Contents lists available at ScienceDirect

Expert Systems With Applications

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

A speed-up procedure for the hybrid flow shop scheduling problem

Victor Fernandez-Viagas

Industrial Management, School of Engineering, University of Seville, Camino de los Descubrimientos s/n, 41092 Seville, Spain

ARTICLE INFO

Keywords:
Scheduling
Hybrid flowshop
Iterated greedy
Speed-up
Accelerations
Critical path
Makespan
Computational evaluation
HFS

ABSTRACT

During the last decades, hundreds of approximate algorithms have been proposed in the literature addressing flow-shop-based scheduling problems. In the race for finding the best proposals to solve these problems, speed-up procedures to compute objective functions represent a key factor in the efficiency of the algorithms. This is the case of the well-known Taillard's accelerations proposed for the traditional flow shop with makespan minimisation or several other accelerations proposed for related scheduling problems. Despite the interest in proposing such methods to improve the efficiency of approximate algorithms, to the best of our knowledge, no speed-up procedure has been proposed so far in the hybrid flow shop literature. To tackle this challenge, we propose in this paper a speed-up procedure for makespan minimisation, which can be incorporate in insertion-based neighbourhoods using a complete representation of the solutions. This procedure is embedded in the traditional iterated greedy algorithm. The computational experience shows that even incorporating the proposed speed-up procedure in this simple metaheuristic results in outperforming the best metaheuristic for the problem under consideration.

1. Introduction

The hybrid flow shop scheduling problem (denoted as HFSP) is one of the most common and studied problem in the scheduling literature, which arises from a natural extension of the traditional flow shop scheduling problem. In the HFSP, there are n jobs to be processed on m stages, where each job follows the same route of stages. In each stage, there are parallel machines which can process the jobs. The problem then consists in finding the best schedule which minimises a certain objective function. Due to its many applications in real manufacturing fields as iron and steel production, textiles, paper making, and chemicals (in this regard, see e.g. Bozorgirad & Logendran, 2016; Long et al., 2018; Peng et al., 2018), hundreds of contributions have been developed in the literature addressing this scheduling problem. Most of them focused in proposing new methods to solve the problem (Fernandez-Viagas & Framinan, 2020), since it was characterised as NP-hard, in most cases. This is also the case of the objective under consideration, makespan minimisation, whose NP-hard nature was demonstrated by Gupta (1988) for two stages and at least two machines in one. Note that, among the objective functions addressed in the literature to solve the problem, the makespan minimisation is the most employed one due to its close relationship with the maximisation of machine utilisation and/or the minimisation of production run. The problem under consideration can be denoted by $HFm||C_{max}$, $FFm||C_{max}$, or $FHm,(PM^k)_{k=1}^m||C_{max}$ (see Graham et al., 1979; Ruiz & Vázquez-Rodríguez, 2010).

The literature proposing approximated algorithms to efficiently solve this scheduling problem is very extensive. We refer in this regards to the reviews carried out by Ribas, Leisten et al. (2010), Ruiz and Vázquez-Rodríguez (2010). According to Fernandez-Viagas et al. (2017), these proposals can be divided in either heuristics or metaheuristics depending if they naturally stop or not. Regarding the former, several constructive and improvement heuristics have proposed to find a fast solution of the problem as e.g. Acero-Dominguez and Paternina-Arboleda (2004), Brah and Loo (1999), Fernandez-Viagas, Molina-Pariente et al. (2018), Kizilay et al. (2015), Pan et al. (2014) and Santos et al. (1996). Among them, the proposals developed by Fernandez-Viagas, Molina-Pariente et al. (2018) (two Johnson-based Heuristics, denoted by JbH1 and JbH2, and a Memory-based Constructive Heuristic, MCH) highlight as the state-of-the-art heuristics for the problem. Regarding the latter, we find also several proposals in the literature. An example of Artificial Immune Systems and Ant Colony Optimisations can be found in Alaykýran et al. (2007) and Engin and Döyen (2004), respectively. Quantum algorithm, Particle Swarm Optimisation, Estimation of Distribution Algorithm or Discrete Artificial Bee Colony have also been proposed by Liao et al. (2012), Niu et al. (2009), Pan et al. (2014), and Wang et al. (2013), respectively, while a local-search based iterated algorithm has been proposed by Negenman (2001). Among all these proposals, the Discrete Artificial Bee Colony (DABC), proposed by Pan et al. (2014), is so far the best performing metaheuristic for the problem under consideration. This efficiency was tested in an

E-mail address: vfernandezviagas@us.es.

https://doi.org/10.1016/j.eswa.2021.115903
Received 18 May 2021; Received in revised form 3 August 2021; Accepted 10 September 2021
Available online 21 September 2021

extensive computational evaluation comparing it with several other metaheuristics proposed for the problem under consideration or related one (as e.g. the ILS metaheuristic proposed by Naderi et al., 2010). In fact, nowadays this DABC metaheuristic is probably the cornerstone metaheuristic in the hybrid flowshop, where many variations of it are state-of-the-art algorithms for several hybrid flow shop scheduling problems (see e.g. Pan, 2016; Pan et al., 2017; Zhang et al., 2019). After this DABC metaheuristic, some metaheuristics have also been proposed in the literature as the Iterated Greedy (IG) proposed by Kizilay et al. (2015) or the Chaos-enhanced Simulated Annealing (CSA) proposed by Lin et al. (2021). The former metaheuristic was outperformed by the DABC algorithm (see Kizilay et al., 2015), while we are not aware of any previous comparison between DABC and CSA.

Despite the high number of proposals in the literature to solve the problem under consideration, recent advances in the literature hint that there is still room for new improvements. On the one hand, Fernandez-Viagas et al. (2019) review and compare different representations of the solutions used by the authors in the problem, as well as give some recommendations for future approaches. In this regard, while the use of a unique sequence to represent the solution is very extended in the literature, its combination with complete representations in local search phases is very scarce and is recommended by the authors. On the other hand, it is worth highlighting that in the reduced version of the problem when there is one machine in each stage (i.e. in the flowshop scheduling problem), the mechanism proposed by Taillard (1990) to accelerate approximated algorithms is still nowadays one of the key factors to achieve an effective performance in every proposal (Fernandez-Viagas et al., 2017). Although much effort has been made to propose speed-up procedures for related flow-shop-based scheduling problem (see e.g. Fernandez-Viagas & Framinan, 2015b; Fernandez-Viagas et al., 2020; Naderi & Ruiz, 2010; Rios-Mercado & Bard, 1998), to our best knowledge no success has been found so far for the problem under consideration.

To tackle these opportunities, the contribution of this paper is four-fold: we first present several definitions and theorems for the problem under consideration. Secondly, based on these theoretical results, we propose a speed-up procedure to reduce the complexity in insertion-based mechanisms by $O(n)$. Thirdly, we develop a simple iterated greedy algorithm to solve the problem using the previous speed-up procedure. And finally, we perform a computational evaluation comparing our proposal with the most promising metaheuristics in the literature, under the new set of instances proposed by Fernandez-Viagas and Framinan (2020).

To deal with these issues, the paper is organised as follows: in Section 2 we describe the problem under consideration. For this problem, we present some new theoretical results in Section 3. These results are used to propose a novelty speed-up procedure in Section 4. This procedure is incorporated in a simple local-search-based metaheuristic in Section 5, which is compared with the state-of-the-art metaheuristics in Section 6. Finally, some conclusions are discussed in Section 7.

