed up mechanisms have been proposed in the literature during the last years in flowshop environments (see e.g. Fernandez-Viagas et al., 2016a,b; Framinan and Leisten, 2008; Li et al., 2009; Naderi and Ruiz, 2010). Similarly to Fernandez-Viagas et al. (2016a), and Li et al. (2009), we propose two simple speed up mechanisms for the representations of the solutions analysed in Subsection 4.1, which are applied in each implemented algorithm in our study:

- Speed up Procedure 1. When using the $\mathcal{R}_1$ representation of the solutions, each job in a solution is placed at the end of the sequence of a factory. This speed up mechanism then stores the completion times in each machine of the last job in each factory, so when a job is inserted in the last position, the completion times of this inserted job are computed with the previous last job of the factory instead of completely evaluating the sequence.

- Speed up Procedure 2. It is used for the $\mathcal{R}_2$ representation of the solutions. Similarly to the previous procedure, this mechanism stores some data of the solution to reduce the complexity in the evaluation of each tested solution. More specifically, this mechanism stores the completion times of each job in each machine of each factory. So, when a job is tested in a position of a factory, the completion times of all previous jobs are known as states unalterable. Only the completion times after this position must be recalculated.

5 Proposed NEH-based constructive heuristics

In the NEH, jobs are first sorted according to a specific criterion (originally the decreasing sums of processing times). Following this order, each job is iteratively inserted in the position of a partial sequence that minimises the objective function. Due to both its excellent performance and its simple adaptations to other scheduling problems, several NEH-based heuristics have been proposed for the $DF|prmu|C_{max}$ problem (Naderi and Ruiz, 2010), and for several other related scheduling problems (see e.g. Vázquez-Rodríguez and Ochoa, 2011; Companys et al., 2010). In fact, some variations remain nowadays as state-of-the-art algorithms for several scheduling problems (Fernandez-Viagas and Framinan, 2015c; Naderi and Ruiz, 2010; Fernandez-Viagas and Framinan, 2015b).

In this paper, we propose eighteen constructive heuristics based on the NEH mechanism to construct the solution. All of them are constructed by combining the representations of the solutions of Subsection 4.1, with the assignment rules of Subsection 4.2 (i.e. each combination uses a different representation of the solutions and/or a different assignment rule), plus six heuristics using the theorems in Section 3.2. The proposed constructive heuristics can be divided as follows: firstly into two sets depending on whether $\mathcal{R}_1$ or $\mathcal{R}_2$ is applied, denoted by $NEH(\mathcal{R}_1, \mathcal{A}_i)$

and $NEH(\mathcal{R}_2, \mathcal{A}_i)$ (with $i \in \{1, \ldots, 6\}$), respectively; and finally, into a set of six heuristics by incorporating Theorems 3.1, 3.2, and 3.3 on the $NEH(\mathcal{R}_2, \mathcal{A}_3)$ and $NEH(\mathcal{R}_2, \mathcal{A}_4)$ heuristics.

For $NEH(\mathcal{R}_1, \mathcal{A}_i)$, a solution is given by a unique sequence using $\mathcal{R}_1$. We first order the jobs in descending order of the sum of their processing times, denoted by $\Omega := (\omega_1, \ldots, \omega_n)$. Let Π be a partial sequence initially formed by only ω_1, $\Pi := (\omega_1)$. Next, iteratively from $k = 2$ to n, job ω_k is tested in every position of partial sequence Π. To compute the total completion time, the sequence is decoded using the $\mathcal{A}_i$ assignment rule (each job in the sequence is assigned to a factory). For each iteration, ω_k is then inserted in the position that minimises the total completion time and Π is updated accordingly. The pseudocode of the $NEH(\mathcal{R}_1, \mathcal{A}_i)$ heuristic is shown in Figure 1.

Procedure $NEH(\mathcal{R}_1, \mathcal{A}_i)$

  //Initial Order

  $\Omega :=$ Jobs sorted in non-increasing sums of processing times ($\Omega = \{\omega_1, \ldots, \omega_n\}$);

  $\Pi := (\omega_1)$;

  for $k = 2$ **to** n **do**

    //Construction of partial sequences

    $\Pi^{\omega_k l}$:= Partial sequence formed by inserting job ω_k in position l of partial sequence Π, $\forall l \in \{1, \ldots, k\}$;

    //Total completion time evaluation

    Compute each partial sequence $\Pi^{\omega_k l}$ by using assignment rule $\mathcal{A}_i$ to allocate jobs to factories at the end of that factory. Let l^* be the index l whose total completion time is minimal, i.e. $l^* := \arg\min_{\forall l}\{C_{sum}(\Pi^{\omega_k l}\}$;

    //Sequence selection

    $\Pi := \Pi^{\omega_k l^*}$;

  end

end

Figure 1: $NEH(\mathcal{R}_1, \mathcal{A}_i)$

Regarding $NEH(\mathcal{R}_2, \mathcal{A}_i)$, a sequence for each factory is used to construct the final sequence. More specifically, jobs are again sorted in non-increasing sum of processing times. The first job,

ω_1, is inserted in the partial sequence of the first factory, i.e. $\Pi_1 = (\omega_1)$. The other partial sequences are initially empty, i.e. $\Pi_f = \varnothing$, $\forall f \neq 1$. Then, each job ω_k $(2 \leq k \leq n)$ is inserted in partial sequence Π_f on position l^* according to assignment rule $\mathcal{A}_i$.