2. Problem description and background

The problem under consideration can be defined as follows. There are n jobs to be processed in each one of m stages, following each job the same route across the stages. Each job σ is composed of m operation to be processed in each stage of the shop. In stage i, there are m_i identical machines to perform the operations. Let $O_{i\sigma}$ denote the operation which corresponds to job σ performed on stage i. The processing time of job σ when is processed on stage i is denoted by $p_{i,\sigma}$.

The goal of the problem is to find the best schedule which minimises the makespan, C_{max}. Since it is clear that at least one optimal solution can be reached by considering semi-active schedules (see Pinedo, 1995 for a definition), we will use the term schedule in this paper to refer exclusively to semi-active schedules. In this case, a schedule, $\mathcal{S}$, can be fully defined by a sequence of jobs on each machine and consequently, there are $|\mathcal{S}|$ different solutions represented by the following equation (Urlings et al., 2010):

$$|S| = (n!)^m \prod_{i=1}^{m} \binom{n+m_i-1}{m_i-1} \tag{1}$$

In order to decrease the number of solutions to be explored for an algorithm, note that several authors have considered different representations of the solutions in the HFS. In this regard, we refer to Fernandez-Viagas et al. (2019) for a complete review and analysis of them.

Considering a full representation of the solution formed by the sequences of jobs on each machine (denoted as $\mathcal{R}_1^S$ by Fernandez-Viagas et al., 2019), let $\Pi_{ik} := (\pi_{ik1}, \ldots, \pi_{ikj}, \ldots, \pi_{ikn_{ik}})$ denote the sequence of jobs in stage i and machine k, where n_{ik} is the number of jobs assigned to that machine. In this case, the completion time of each job π_{ikj} on stage i (i.e. operation $O_{i\pi_{ikj}}$), denoted as $C_{i,\pi_{ikj}}$, can be calculated as:

$$C_{i,\pi_{i,k,j}} = \max\{C_{i-1,\pi_{i-1,k,j}}, C_{i,\pi_{i,k,j-1}}\} + p_{i,\pi_{i,k,j}},$$
$$\forall i \in \{1,\ldots,m\}, k \in \{1,\ldots,m_i\}, j \in \{1,\ldots,n_{ik}\} \tag{2}$$

with $C_{0,\pi_{0,k,j}} = C_{i,\pi_{i,k,0}} = 0$.

A Mixed Integer Linear Programming model for the problem can be stated as in Box I (Naderi et al., 2014).
where M is a large number, X_{ijl} is equal to 1 if job j is processed after job k and X_{ijk} (0 otherwise), while Y_{ilj} is 1 if job j is assigned to machine k at stage i (0 otherwise). Regarding the constraints, Eq. (M1) impose that each job is processed in a unique machine on each stage. The completion times are calculated by Eqs. (M2), (M3), and (M4). The objective function (makespan) is defined in Eqs. (M5). Finally, Eqs. (M6), (M7), and (M8) limit the variables of the model.

3. Preliminary theoretical results

In this section, we present some theoretical tools needed to propose our methodology in Section 4. However, prior to prove some relevant results for our proposal, several definitions are necessary. Firstly, once a representation of the solutions Π_{ik} has been defined, a schedule has to be constructed by placing the jobs. The most traditional approach in this regard is the forward schedule (see Definition 3.1). However, in order to demonstrate some of the theoretical results included in this paper, we will also make use of the reverse schedule (see Definition 3.2), also denoted as backward schedule (see e.g. Pan et al., 2014; Wang et al., 2013; Xu et al., 2013 for previous uses of this approach), which was originally proposed for the traditional flowshop (see Ribas, Companys et al., 2010; Ribas et al., 2013). After that, we define two important characteristics of the problem (critical set and path) and based on them, we present several theorems.

Definition 3.1 (*Forward Schedule*)**.** Given an instance $\mathcal{I}$ of the $HFm||C_{max}$ problem and a solution $\Pi_{ik} := (\pi_{ik1}, \ldots, \pi_{ikj}, \ldots, \pi_{ikn_{ik}})$, let us define $\mathcal{S}^F$ the forward schedule as the schedule obtained when this solution is represented without forcing idle times, and in ascending order of stages and jobs by applying Eq. (2). As usual, jobs are grouped from the left to the right.

Definition 3.2 (*Backward Schedule*)**.** Given an instance $\mathcal{I}$ of the $HFm||C_{max}$ problem and a solution $\Pi_{ik} := (\pi_{ik1}, \ldots, \pi_{ikj}, \ldots, \pi_{ikn_{ik}})$, let us define $\mathcal{S}^B$ the backward schedule as the schedule obtained when this solution is represented without forcing idle times, and in descending order of stages and positions (i.e. starting to represent the solution from the last stage and the last job on each machine) by applying Eq. (3). Jobs are then grouped from the right to the left (instead of from the left to the right).

minimise	C_{max}		
subject to			
$\sum_{k\in\{1,\dots,m_i\}} Y_{ikj} = 1$	$i \in \{1,\dots,m\}, j \in \{1,\dots,n\}$	$(M1)$
$C_{ij} \geq C_{i-1,j} + p_{ij}$	$i \in \{1,\dots,m\} - \{1\}, j \in \{1,\dots,n\}$	$(M2)$
$C_{ij} \geq C_{il} + p_{ij} - M \cdot (3 - X_{ijl} - Y_{ikj} - Y_{ikl})$	$i \in \{1,\dots,m\}, k \in \{1,\dots,m_i\}, j,l \in \{1,\dots,n\} \| l > j$	$(M3)$
$C_{il} \geq C_{ij} + p_{il} - M \cdot X_{ijl} - M \cdot (2 - Y_{ikj} - Y_{ikl})$	$i \in \{1,\dots,m\}, k \in \{1,\dots,m_i\}, j,l \in \{1,\dots,n\} \| l > j$	$(M4)$
$C_{max} \geq C_{mj}$	$j \in \{1,\dots,n\}$	$(M5)$
$C_{ij} \geq 0$	$i \in \{1,\dots,m\}, j \in \{1,\dots,n\}$	$(M6)$
$X_{ijl} \in \{0,1\}$	$i \in \{1,\dots,m\}, j,l \in \{1,\dots,n\} \| l > j$	$(M7)$
$Y_{ikj} \in \{0,1\}$	$i \in \{1,\dots,m\}, k \in \{1,\dots,m_i\}, j \in \{1,\dots,n\}$	$(M8)$

Box I.

Table 1
Sequences of jobs on each machine for the example.

Π_{ik}	Sequence
π_{11j}	(1,4,6)
π_{12j}	(2,3,5)
π_{21j}	(2,5,6)
π_{22j}	(1,4,3)
π_{31j}	(2,4,6)
π_{32j}	(1,5,3)

$$\bar{C}_{i,\pi_{i,k,j}} = \max\{\bar{C}_{i+1,\pi_{i+1,k,j}}, \bar{C}_{i,\pi_{i,k,j+1}}\} + p_{i,\pi_{i,k,j}},\quad \forall i \in \{m,\dots,1\}, k \in \{m_i,\dots,1\}, j \in \{n_{ik},\dots,1\} \tag{3}$$

with $\bar{C}_{m+1,\pi_{m+1i,k,j}} = \bar{C}_{i,\pi_{i,k,n_{ik}+1}} = 0$.

We show an example of forward and backward in Figs. 1 and 2, respectively. In this example we consider six jobs, three stages, and two machines per stages. The sequence of jobs followed on each stage is shown in Table 1, which is represented in forward and backward manner in both figures, respectively.

Next, two definitions, denoted as critical set and critical path, are needed both to identify the importance of each operation in the schedule and to be able to demonstrate some posterior theoretical results.