In addition, six heuristics are proposed by applying the proposed theorems of Section 3.2 over the $NEH(\mathcal{R}_2, \mathcal{A}_3)$ and $NEH(\mathcal{R}_2, \mathcal{A}_4)$ heuristics. Let $NEH(\mathcal{R}_2, \mathcal{A}_3^{th1})$, $NEH(\mathcal{R}_2, \mathcal{A}_3^{th2})$, and $NEH(\mathcal{R}_2, \mathcal{A}_3^{th3})$ be the heuristics resulting of applying Theorems 3.1, 3.2, and 3.3, respectively, on $NEH(\mathcal{R}_2, \mathcal{A}_3)$. Let $NEH(\mathcal{R}_2, \mathcal{A}_4^{th1})$, $NEH(\mathcal{R}_2, \mathcal{A}_4^{th2})$, and $NEH(\mathcal{R}_2, \mathcal{A}_4^{th3})$ be the other three heuristics. Thus, when a job is tested in every position of each factory during the assignment rule, a factory is discarded if its lower bound (according to the corresponding theorem) is higher than the minimal completion time found in the iteration. Otherwise, this job is tested in that factory. The pseudocodes of the $NEH(\mathcal{R}_2, \mathcal{A}_3^{th1})$ and $NEH(\mathcal{R}_2, \mathcal{A}_4^{th1})$ algorithms are shown in Figures 3 and 4, being the pseudocodes of the other heuristics analogous.

Procedure $NEH(\mathcal{R}_2, \mathcal{A}_i)$
  //Initial Order
  $\Omega :=$ Jobs sorted in non-increasing sums of processing times ($\Omega = \{\omega_1, \ldots, \omega_n\}$);
  $\Pi_1 := (\omega_1)$;
  $\Pi_f = \varnothing$, $\forall f \neq 1$;
  for $k = 2$ **to** n **do**
    //Assignment rule
    $(l, f) := \mathcal{A}_i$;
    //Construction of partial sequences
    $\Pi_f :=$ Sequence formed by inserting job ω_k in position l of partial sequence Π_f;
  end
end

Figure 2: $NEH(\mathcal{R}_2, \mathcal{A}_i)$

Procedure $NEH(\mathcal{R}_2, \mathcal{A}_3^{th1})$

//Initial Order

$\Omega :=$ Jobs sorted in non-increasing sums of processing times ($\Omega = \{\omega_1, \ldots, \omega_n\}$);

$\Pi_1 := (\omega_1)$;

$\Pi_f = \varnothing, \forall f \neq 1$;

for $k = 2$ **to** n **do**

//Assignment rule

$\delta :=$ Vector of factories sorted in non-decreasing total completion time;

$B :=$Total completion time of factory $\delta[1]$ after testing job ω_k on the position l^* yielding the minimal total completion time of partial sequence $\Pi_{\delta[1]}$ ($B = C_{sum,\delta[1]}$);

$f^* := \delta[1]$;

for $f = 2$ **to** F **do**

$LB1_{\delta[f]}:=$ Lower bound of inserting ω_k in sequence $\Pi_{\delta[f]}$ according to Theorem 3.1;

if $LB1_{\delta[f]} < B$ **then**

$C_{sum,\delta[f]} :=$ Total completion time of factory $\delta[f]$ after testing job ω_k on the position $l^{'}$ of partial sequence $\Pi_{\delta[f]}$ yielding the minimal total completion time on that factory;

if $C_{sum,\delta[f]} < B$ **then**

$B = C_{sum,\delta[f]}$;

$l^* = l^{'}$;

$f^* = \delta[f]$;

end

end

end

//Construction of partial sequences

$\Pi_{f^*}:=$ Sequence formed by inserting job ω_k in position l^* of partial sequence Π_{f^*};

end

end

Figure 3: $NEH(\mathcal{R}_2, \mathcal{A}_3^{th1})$

Procedure $NEH(\mathcal{R}_2, \mathcal{A}_4^{th1})$

//Initial Order

$\Omega :=$ Jobs sorted in non-increasing sums of processing times ($\Omega = \{\omega_1, \ldots, \omega_n\}$);

$\Pi_1 := (\omega_1)$;

$\Pi_f = \varnothing, \forall f \neq 1$;

for $k = 2$ **to** n **do**

//Assignment rule

$\delta :=$ Vector of factories sorted according to increasing total completion time;

$C_{sum}^{best} :=$ Total completion time ($C_{sum}^{best} = \sum_{f \in \mathcal{F}} C_{sum,f}^{best}$) after testing job ω_k on the position l^* of partial sequence $\Pi_{\delta[1]}$ of Factory $\delta[1]$ yielding the minimal total completion time of partial sequence $\Pi_{\delta[1]}$;

$f^* := \delta[1]$;

for $f = 2$ **to** F **do**

$LB1_{\delta[f]}:=$ Lower bound of inserting ω_k in sequence Π_f according to Theorem 3.1;

if $LB1_{\delta[f} + \sum_{f_1 \in \mathcal{F} - \{\delta[f]\}} C_{sum,f}^{best} < C_{sum}^{best}$ **then**

$C_{sum} :=$ Total completion time after testing job ω_k on the position $l^{'}$ of factory $\delta[f]$ yielding the minimal total completion time;

if $C_{sum} < C_{sum}^{best}$ **then**

$C_{sum}^{best} = C_{sum}$;

$l^* = l^{'}$;

$f^* = \delta[f]$;

end

end

end

//Construction of partial sequences

$\Pi_{f^*}:=$ Sequence formed by inserting job ω_k in position l^* of partial sequence Π_{f^*};

end

end

Figure 4: $NEH(\mathcal{R}_2, \mathcal{A}_4^{th1})$

6 Proposed simple evolutionary algorithm

In this section we propose a standard metaheuristic to test both the efficiency of the problem properties and encoding procedures introduced in Section 4 (i.e. the representation of the solutiosn, factory assignment rules, and speed up procedures), and the constructive heuristics presented in Section 5. More specifically, we propose a simple Evolutionary Algorithm, denoted as EA, which iteratively looks for the local optimum in the neighbourhood of the mutated individuals in the population (examples of the excellent performance of evolutionary algorithms for flowshop scheduling problems can be found in Ruiz et al., 2006, Vallada and Ruiz, 2010, Sukkerd and Wuttipornpun, 2016, and Shao et al., 2017). The steps of the proposed metaheuristic are the following:

1. Initial population
2. Repeat until the stopping criterion is reached:
   (a) Selection
   (b) Mutation
   (c) Evaluation
   (d) Local search
   (e) Update of population

Given the computational results found for the constructive heuristics in Section 7, we use $\mathcal{R}_2$ to represent the solutions (chromosome representation) in the EA. In addition, let γ be the population size and P represent the population, i.e. $P := (\Pi^1, \ldots, \Pi^\gamma)$. Below, we further detail the phases of the proposed algorithm:

- Initial population: Each individual in the population is constructed according to $NEH(\mathcal{R}_2, \mathcal{A}_4)$ but changing the initial order of this heuristic. Thus, an individual is first constructed using the original order (i.e. decreasing sum of processing times), while a random initial order is used for the rest of the individuals. Once each individual is generated, we look for improvement in the neighbourhood of the best individual by applying the relative local

search method explained next. After this local search, the population update phase is performed.

- Selection: In this phase, an individual of the population is randomly selected.
- Mutation: The selected individual is mutated in this phase. The mutation randomly interchanges two jobs of the individual. More specifically, the procedure first chooses two random factories and two positions of these factories. If the factories and positions are not the same, then the jobs are interchanged.
- Evaluation: The mutated individual is evaluated. If the best-so-far solution is improved, it is updated.
- Relative local search (RLS): This method iteratively removes a job from the actual solution and re-inserts it according to the $\mathcal{A}_4$ assignment rule. The job to be removed in each iteration, denoted as σ, is the kth job from the reference solution, Π^{ref}. Note that k is the same as in the last call of the function. The pseudocode of this method is shown in Figure 5. In this procedure we use the current best solution, Π^{best}, as the reference solution.
- Population update: In this phase, the current population is updated. Firstly, all individuals in the population are removed if the best solution has been improved in the last iteration. Otherwise, the new individual is introduced in the population if it was not already there, and $|P|$ is increased by 1. Finally, if $|P|$ is greater than γ, then the worst individual of the population is removed. The pseudocode is given in Figure 6.

A general pseudocode of the EA is shown in Figure 7. Regarding the stopping criterion, we stop the algorithm when a certain time based on the size of the problem is reached (see Section 7 for details).

Procedure *RLS(*Π, Π^{ref}, k*)*

```
for i = 1 to n do
    σ:= kth job in Π^ref.
    Π'  := Solution obtained after removing job σ from Π;
    Π'' := Insert job σ in Π' applying the A_4 assignment rule;
    if C_sum(Π'') < C_sum(Π) then
        Π = Π'';
        i = 1;
    end
    k := k + 1;
    if k > n then
        k = 1;
    end
end
return Π, k;
```

end

Figure 5: Relative local search, RLS

Procedure *Population_update(*P, Π, $flag$*)*

```
if flag=true then
    P = ∅; //P is emptied;
end
if Π ∉ P then
    P := P ∪ {Π}; //append Π
    if |P| > γ then
        Find Π^w the worst individual in P;
        P := P − {Π^w}; //remove Π^w from P
    end
end
return P;
```

end

Figure 6: Population update

Procedure *EA()*

```
    //Initial Population
    k:=1; improvement:=false;
    Π^1:= NEH(R_2, A_4);
    P := {Π^1}; Π^best := Π^1;
    for i = 2 to γ do
        Π^i:= NEH(R_2, A_4) using a random sequence as initial solution;
        P := P ∪ {Π^i};
        if C_sum(Π^i) < C_sum(Π^best) then
            Π^best := Π^i;
        end
    end
    //Relative local search
    (k, Π'):=RLS(Π^best, Π^best, k);
    if C_sum(Π') < C_sum(Π^best) then
        Π^best := Π';
        improvement=true;
    end
    //Population updated
    P=Population_update(P, Π', improvement);
    while stopping criterion is not reach do
        improvement=false;
        //Selection
        Π^s:=Random sequence from P;
        //Mutation
        repeat
            (f_1, f_2):= Two randomly chosen factories;
            (p_1, p_2):= Two randomly chosen positions of factories f_1 and f_2, respectively;
        until !((f_1 = f_2) && (p_1 = p_2));
        Π'':= Sequence Π^s of population P where the jobs in position p_1 of factory f_1, and in
        position p_2 of factory f_2 are interchanged;
        //Evaluation
        if C_sum(Π'') < C_sum(Π^best) then
            Π^best := Π'';
        end
        //Relative local search
        (k, Π'):=RLS(Π'', Π^best, k);
        if C_sum(Π') < C_sum(Π^best) then
            Π^best := Π';
            improvement:=true;
        end
        //Population update
        P:=Population_update(P, Π',improvement);
    end
    return Π^i;
end
```