Definition 3.3 (*Critical Set*)**.** Given an instance $\mathcal{I}$ of the $HFm||C_{max}$ problem and a schedule $\mathcal{S}$, let us define $C_{\mathcal{I}}(\mathcal{S})$ the critical set of that schedule as the set formed for all operations whose starting times cannot be delayed without increasing the objective function of the problem.

Let $\mathcal{O}$ and $\mathcal{F}$ denote two artificial nodes which represent the initial and final operations, respectively. By using them, we can define the critical path of a schedule as follows.

Definition 3.4 (*Critical Path*)**.** Given an instance $\mathcal{I}$ of the $HFm||C_{max}$ problem and a schedule $\mathcal{S}$, let us define $\mathcal{P}_{\mathcal{I}}(\mathcal{S})$ the critical path of that schedule as the path formed by the operations in the critical set, which allow us going from $\mathcal{O}$ to $\mathcal{F}$ without repeating any operation and always moving to either the following job or stage.

Following with the previous example, we show the critical set and path of the previous schedule in Figs. 3 and 4, respectively. The critical path was introduced by Nowicki and Smutnicki (1996) in the $Fm|prmu|C_{max}$ problem and has been used for several approximated algorithms for that objective function (see e.g. Grabowski & Wodecki, 2004; Nowicki & Smutnicki, 1998, 2006), and recently by Fernandez-Viagas et al. (2020) to propose a speed-up procedure for the flowshop layout using other objective functions, which is the most related research in the literature. However, this previous speed-up procedure cannot be applied to the problem under consideration and we are not aware of previous acceleration procedures in the HFS problem. To cover this opportunity, we propose below several critical-path based theorems to be able to develop a new speed-up procedure in Section 4.

Theorem 3.1. *Given an instance $\mathcal{I}$ of the $HFm||C_{max}$ problem and a schedule $\mathcal{S}$, then at least one critical path $\mathcal{P}_{\mathcal{I}}(\mathcal{S})$ exists.*

Proof. We start the proof considering the operation with maximum completion time (or any of these operations in case there are several), which consequently defines the C_{max} of $\mathcal{S}$. According to Definition 3.3, it is clear that this operation belongs to the critical set $C_{\mathcal{I}}(\mathcal{S})$, as cannot be delayed. In addition, the completion time of this operation can be calculated using the following expression:

$$C_{i,\pi_{i,k,n_{ik}}} = \max\{C_{i-1,\pi_{i-1,k,n_{ik}}}, C_{i,\pi_{i,k,n_{ik}-1}}\} + p_{i,\pi_{i,k,n_{ik}}} \tag{4}$$

More specifically, this equation states that there is at least one operation ($O_{i-1,\pi_{i-1,k,n_{ik}}}$ or $O_{i,\pi_{i,k,n_{ik}-1}}$), which is connected with $O_{i,\pi_{i,k,n_{ik}}}$ without adding any idle or waiting time, i.e. one of the following two equations must be fulfil:

$$C_{i,\pi_{i,k,n_{ik}}} = C_{i-1,\pi_{i-1,k,n_{ik}}} + p_{i,\pi_{i,k,n_{ik}}} \tag{5}$$

$$C_{i,\pi_{i,k,n_{ik}}} = C_{i,\pi_{i,k,n_{ik}-1}} + p_{i,\pi_{i,k,n_{ik}}} \tag{6}$$

By Definition 3.3, this new operation must belong to set $C_{\mathcal{I}}(\mathcal{S})$, since delaying this operation to the right would necessarily worsen the makespan. Proceeding analogously, the completion time of a generic operation $O_{i,\pi_{i,k,j}}$ (with $i > 1$ and $j > 1$) belonging to $C_{\mathcal{I}}(\mathcal{S})$ can be calculated by:

$$C_{i,\pi_{i,k,j}} = \max\{C_{i-1,\pi_{i-1,k,j}}, C_{i,\pi_{i,k,j-1}}\} + p_{i,\pi_{i,k,j}} \tag{7}$$

And again, either operation $O_{i-1,\pi_{i-1,k,j}}$ or $O_{i,\pi_{i,k,j-1}}$ joins to the previous operation without adding waiting or idle time and therefore, this operation must also belong to $C_{\mathcal{I}}(\mathcal{S})$. This procedure can be repeated until i or j are equal to 1. But in both cases a similar conclusion can be obtained. Let beginning with case $j = 1$ for a generic i and k. In this case, the completion time of operation $O_{i,\pi_{i,k,1}}$ is calculated by the following expression, as it is the first job in machine k:

$$C_{i,\pi_{i,k,1}} = C_{i-1,\pi_{i-1,k,1}} + p_{i,\pi_{i,k,j}} \tag{8}$$

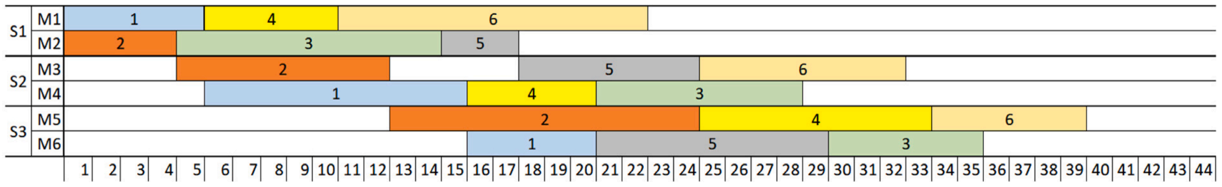

Fig. 1. Example of forward schedule with six jobs.

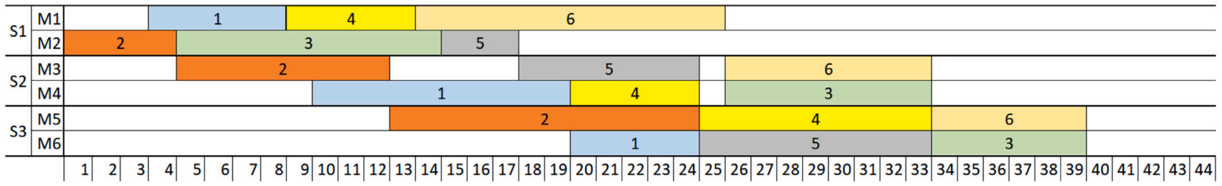

Fig. 2. Example of backward schedule with six jobs.

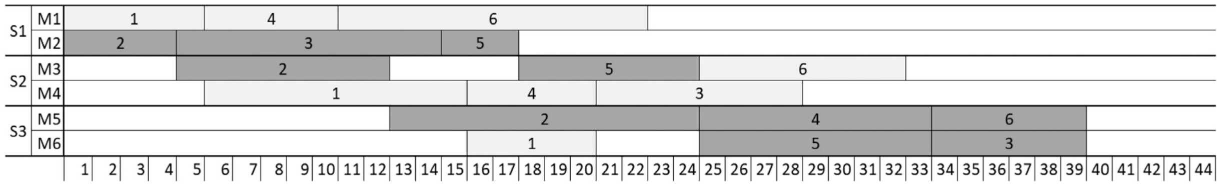

Fig. 3. Example of critical set.

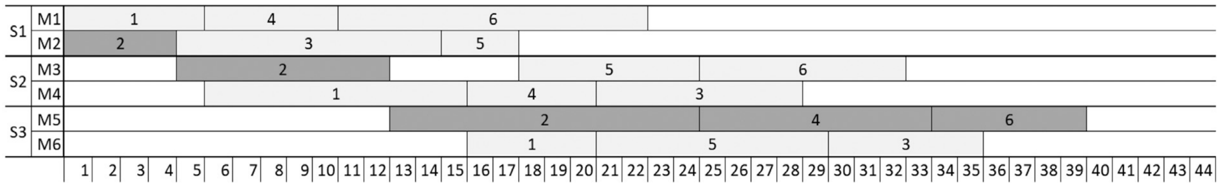

Fig. 4. Example of critical path.