Figure 7: Evolutionary Algorithm, EA

7 Computational Results

In this section we present the computational results obtained in our study, organised as follows: EA is calibrated in Subsection 7.1; the proposed NEH-based constructive heuristics are compared in Subsection 7.2; and finally, EA is compared against the state-of-the-art metaheuristics of the most related scheduling problem in Subsection 7.3. The experimentation has been carried out on a Intel Core i7-3770 with 3.4 GHz and 16 GB RAM. Each algorithm under study has been completely re-coded using the same programming language (C#) and using the same common functions and libraries to have a fair comparison between them.

7.1 Experimental parameter tuning

The proposed EA metaheuristic has a single parameter γ (population size) which must be calibrated. After some preliminary tests to bound the range of this parameter, we tested the following values: $\gamma \in \{2, 3, 4, 5, 6, 7, 8, 9\}$. The calibration has been carried out on a benchmark with $I = 600$ instances. This benchmark, denoted as β_1, is composed of 120 combinations of the parameters n, m, and F, as follows: $n \in \{20, 50, 100, 200, 500\}$, $m \in \{5, 10, 15, 20\}$, and $F \in \{2, 3, 4, 5, 6, 7\}$, with five instances for each combination. The processing times are generated using a uniform distribution $[1, 99]$. To increase the power of the calibration, we perform five runs per instance and the average values are stored. For each run, the metaheuristic is stopped after $n \cdot m \cdot F \cdot 1.5$ milliseconds. Note that this benchmark is different than the one proposed for the comparisons of the algorithms (β_2), in order to avoid an overcalibration of parameter γ.

The computational results are shown in Figure 8 in terms of $ARPD1$, defined by the following expression (see e.g. Pan and Ruiz, 2013; Ribas et al., 2015; Fernandez-Viagas et al., 2018):

$$ARPD1 = \frac{1}{I} \sum_{i=1}^{I} \frac{C_{sum}^{i} - Best_i}{Best_i}, \tag{7}$$

where C_{sum}^{i} is the total completion time for instance i, with $i \in [1, I]$, and $Best_i$ is the best total completion times among all tested parameters. The best results are found for $\gamma = 5$, which is used in the subsequence experiments. In addition, we carry out a non-parametric Kruskal-Wallis analysis which reveals that there is statistically significant difference between the levels

of the parameters (all p-values found equals to 0.000).

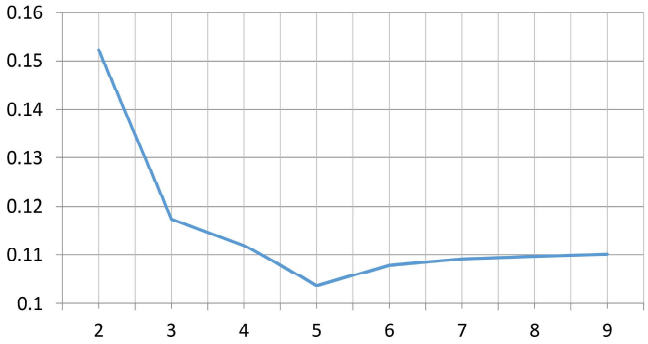

Figure 8: γ (x-axis) versus ARPD1 (y-axis)

7.2 Comparison among constructive heuristics

The constructive heuristics described in Section 5 are evaluated using the set of instances of Naderi and Ruiz (2010) (available in http://soa.iti.es). This benchmark, denoted as β_2, is composed of 720 instances with the following values of $n \in \{20, 50, 100, 200, 500\}$, $m \in \{5, 10, 15, 20\}$, and $F \in \{2, 3, 4, 5, 6, 7\}$, with 10 instances for each combination of parameters. The processing times are generated using a uniform distribution $[1, 99]$. In addition, we compare the heuristics by means of the $ARPD2$ (see Equation 8), and of the average CPU time (in seconds), denoted as ACT. Note that the $ARPD2$ uses the best total completion times among all tested algorithms for each instance i, $Best_i^{'}$ (each value of $Best_i^{'}$ is presented as on-line material).

$$ARPD2 = \frac{1}{I}\sum_{i=1}^{I}\frac{C_{sum}^{i} - Best_i^{'}}{Best_i^{'}}, \tag{8}$$

The computational results are summarised in Table 1. It is worth highlighting the excellent performance of the $\mathcal{R}_2$ representation of the solutions as compared to $\mathcal{R}_1$. Thus, $NEH(\mathcal{R}_2, \mathcal{A}_i)$ clearly outperforms $NEH(\mathcal{R}_1, \mathcal{A}_i)$ both in terms of quality of the solutions and computational effort, regardless the assignment used rule (i.e. $\forall i \in [1, 6]$). For instance, the $ARPD2$ and ACT values are 16.355 and 2.288 for $NEH(\mathcal{R}_1, \mathcal{A}_1)$, while 3.720 and 0.027 for $NEH(\mathcal{R}_2, \mathcal{A}_1)$, respectively. In terms of $ARPD2$, the best results are found for $NEH(\mathcal{R}_2, \mathcal{A}_4^{th1})$, $NEH(\mathcal{R}_2, \mathcal{A}_4^{th2})$, and $NEH(\mathcal{R}_2, \mathcal{A}_4^{th3})$. Graphically, computational results are shown in Figure 9. To make the figure

clearer, $NEH(\mathcal{R}_2, \mathcal{A}_3)$, $NEH(\mathcal{R}_2, \mathcal{A}_3^{th1})$, $NEH(\mathcal{R}_2, \mathcal{A}_3^{th3})$, $NEH(\mathcal{R}_2, \mathcal{A}_4^{th1})$, $NEH(\mathcal{R}_2, \mathcal{A}_4^{th2})$, and $NEH(\mathcal{R}_2, \mathcal{A}_4^{th3})$ are removed, as their results are similar to those by $NEH(\mathcal{R}_2, \mathcal{A}_3^{th2})$ and $NEH(\mathcal{R}_2, \mathcal{A}_4)$.

In view of the results, the efficient heuristics for the problem are: $NEH(\mathcal{R}_2, \mathcal{A}_1)$, $NEH(\mathcal{R}_2, \mathcal{A}_2)$, $NEH(\mathcal{R}_2, \mathcal{A}_4)$, $NEH(\mathcal{R}_2, \mathcal{A}_4^{th2})$, $NEH(\mathcal{R}_2, \mathcal{A}_5)$, and $NEH(\mathcal{R}_2, \mathcal{A}_6)$. A non-parametric Wilcoxon signed-rank test was carried out to support this statement by comparing $NEH(\mathcal{R}_2, \mathcal{A}_3^{th2})$ against $NEH(\mathcal{R}_2, \mathcal{A}_5)$, which resulted in a p-value of 0.000. Thereby, due to the excellent results in terms of $ARPD2$ and ACT, we use the $NEH(\mathcal{R}_2, \mathcal{A}_4)$ heuristic as the initial solution of the proposed EA (see Section 6).

Finally, note that the use of Theorem 3.2 to reduce the number of factories analysed in $NEH(\mathcal{R}_2, \mathcal{A}_3)$ decreases the average CPU time around 15% (e.g. ACT equals to 0.09 for $NEH(\mathcal{R}_2, \mathcal{A}_3)$, and 0.076 for $NEH(\mathcal{R}_2, \mathcal{A}_3^{th2})$). However, this reduction is still not enough to make the heuristic efficient. In addition, its use on $NEH(\mathcal{R}_2, \mathcal{A}_4)$ does not reduce its CPU times, although slightly decreases the $ARPD2$ values. In this case the increase in the complexity of the algorithm due to the computation of the lower bounds has more influence on the computational effort than on the number of discarded factories.

$NEH(\cdot,\cdot)$	$ARPD2$	ACT	$NEH(\cdot,\cdot)$	$ARPD2$	ACT	$NEH(\cdot,\cdot)$	$ARPD2$	ACT
$\mathcal{R}_1, \mathcal{A}_1$	16.355	2.288	$\mathcal{R}_2, \mathcal{A}_1$	3.720	0.027	$\mathcal{R}_2, \mathcal{A}_3^{th1}$	2.057	0.079
$\mathcal{R}_1, \mathcal{A}_2$	15.654	2.279	$\mathcal{R}_2, \mathcal{A}_2$	3.298	0.028	$\mathcal{R}_2, \mathcal{A}_3^{th2}$	2.057	0.076
$\mathcal{R}_1, \mathcal{A}_3$	16.061	12.900	$\mathcal{R}_2, \mathcal{A}_3$	2.053	0.090	$\mathcal{R}_2, \mathcal{A}_3^{th3}$	2.057	0.097
$\mathcal{R}_1, \mathcal{A}_4$	13.077	12.876	$\mathcal{R}_2, \mathcal{A}_4$	0.270	0.090	$\mathcal{R}_2, \mathcal{A}_4^{th1}$	0.257	0.092
$\mathcal{R}_1, \mathcal{A}_5$	13.168	10.757	$\mathcal{R}_2, \mathcal{A}_5$	0.906	0.061	$\mathcal{R}_2, \mathcal{A}_4^{th2}$	0.257	0.091
$\mathcal{R}_1, \mathcal{A}_6$	14.057	8.655	$\mathcal{R}_2, \mathcal{A}_6$	1.394	0.047	$\mathcal{R}_2, \mathcal{A}_4^{th3}$	0.257	0.096

Table 1: Average results in terms of $ARPD2$ and ACT of each constructive heuristic