Which means that operation $O_{i-1,\pi_{i-1,k,1}}$ must be in $C_{\mathcal{I}}(\mathcal{S})$. On the other hand, in case $i = 1$ the completion time of operation $O_{1,\pi_{1,k,j}}$ can be calculated by the following expression, as it is the first stage:

$$C_{1,\pi_{1,k,j}} = C_{1,\pi_{1,k,j-1}} + p_{i,\pi_{i,k,j}} \tag{9}$$

And again, operation $O_{1,\pi_{i,k,j-1}}$ must belong to $C_{\mathcal{I}}(\mathcal{S})$. Applying the procedure recursively, it finishes when we reach some operation $O_{1,\pi_{1,k,1}}$ also belonging to the critical set. I.e. we have moved from the final node $\mathcal{F}$ to the initial node $\mathcal{B}$ crossing only operation in the critical set and without both repeating any operation or moving to a previous stage or job, which corresponds to the definition of the critical path (see Definition 3.4). □

A result of this theorem is that the same makespan is obtained constructing a solution from a representation of the solutions Π_{ij}, regardless if it is schedule in a forward or backward manner (see Corollary 3.1).

Corollary 3.1. *Given an instance $\mathcal{I}$ of the $HFm||C_{max}$ problem, then the makespan obtained by the forward schedule $\mathcal{S}$ and its corresponding backward schedule $\mathcal{S}^B$ is the same.*

Proof. The proof of this corollary is obvious in view of Theorem 3.1. □

Finally, two more theoretical results are needed to prove that displacing some operation, both in the forward and backward schedules, would increase the makespan that time.

Theorem 3.2. *Given an instance $\mathcal{I}$ of the $HFm||C_{max}$ problem and a forward schedule $\mathcal{S}^F$, then displacing l units any operation (excepting the last one of the schedule) to the left would increases the total use of the system or makespan, exactly the same l units.*

Proof. The proof is obvious using the same reasoning as applied in Theorem 3.1. As a generic operation $O_{i,\pi_{i,k,j}}$ is moved to the left, then either $O_{i-1,\pi_{i-1,k,j}}$ or $O_{i,\pi_{i,k,j-1}}$ would be also moved to the left. The procedure could be recursively extended until to displace the first job of the schedule proceeding analogously than before (see example in Fig. 5). □

Theorem 3.3. *Given an instance $\mathcal{I}$ of the $HFm||C_{max}$ problem and a backward schedule $\mathcal{S}^B$, then displacing l units any operation (excepting the last one of the schedule) to the right would increases the total use of the system or makespan, exactly the same l units.*

Proof. The proof is obvious using the same reasoning as applied in Theorem 3.2. □

4. Proposed speed-up procedure

In this section, we detail the proposed speed-up procedure, which can be applied for full representations of the solutions ($\mathcal{R}_1^S$). This procedure makes use of the results obtained in Section 3 to highly reduce the computational effort needed. After presenting the accelerations, we incorporate the speed-up procedure in a insertion-based local search with partial selection of operations.

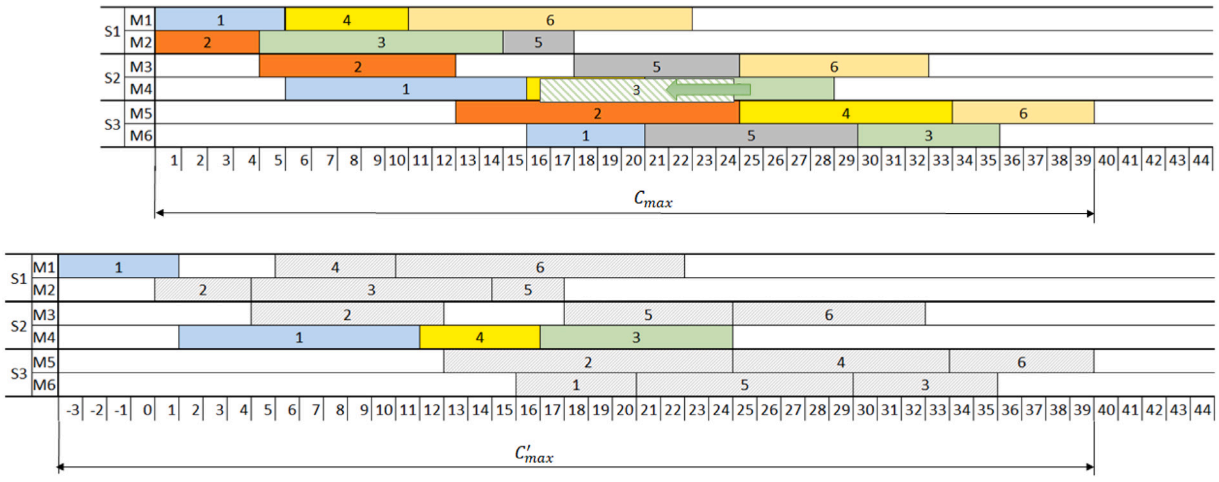

Fig. 5. Example of Theorem 3.2.

Firstly, let us suppose that operation $O_{i\sigma}$ of a solution Π_{ik} (belonging to job σ) has previously been removed from stage i and has to be reinserted in the same stage. Under this condition, the speed-up procedure to re-insert this operation in every position is detailed as follows:

STEP1. Calculate the forward schedule (i.e. the completion time $C_{i,\pi_{i,k,j}}$ of each operation $\pi_{i,k,j}$). Using the previous example, in Fig. 6, we calculate the completion times $C_{i,\pi_{i,k,j}}$ after removing job five from the second stage.

STEP2. Calculate the backward schedule (i.e. the completion time $\bar{C}_{i,\pi_{i,k,j}}$ of each operation $\pi_{i,k,j}$) beginning with $i = m$ and $j = n_{ik}$ for any $k \in m_i$. In Fig. 7 we calculate the completion times $\bar{C}_{i,\pi_{i,k,j}}$ after removing job five, i.e. $\sigma = 5$, from the second stage.

STEP3. For each machine $k \in m_i$

STEP 3.1. For each position $j \in [1, n_{ik}]$

STEP 3.1.1. Calculate the makespan incurred by introducing job σ in position j of machine k (denoted as C^{jk}_{max}) by using Theorem 4.1 (detailed below). See example of this calculation in Fig. 8.

$$C^{jk}_{max} = \max\{C_{max}, \max\{C_{i,\pi_{ikj}}, C_{i-1,\sigma}\} + p_{i,\sigma} + \max\{\bar{C}_{i,\pi_{i,k,j+1}}, \bar{C}_{i+1,\sigma}\}\}$$

STEP4. Insert operation $O_{i\sigma}$ in position j of machine k (stage i) which minimises C^{jk}_{max}.

STEP5. The new makespan is $C_{max} = \min_{\forall j,k}\{C^{jk}_{max}\}$.

Where Theorem 4.1 and its demonstration is stated as follows.