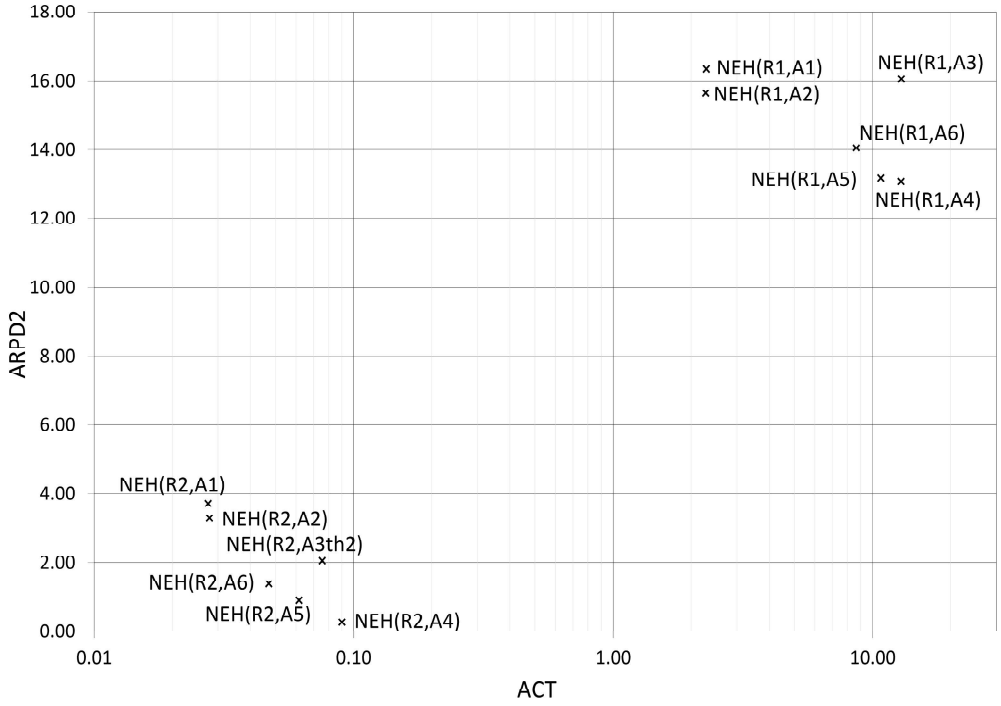

Figure 9: $ARPD2$ against ACT for the constructive heuristics.

7.3 Comparison of metaheuristics

In this subsection, we compare the proposed metaheuristic against the most efficient metaheuristics for the $DF|prmu|C_{max}$ problem, which are the BSIG, MIG, and SS metaheuristics proposed by Fernandez-Viagas and Framinan (2015a), Lin et al. (2013), and Naderi and Ruiz (2014), respectively (see Section 2). Note that, when adapting the BSIG, we remove the parts based on the specific properties of the makespan objective. After re-coding these algorithms, they are run on benchmark β_2, under three different stopping criteria: $t \cdot n \cdot m \cdot F$ with $t \in \{0.5, 1, 2\}$ (see e.g. Fernandez-Viagas and Framinan, 2015a for similar criteria).

The computational results are shown in Table 2 grouped by parameters n and m, and in Table 3 grouped by parameter F. The best results are found by EA, regardless the stopping criterion. More specifically, we obtain an $ARPD2$ value of 0.364, 0.209, and 0.056 for the stopping criteria $t = 0.5$, $t = 1$, and $t = 2$, respectively. Regarding the adapted algorithms, it is noteworthy to mention the excellent performance of SS, which clearly outperforms the other metaheuristics (MIG and BSIG). The average results and the trend of the metaheuristics are

also shown in Figure 10. The excellent convergence of the proposed metaheuristic when the stopping criterion increases can be observed. Thus, the $ARPD2$ of EA decreases from 0.209 to 0.056 (a reduction of 0.153) from stopping criterion $t = 1$ to $t = 2$, while SS decreases from 0.336 to 0.230 (a reduction of 0.106). Finally, three non-parametric Mann-Whitney tests are carried out to establish statistically significant differences between the proposed EA metaheuristic and SS. Each test checks the hypothesis that the means found for both metaheuristics are equal, for each stopping criterion. The results show that EA statistically outperforms SS since the three hypotheses are rejected, being the p-value found in each test equal to 0.000.

n x m	NEH	$t = 0.5$				$t = 1$				$t = 2$			
	$(\mathcal{R}_2, \mathcal{A}_4)$	MIG	BSIG	SS	EA	MIG	BSIG	SS	EA	MIG	BSIG	SS	EA
20 x 5	2.746	0.092	0.386	0.322	0.045	0.063	0.325	0.279	0.030	0.054	0.257	0.232	0.021
20 x 10	2.455	0.085	0.342	0.203	0.027	0.083	0.293	0.178	0.015	0.065	0.257	0.149	0.009
20 x 20	2.173	0.057	0.231	0.138	0.010	0.043	0.192	0.122	0.009	0.031	0.178	0.107	0.005
50 x 5	4.539	0.618	1.517	0.692	0.444	0.440	1.259	0.615	0.209	0.347	1.057	0.545	0.032
50 x 10	3.798	0.832	1.324	0.622	0.308	0.691	1.102	0.534	0.168	0.594	0.916	0.536	0.002
50 x 20	3.718	0.655	0.990	0.461	0.198	0.553	0.811	0.456	0.075	0.491	0.658	0.408	0.012
100 x 5	4.771	0.991	1.559	0.473	0.663	0.753	1.266	0.248	0.419	0.567	1.017	0.110	0.094
100 x 10	4.452	1.199	1.598	0.517	0.590	1.013	1.305	0.343	0.336	0.875	1.064	0.210	0.059
100 x 20	3.743	1.157	1.356	0.484	0.436	1.011	1.156	0.318	0.231	0.901	0.915	0.266	0.039
200 x 10	3.944	1.554	1.454	0.587	0.708	1.337	1.141	0.319	0.459	1.114	0.818	0.010	0.229
200 x 20	3.303	1.338	1.276	0.471	0.506	1.170	1.031	0.252	0.323	0.978	0.745	0.021	0.110
500 x 20	2.572	1.417	1.318	0.737	0.437	1.216	1.135	0.366	0.234	1.051	0.987	0.165	0.066
Average	3.518	0.833	1.113	0.476	0.364	0.698	0.918	0.336	0.209	0.589	0.739	0.230	0.056