Theorem 4.1. *Given an instance $\mathcal{I}$ of the $HFm||C_{max}$ problem and a schedule $\mathcal{S}$ represented by Π_{ik} with a makespan equals to C_{max}, then the new makespan (C^{new}_{max}) obtained after inserting operation $O_{i\sigma}$ in position j on the machine k of stage i is:*

$$C^{new}_{max} = \max\{C_{max}, \max\{C_{i,\pi_{ikj}}, C_{i-1,\sigma}\} + p_{i,\sigma} + \max\{\bar{C}_{i,\pi_{i,k,j+1}}, \bar{C}_{i+1,\sigma}\}\} \quad (10)$$

Proof. When inserting operation σ in position j on the machine k of stage i, there are two possibilities:

1. No operation in $\mathcal{S}$ is affected.
2. We delay some operation in $\mathcal{S}$.

Regarding the first case, it is clear that the makespan stays the same if no operation of the schedule has been affected by the insertion, with the exception of an insertion in the last position of any machine in the last stage (i.e. $i = m$ and $j = n_{mk}$). In this last case, the new makespan should be the maximum between the last makespan (C_{max}) and the completion time of the new inserted operation (i.e. $\max\{C_{m,\pi_{m,k,n_{mk}}}, C_{m-1,\sigma}\} + p_{m,\sigma}$). As we have inserted the job in the last operation of the last stage, then $\bar{C}_{m,\pi_{m,k,n_{mk}+1}} = 0$ and $\bar{C}_{m+1,\sigma} = 0$ and Eq. (10) is satisfied.

Regarding the second case, according to Eq. (2), there are two operations which could be affected by this insertion which are either $\pi_{i+1,k,j}$ or $\pi_{i,k,j+1}$. The delay of these operations could potentially increase the value of the new makespan according to Theorem 3.3 (note that in that theorem is stated that a delay of any of these operations would delay the backward schedule exactly that value). In order to calculate this delay, firstly, we know that the new operation σ should start either when the previous one in the same machine is finished or when the same job in the previous stage finishes, i.e. $\max\{C_{max}, \max\{C_{i,\pi_{ikj}}, C_{i-1,\sigma}\}\}$, and it should therefore finish after this value plus its processing time, i.e. $\max\{C_{max}, \max\{C_{i,\pi_{ikj}}, C_{i-1,\sigma}\} + p_{i,\sigma}\}$. As a consequence, the posterior operation (either $\pi_{i+1,k,j}$ or $\pi_{i,k,j+1}$) cannot start before this time and then, all operations would be delayed. In addition, we know that the time between this posterior operation and the end is defined by the backward schedule, i.e. either $\bar{C}_{i+1,\sigma}$ or $\bar{C}_{i,\pi_{i,k,j+1}}$, respectively. Using both issues and taking into account that the new makespan must be greater than C_{max}, we have:

$$C^{new}_{max} = \max\{C_{max}, \max\{C_{i,\pi_{ikj}}, C_{i-1,\sigma}\} + p_{i,\sigma} + \max\{\bar{C}_{i,\pi_{i,k,j+1}}, \bar{C}_{i+1,\sigma}\}\} \quad (11)$$

which is exactly Eq. (10). □

Regarding the complexity of inserting operation $O_{i\sigma}$ in every position of stage i, we know that the calculation of the forward and backward schedules can be done in complexity $O(n \cdot m)$ (using a full representation of the solutions, $\mathcal{R}^S_1$) and the calculation of the makespan has complexity $O(1)$, as $C_{i,\pi_{i,k,j}}$ and $\bar{C}_{i,\pi_{i,k,j}}$ are already known. So, the final complexity of this procedure is $O(n \cdot m)$ instead of $O(n^2 \cdot m)$ of the procedure without the accelerations, i.e. by using this procedure the complexity of the insertion of operation $O_{i\sigma}$ in every position of stage i is decreased by a complexity $O(n)$.

Once a procedure to accelerate the insertion of a job in every position of the stage has been defined, we repeat this procedure for several operations based on the following theorem:

Theorem 4.2. *Given an instance $\mathcal{I}$ of the $HFm||C_{max}$ problem and a schedule $\mathcal{S}$ represented by Π_{ik} with a makespan equals to C_{max}, then removing an operation of a stage i, not belonging to any critical path of $\mathcal{S}$, and re-inserting it in another position of that stage cannot decrease the actual makespan.*

Proof. The proof is obvious based on Theorem 3.1. As already known, the critical path is formed by operations which join without any idle

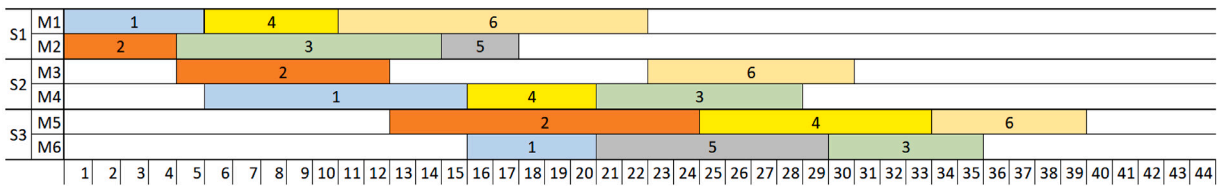

Fig. 6. Example of the completion times of a forward schedule in the speed-up procedure.

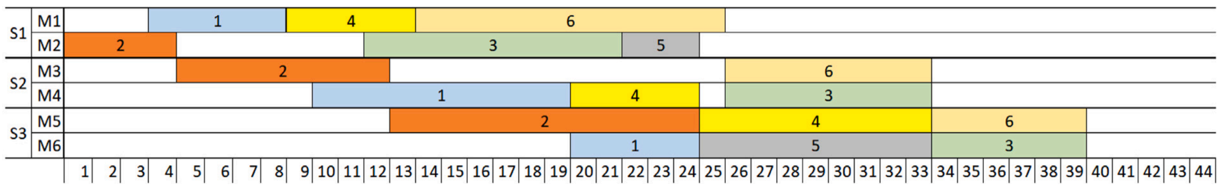

Fig. 7. Example of the completion times of a backward schedule in the speed-up procedure.

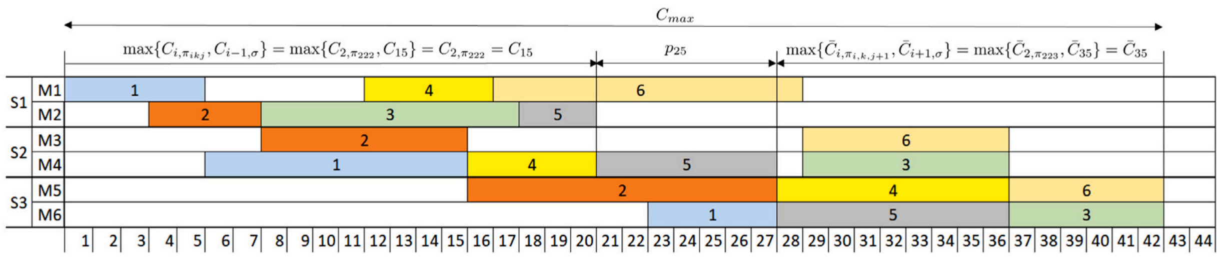

Fig. 8. Example of the calculation of the makespan using the speed-up procedure (operations are scheduled in backward manner).

or waiting time and cannot be compacted more. As a consequence, the sum of the processing times of these operations is the makespan of the solution. Hence, if both these operations cannot be compacted more because they are related to each other, and no operation of the critical path is removed or changed, then it is clear that the sum of the processing times of the critical path cannot be reduced and its lower bound is the previous makespan. □

With this in mind and taking into account that there could be several critical paths, we design a local search method (denoted as LSwS), selecting operations only in the critical set. These operations are removed, one by one, and re-inserted in the same stage using the previous speed-up method. This procedure is repeated until no more improvements are found. Using the previous example, operations $O_{1,2}$, $O_{1,3}$, $O_{1,5}$, $O_{2,2}$, $O_{2,5}$, $O_{3,2}$, $O_{3,3}$, $O_{3,4}$, $O_{3,5}$, and $O_{3,6}$ (see Fig. 3) should be removed from its corresponding stage and re-inserted in the best position and machine of that stage, applying the speed-up procedure. A detailed procedure of the insertion-based local search using these accelerations is detailed in Fig. 9. As the number of operations in the critical set ($|C|$) must be lower than or equal to $n \cdot m$ (total number of operations), the complexity of this local search method is $O(n^2 \cdot m^2)$, as compared to $O(n^3 \cdot m^2)$, which is the complexity of the same local search method without using the speed-up procedure.