Table 2: $ARPD2$ of each metaheuristic grouped by n and m

F	NEH	$t = 0.5$				$t = 1$				$t = 2$			
	$(\mathcal{R}_2, \mathcal{A}_4)$	MIG	BSIG	SS	EA	MIG	BSIG	SS	EA	MIG	BSIG	SS	EA
2	4.319	1.289	1.075	0.514	0.467	1.071	0.786	0.331	0.260	0.890	0.575	0.172	0.063
3	3.767	0.993	1.129	0.484	0.409	0.839	0.921	0.309	0.246	0.712	0.707	0.162	0.078
4	3.523	0.833	1.061	0.516	0.373	0.681	0.884	0.326	0.237	0.578	0.717	0.206	0.072
5	3.303	0.712	1.238	0.441	0.348	0.606	1.059	0.349	0.201	0.513	0.903	0.258	0.055
6	3.171	0.625	1.125	0.448	0.306	0.525	0.977	0.329	0.163	0.449	0.803	0.285	0.044
7	3.023	0.546	1.047	0.451	0.283	0.464	0.882	0.369	0.148	0.392	0.728	0.295	0.028
Average	3.518	0.833	1.113	0.476	0.364	0.698	0.918	0.336	0.209	0.589	0.739	0.230	0.056

Table 3: $ARPD2$ of each metaheuristic grouped by parameter F

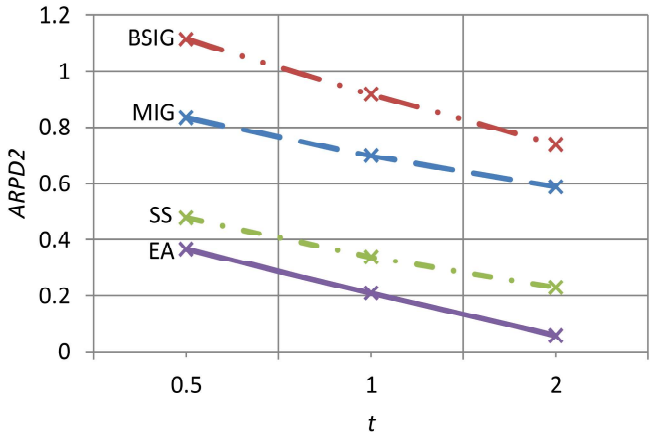

Figure 10: Evolution of the metaheuristics

8 Conclusions

In this paper, we address the distributed permutation flow shop scheduling problem to minimise the total completion time. First, we analyse in detail the problem properties, obtaining three different dominance rules. Secondly, we propose and compare two representations of the solutions and six assignment rules by inserting them in NEH-based algorithm. In addition, we propose two speed up procedures to reduce the number of evaluations in the computation of the total completion time. Finally, we present a simple evolutionary algorithm to solve the problem.

The experiments performed show, on the one hand, the excellent performance of the proposed constructive heuristics when a sequence for each factory is employed as a representation of

the solutions. By doing so, the average relative percentage deviation is substantially reduced regardless the assignment rule used. On the other hand, the proposed evolutionary algorithm can be considered as the state-of-the-art metaheuristic for the problem, since it statistically outperforms the adaptations of efficient algorithms for related problems.

Acknowledgements

The authors are sincerely grateful to the anonymous referees, who provide very valuable comments on the earlier version of the paper. This research has been funded by the Spanish Ministry of Science and Innovation, under the project "PROMISE" with reference DPI2016-80750-P.

References

Cao, D. and Chen, M. (2003). Parallel flowshop scheduling using tabu search. *International Journal of Production Research*, 41(13):3059–3073.

Companys, R., Ribas, I., and Mateo, M. (2010). Improvement tools for neh based heuristics on permutation and blocking flow shop scheduling problems. *IFIP Advances in Information and Communication Technology*, 338 AICT:33–40.

David, W., Kusiak, A., and Artiba, A. (1996). A scheduling problem in glass manufacturing. *IIE Transactions (Institute of Industrial Engineers)*, 28(2):129–139.

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197.

Deng, J., Wang, L., Wang, S.-Y., and Zheng, X.-L. (2017). A competitive memetic algorithm for the distributed two-stage assembly flow-shop scheduling problem. *International Journal of Production Research*, 54(12):3561–3577.

Fernandez-Viagas, V., Dios, M., and Framinan, J. (2016a). Efficient constructive and composite heuristics for the permutation flowshop to minimise total earliness and tardiness. *Computers and Operations Research*, 75:38–48.

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

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

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

Fernandez-Viagas, V., Leisten, R., and Framinan, J. (2016b). A computational evaluation of

constructive and improvement heuristics for the blocking flow shop to minimise total flowtime. *Expert Systems with Applications*, 61:290–301.

Fernandez-Viagas, V., Ruiz, R., and Framinan, J. (2017). A new vision of approximate methods for the permutation flowshop to minimise makespan: State-of-the-art and computational evaluation. *European Journal of Operational Research*, 257(3):707–721.

Fernandez-Viagas, V., Valente, J., and Framinan, J. (2018). Iterated-greedy-based algorithms with beam search initialization for the permutation flowshop to minimise total tardiness. *Expert Systems with Applications*, 94:58–69.

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

Gao, J. and Chen, R. (2011). A hybrid genetic algorithm for the distributed permutation flowshop scheduling problem. *International Journal of Computational Intelligence Systems*, 4(4):497–508.

Gao, J., Chen, R., and Deng, W. (2013). An efficient tabu search algorithm for the distributed permutation flowshop scheduling problem. *International Journal of Production Research*, 51(3):641–651.

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

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.

Kahn, K., Castellion, G., and Griffin, A. (2004). *The PDMA Handbook of New Product Development: Second Edition*. Wiley.

Li, X., Wang, Q., and Wu, C. (2009). Efficient composite heuristics for total flowtime minimization in permutation flow shops. *Omega*, 37(1):155–164.

Lin, J. and Zhang, S. (2016). An effective hybrid biogeography-based optimization algorithm for the distributed assembly permutation flow-shop scheduling problem. *Computers and Industrial Engineering*, 97:128–136.

Lin, S.-W. and Ying, K.-C. (2016). Minimizing makespan for solving the distributed no-wait flowshop scheduling problem. *Computers and Industrial Engineering*, 99:202–209.

Lin, S.-W., Ying, K.-C., and Huang, C.-Y. (2013). Minimising makespan in distributed permutation flowshops using a modified iterated greedy algorithm. *International Journal of Production Research*, 51(16):5029–5038.

Liu, J. and Reeves, C. (2001). Constructive and composite heuristic solutions to the $P||\sum c_i$ scheduling problem. *European Journal of Operational Research*, 132:439–452.

Moon, C., Kim, J., and Hur, S. (2002). Integrated process planning and scheduling with minimizing total tardiness in multi-plants supply chain. *Computers and Industrial Engineering*, 43(1-2):331–349.

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

Naderi, B. and Ruiz, R. (2014). A scatter search algorithm for the distributed permutation flowshop scheduling problem. *European Journal of Operational Research*, 239(2):323–334.

Nawaz, M., Enscore, J. 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.

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.

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

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

Ribas, I., Companys, R., and Tort-Martorell, X. (2015). An efficient discrete artificial bee colony algorithm for the blocking flow shop problem with total flowtime minimization. *Expert Systems with Applications*, 42(15-16).

Ribas, I., Companys, R., and Tort-Martorell, X. (2017). Efficient heuristics for the parallel blocking flow shop scheduling problem. *Expert Systems with Applications*, 74:41–54.

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

Ruiz, R., Maroto, C., and Alcaraz, J. (2006). Two new robust genetic algorithms for the flowshop scheduling problem. *Omega*, 34(5):461–476.

Shao, W., Pi, D., and Shao, Z. (2017). Memetic algorithm with node and edge histogram for no-idle flow shop scheduling problem to minimize the makespan criterion. *Applied Soft Computing Journal*, 54:164–182.

Sukkerd, W. and Wuttipornpun, T. (2016). Hybrid genetic algorithm and tabu search for finite capacity material requirement planning system in flexible flow shop with assembly operations. *Computers and Industrial Engineering*, 97:157 – 169.

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

Vairaktarakis, G. and Elhafsi, M. (2000). The use of flowlines to simplify routing complexity in two-stage flowshops. *IIE Transactions (Institute of Industrial Engineers)*, 32(8):687–699.

Vallada, E. and Ruiz, R. (2010). Genetic algorithms with path relinking for the minimum tardiness permutation flowshop problem. *Omega*, 38(1-2):57–67.

Vázquez-Rodríguez, J. and Ochoa, G. (2011). On the automatic discovery of variants of the neh procedure for flow shop scheduling using genetic programming. *Journal of the Operational Research Society*, 62(2):381–396.

Wang, K., Huang, Y., and Qin, H. (2016). A fuzzy logic-based hybrid estimation of distribution algorithm for distributed permutation flowshop scheduling problems under machine breakdown. *Journal of the Operational Research Society*, 67(1):68–82.

Wang, S.-Y., Wang, L., Liu, M., and Xu, Y. (2013). An effective estimation of distribution algorithm for solving the distributed permutation flow-shop scheduling problem. *International Journal of Production Economics*, 145(1):387–396.

Xu, K., Feng, Z., and Jun, K. (2010). A tabu-search algorithm for scheduling jobs with controllable processing times on a single machine to meet due-dates. *Computers and Operations Research*, 37(11):1924–1938.

Ying, K.-C., Lin, S.-W., Cheng, C.-Y., and He, C.-D. (2017). Iterated reference greedy algorithm for solving distributed no-idle permutation flowshop scheduling problems. *Computers and Industrial Engineering*, 110:413 – 423.

Zhang, X. and Van De Velde, S. (2012). Approximation algorithms for the parallel flow shop problem. *European Journal of Operational Research*, 216(3):544–552.