5. Application in an iterated greedy algorithm

In this section, we incorporate the proposed acceleration in a local-search-based metaheuristic. More specifically, we consider a simple iterated greedy algorithm. This algorithm was originally proposed by Ruiz and Stützle (2007) (denoted as IG) and adapted to the problem under consideration by Kizilay et al. (2015). Basically it consists in carrying out a local search phase after a greedy diversification of the solution. In this diversification phase, d jobs are destructed (i.e. removed from the actual sequence) and then constructed in the sequence following a greedy procedure, where each destructed job is inserted one by one in the best position. After performing the diversification phase and the local search phase, a simple simulated annealing phase is carried out to decide the reference sequence of the next iteration. We have chosen such a simple metaheuristic due to the following two reasons. Firstly, our objective is to analyse the influence exclusively of the proposed speed-up mechanism in insertion-based local searches. In this regard, the iterated greedy algorithm is a variant of the iterated local search which is known to be one of the simplest metaheuristics for scheduling problems. Therefore we avoid using more complex or population-based metaheuristics, since some sophisticated phases could disturb our purpose. Secondly, although the iterated greedy algorithm was originally proposed for the permutation flowshop scheduling problem to minimise the makespan and despite its simplicity, it is one of the best performing metaheuristics in scheduling problem and in fact is state-of-the-art for several different problems (see e.g. Fanjul-Peyro & Ruiz, 2010; Fernandez-Viagas & Framinan, 2015a, 2019; Fernandez-Viagas, Valente et al., 2018; Huang et al., 2021; Mao et al., 2021; Zou et al., 2021).

The proposed metaheuristic, denoted as IGwS, introduces an intensive local search, using a full representation of the solution and the previous speed-up procedure, in a destruction/phase mechanism, which diversifies the solutions using a single sequence. To obtain a semi-active schedule from the single sequence, we applied the First Available Machine rule (FAM) and the First-In-First-Out rule (FIFO). According to Fernandez-Viagas and Framinan (2019), this representation and decoding procedure is denoted by $\mathcal{R}_4^F(FAM, FIFO)$. More specifically, the proposal is composed of the following phases (see pseudo code in Fig. 10):

- *Initial solution*: We use the traditional NEH algorithm, adapted for the problem by Brah and Loo (1999), as the seed sequence. Basically, following the LPT rule, jobs are inserted in the best position of an initial empty partial sequence. Note that, following the reasoning mentioned above, we have selected this initial algorithm instead of the best heuristics for the problem (JbH1, JbH2,

```
Procedure $LSwS(\Pi_{ik})$
    $improve = true$;
    while $improve$ is $true$ do
        $improve = false$;
        Calculate the operation in critical set $\mathcal{C}$. Let $|\mathcal{C}|$ denote the total number of operations in the set. In addition, let $\sigma(o)$ and $i(o)$ be the job and stage, respectively, corresponding to the $o$th operation in the set;
        for $o = 1$ to $|\mathcal{C}|$ do
            Remove the $o$th operation of the critical set $\mathcal{C}$ in $\Pi_{jk}$;
            Calculate the forward schedule (i.e. the completion time $C_{i,\pi_{i,k,j}}$ of each operation $\pi_{i,k,j}$);
            Calculate the backward schedule (i.e. the completion time $\bar{C}_{i,\pi_{i,k,j}}$ of each operation $\pi_{i,k,j}$) beginning with $i = m$ and $j = n_{ik}$ for any $k \in m_i$;
            for $k = 1$ to $m_{i(o)}$ do
                for $j = 1$ to $j \in [1, n_{i(o)k}]$ do
                    Calculate the makespan incurred by introducing job $\sigma(o)$ in position $j$ of machine $k$ (denoted as $C_{max}^{jk}$) by using Theorem 4.1: $C_{max}^{jk} = \max\{C_{max}, \max\{C_{i,\pi_{ikj}}, C_{i-1,\sigma}\} + p_{i,\sigma} + \max\{\bar{C}_{i,\pi_{i,k,j+1}}, \bar{C}_{i+1,\sigma}\}\}$
                end
            end
            Insert job $\sigma(o)$ in position $j$ in $\Pi_{jk}$, which minimises $C_{max}^{jk}$;
            $C'_{max} = \min_{\forall j,k}\{C_{max}^{jk}\}$;
            if $C'_{max} < C_{max}$ then
                $improve = true$;
                $C_{max} = C'_{max}$;
                Update critical set $\mathcal{C}$. Let $|\mathcal{C}|$ denote the total number of operations in the set. In addition, let $\sigma(o)$ and $i(o)$ be the job and stage of the $o$th operation in the set;
            end
        end
    end
end
```

Fig. 9. Insertion-based local search using the proposed speed-up procedure.

or MCH) to simplify the proposal and analyse only the influence of the new speed-up mechanism in the iterative procedure of the metaheuristics. After obtaining this seed sequence, it is improved by the original insertion-based local search method of the iterated greedy algorithm. One by one, this method removes all jobs in the sequence and reinserts them in the best position. This iterative procedure (denoted as IterativeImprovement_Insertion) is repeated until no more improvements are found. Let Π denote the sequence obtained after this local search method. During this initial procedure, we apply $\mathcal{R}_4^F(FAM, FIFO)$.

- *Destruction phase*: in each iteration, d jobs are randomly removed from the single sequence ($\mathcal{R}_4^F$). Let Π^D denote the jobs randomly removed, and Π^R be the remaining jobs from Π after removing jobs Π^D.
- *Construction phase*: following a greedy procedure, the jobs in Π^D are, one by one, re-inserted in the best position of sequence Π^R. Let Π^C denote the sequence after this construction phase. Once a single sequence is obtained (using $\mathcal{R}_4^F$), we apply the FIFO and FAM rules to obtain a full sequence (i.e. $\mathcal{R}_1^S$), denoted by Π_{ik}^C.
- *Local search method using the proposed speed-up procedure (LSwS)*: the proposed local search explained in the previous section (see Fig. 9) is applied to Π_{ik}^C. Let us denote by Π'_{ik} the obtained sequence.
- *Acceptance criterion*: A simple simulated annealing criterion is applied with a constant parameter denoted as $Temperature$. Π^C is then used as the reference sequence in the following iteration (i.e. $\Pi = \Pi^C$) if either the solution of the previous local search has been improved in this iteration, or if a random number between 0 and 1 is lower than:

$$\exp\{-(C_{max}(\Pi_{ik}^C) - C_{max}(\Pi)) \cdot \frac{n \cdot m \cdot 10}{T \cdot \sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij}}\}$$

```
Procedure $IGwS(d, T)$
    $\Pi_{NEH} := NEH(LPT)$;

    $\Pi := IterativeImprovement\_Insertion(\Pi_{NEH})$;

    while stopping criterion is not met do
        // destruction phase
        Randomly remove $d$ jobs from $\Pi$;
        (let $\Pi^R$ be the remaining sequence and $\Pi^D$ the extracted jobs)

        // construction phase
        $\Pi^C := Construction(\Pi^D, \Pi^R)$;
        $\Pi^C_{ik}$:= full solution obtained after applying the FIFO and FAM rules to $\Pi^C$;
        // local search
        $\Pi'_{ik} := LSwS(\Pi^C_{ik})$;

        // acceptance criterion
        $\Pi := AcceptanceCriterion(\Pi^C, T)$;
    end
end
```

Fig. 10. $IGwS$.

Table 2
ARPD values of implemented algorithms.

n	m	$\lambda = 0.003$							$\lambda = 0.006$						
		IG	DABC	CSA	IGT	IGT_{ALL}	VBIH	IGwS	IG	DABC	CSA	IGT	IGT_{ALL}	VBIH	IGwS
40	5	1.46	1.65	3.99	1.86	2.07	2.04	1.20	0.92	1.23	3.74	1.47	1.39	1.46	0.87
40	10	2.36	2.57	5.63	3.12	3.27	3.51	2.02	1.71	2.14	5.64	2.54	2.72	2.64	1.48
40	15	3.41	3.27	6.45	4.11	4.39	4.68	3.18	2.51	2.68	6.05	3.42	3.62	3.58	2.52
40	20	3.34	3.69	7.47	4.23	4.57	4.54	3.56	2.55	2.99	6.88	3.40	3.85	3.61	2.68
80	5	0.09	0.05	−0.31	0.10	0.08	0.10	0.05	0.05	0.04	−0.71	0.04	0.04	0.04	0.03
80	10	3.25	3.29	7.66	5.46	5.24	5.36	2.44	2.40	2.64	7.36	3.57	3.96	3.83	1.85
80	15	3.25	2.87	6.63	4.77	4.99	4.96	2.65	2.55	2.44	6.55	3.32	3.77	3.42	2.11
80	20	4.21	3.62	7.42	5.39	5.59	5.60	3.40	3.10	2.97	7.47	3.67	4.28	4.10	2.75
120	5	0.07	0.03	0.16	0.07	0.07	0.07	0.01	0.02	0.02	0.20	0.02	0.02	0.02	0.00
120	10	0.68	0.48	1.82	1.29	1.30	1.33	0.31	0.34	0.40	1.60	0.59	0.67	0.65	0.18
120	15	4.05	3.73	7.80	6.47	6.47	6.50	2.87	3.24	3.20	7.49	4.43	4.93	4.82	2.32
120	20	4.71	3.84	7.93	6.13	6.22	6.16	3.67	3.54	3.55	7.45	4.50	4.77	4.84	2.75
160	5	0.06	0.02	−0.10	0.06	0.06	0.06	0.02	0.03	0.02	0.05	0.03	0.02	0.03	0.01
160	10	0.32	0.15	0.98	0.35	0.36	0.36	0.05	0.13	0.10	1.06	0.25	0.25	0.22	0.02
160	15	4.24	3.33	7.35	5.94	5.96	5.98	2.85	3.35	2.98	7.68	5.86	5.75	5.76	2.16
160	20	4.91	3.75	7.79	5.95	5.92	5.97	3.52	3.79	3.34	7.39	5.91	5.86	5.77	2.78
200	5	0.07	0.01	−0.11	0.05	0.05	0.05	0.03	0.05	0.01	−0.32	0.04	0.05	0.05	0.02
200	10	0.18	0.14	0.59	0.18	0.18	0.18	0.07	0.12	0.12	0.55	0.18	0.18	0.17	0.04
200	15	1.16	0.73	2.30	1.21	1.19	1.22	0.55	0.76	0.63	2.31	1.19	1.21	1.20	0.35
200	20	4.50	3.43	7.21	5.63	5.63	5.63	3.19	3.47	3.09	7.13	5.59	5.58	5.62	2.60
240	5	0.05	0.01	−2.19	0.01	0.01	0.01	0.01	0.03	0.00	−2.16	0.00	0.01	0.01	0.00
240	10	0.20	0.11	0.43	0.18	0.18	0.17	0.07	0.12	0.10	0.36	0.17	0.17	0.17	0.03
240	15	0.41	0.29	1.22	0.42	0.43	0.46	0.10	0.23	0.23	1.07	0.45	0.42	0.42	0.04
240	20	4.77	3.81	8.05	5.83	5.89	5.88	3.47	3.83	3.49	7.92	5.84	5.85	5.82	2.76
Average		2.16	1.87	4.01	2.87	2.92	2.95	1.64	1.62	1.60	3.87	2.35	2.47	2.43	1.26

Table 3
Non-parametric Mann–Whitney tests.

Hypothesis	Sample Size	ARPD(Mean)		ARPD(Median)		Mann–Whitney U	Significance
		IGwS	DABC	IGwS	DABC		
IGwS = DABC ($\lambda = 0.003$)	480	1.64	1.85	1.40	1.76	32198.00	0.025
IGwS = DABC ($\lambda = 0.006$)	480	1.27	1.62	1.13	1.49	34383.00	0.000

6. Computational evaluation

In this section, we analyse the performance of the proposed speed-up procedure when it is incorporated in a simple iterated greedy algorithm, IGwS. In order to check the efficiency of the proposal, we first compare the following metaheuristics (according to Section 1):

- The simple iterated greedy algorithm without speed-up procedure (IG) adapted by Kizilay et al. (2015).
- The DABC metaheuristic proposed by Pan et al. (2014).
- The CSA metaheuristic proposed by Lin et al. (2021).
- The IGT metaheuristic proposed by Öztop et al. (2020) for the related $HFm||\sum C_j$ problem.
- The IGT$_{\text{ALL}}$ metaheuristic proposed by Öztop et al. (2020) for the related $HFm||\sum C_j$ problem.
- The VBIH metaheuristic proposed by Öztop et al. (2020) for the related $HFm||\sum C_j$ problem.
- The proposed IGwS metaheuristic.

All these metaheuristics are again reimplemented and compared under the same conditions:

- Using the same computer and operating system (Microsoft Windows 8.1 64 bit, in an Intel Core i7-3770 with 3.4 GHz, 16 GB RAM).
- The same person has re-coded every comparison algorithm using the same common functions and libraries.
- Using the same programming language (C# under Visual Studio 2019).
- Using the same set of instances. In this regard, we use the recent set of big-size instances proposed by Fernandez-Viagas and Framinan (2020). This set is composed of 240 instances varying $n \in \{40, 80, 120, 160, 200, 240\}$ and $m \in \{5, 10, 15, 20\}$ and with 10 instances for each combination of the parameters. The values of processing times are generated using two different uniform distributions: $U(1, 99)$ and $U(1, 40 \cdot m_i)$.
- Using the same stopping criteria for all methods. More specifically, the algorithms are stopped by two different time limits, which depend on the size of the instances according to $\lambda \cdot n^3 \cdot m$ milliseconds with $\lambda \in \{0.003, 0.006\}$ (see Fernandez-Viagas & Framinan, 2020 for similar approaches).
- Using the same parameters than the ones chosen in their original corresponding contributions. In this regard, the proposed IGwS has then not been recalibrated and assumes the original values for the parameters (i.e. $d = 4$ and $T = 0.4$) of the traditional IG, which is the worst case for the proposal.

In addition, each metaheuristic is run 10 times and the average values are stored. Finally, the algorithms are compared using the Average Relative Percentage Deviation (ARPD), defined by the following equation for the metaheuristic h (with $h \in \{IG(\lambda = 0.003), IGwS(\lambda = 0.003), DABC(\lambda = 0.003), IG(\lambda = 0.006), IGwS(\lambda = 0.006), DABC(\lambda = 0.006)\}$ and being UB the upper bound obtained in Fernandez-Viagas & Framinan, 2020):

$$ARPD_h = \frac{1}{I}\sum_{l=1}^{I}\frac{C_{max}^{lh} - UB}{UB} \cdot 100, \forall\, h = 1, \dots, H \tag{12}$$

Computational results in terms of ARPD are shown in Table 2, while in Fig. 11 we show the evolution of the solutions obtained by each algorithm (from the initial solution to $\lambda = 0.006$). The best result is found by the new proposal IGwS, with ARPD equals to 1.64 for $\lambda = 0.003$. For the same stopping criterion, the ARPD value of the second best metaheuristic (i.e. DABC) is 1.87 and the value obtained by the same iterated greedy without speed-up procedure (i.e. IG) is 2.16. Similarly, using the stopping criterion $\lambda = 0.006$, the proposed heuristic found the best result in term of ARPD, 1.26. In this case, the ARPD values found by DABC and IG are 1.60 and 1.62, respectively. In order to establish the proposed procedure as a state-of-the-art metaheuristic for the problem, a non-parametric Mann–Whitney test is executed. Results are shown in Table 3. In both cases $\lambda = 0.003$ and $\lambda = 0.006$, the hypothesis $IGwS = DABC$ is rejected with significance 0.017 and 0.000, respectively. A similar result is found if the hypothesis $IGwS = IG$ is checked, which is rejected using the same test with a significance of 0.000, regardless the stopping criterion. Regarding the evolution of the performance of the metaheuristics for different CPU times, we observe in Fig. 11 the excellent convergence found by the proposed algorithm. Thereby, e.g. its ARPD value decreases from 1.36 (for $\lambda = 0.005$) to 1.26 (for $\lambda = 0.006$), as compared with 1.66 (for $\lambda = 0.005$) and 1.60 (for $\lambda = 0.006$) obtained by DABC.

Finally, a last experimentation is carried out analysing the proposed local search LSwS. In this experimentation, we compare the performance of LSwS with the traditional insertion local search on $\mathcal{R}_1^S$, which removes and reinserts (one by one) every operation of a stage in the best position of that stage. Both local search methods are initialised with the same initial sequence (LPT rule) and tested on the same set of instances. Computational results are shown in Table 4. In average, LS and LSwS need 12.14 and 0.01 s, and obtain an ARPD of 12.12 and 12.18, respectively. More specifically, maintaining approximately the same quality of solution (this hypothesis cannot be rejected using a Mann–Whitney non-parametric test with p-value equals to 0.952), the traditional local search method LS needs 1,212.33 times more CPU time than the proposed local search LSwS.

7. Conclusions

In this paper, we have tackled the traditional hybrid flow shop with makespan minimisation and without additional constraints. For this problem, some specific definitions are first established with respect to the construction of a schedule and the critical operations. Equipped with these concepts, we propose several theoretical results (five theorems and a corollary) to contribute to a deeper understanding of this problem. All these theoretical results are considered to propose a speed-up procedure for the problem, which can be applied using a complete representation of the solutions. Finally, a simple iterated-greedy metaheuristic is proposed including this speed-up procedure.

Under the same computer conditions, the proposed metaheuristic is compared, in an extensive computational evaluation, with the most promising metaheuristics developed either for the problem under consideration or related scheduling problems (including, among others, the DABC algorithm, and the same iterated greedy metaheuristic without the speed-up procedure). The results show the excellent performance of the proposed procedure, which statistically outperforms every other metaheuristic. More specifically, the results have shown that embedding the proposed speed-up procedure in a very simple metaheuristic (without specific knowledge of the problem) has been enough to obtain the state-of-the-art metaheuristic for the problem under consideration. As a consequence, further research lines could come in developing more efficient and advance metaheuristics successfully adapting the here proposed speed-up procedure.

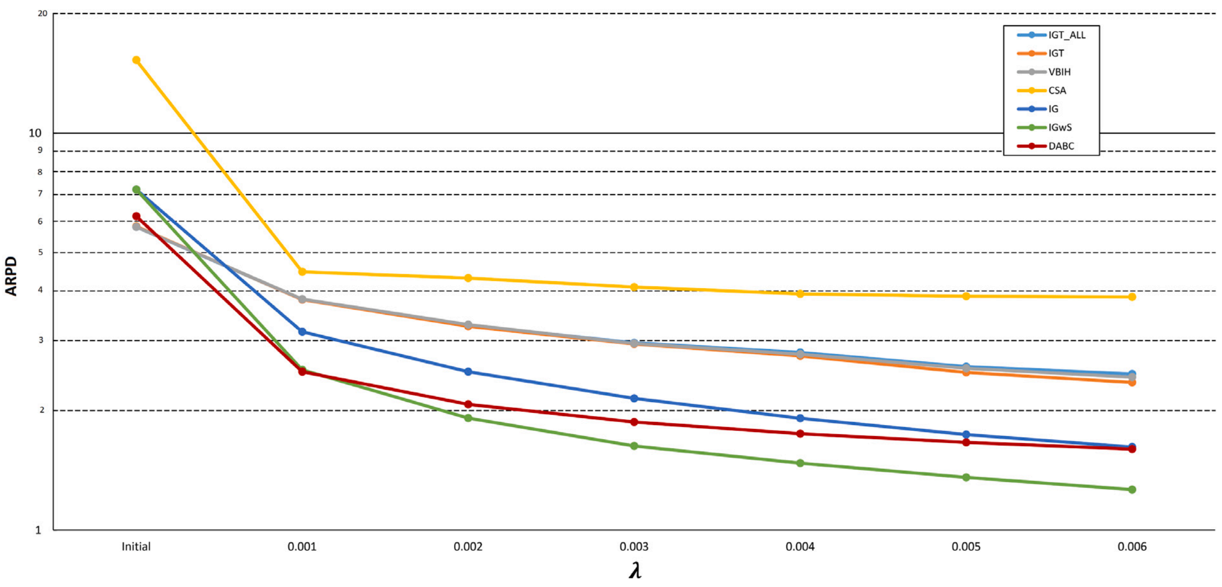

Fig. 11. Evolution curve of the implemented algorithms. Y-axis is shown in logarithmic scale.

Table 4
Comparison between an extensive insertion-based local search and the proposed local search with speed-up procedure.

n	m	$ARPD$		$CPU times$	
		LS	LSwS	LS	LSwS
40	5	23.31	23.91	0.03	0.00
40	10	22.11	22.29	0.12	0.00
40	15	23.08	23.22	0.22	0.00
40	20	23.04	23.09	0.38	0.00
80	5	8.89	8.96	0.20	0.00
80	10	24.06	24.26	0.75	0.00
80	15	20.50	20.49	1.68	0.01
80	20	20.38	20.32	3.06	0.01
120	5	5.50	5.43	0.66	0.00
120	10	10.63	10.66	2.47	0.01
120	15	22.14	22.27	5.19	0.01
120	20	18.93	19.12	9.41	0.01
160	5	5.06	5.03	1.42	0.00
160	10	7.16	7.15	5.43	0.01
160	15	18.72	18.62	12.35	0.01
160	20	18.45	18.49	22.68	0.02
200	5	3.24	3.22	2.76	0.01
200	10	5.47	5.45	10.82	0.01
200	15	9.23	9.14	24.64	0.02
200	20	17.24	17.28	44.30	0.03
240	5	2.86	2.85	4.77	0.01
240	10	4.95	4.96	18.67	0.02
240	15	6.60	6.60	42.57	0.03
240	20	17.37	17.40	76.70	0.03
Average		14.12	14.18	12.14	0.01

CRediT authorship contribution statement

Victor Fernandez-Viagas: Investigation, Software, Formal analysis, Writing – original draft, Writing – review & editing, Visualization, Methodology, Data curation, Conceptualization, Supervision, Validation, Resources, Project administration.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This research has been funded by the Spanish Ministry of Science and Innovation, under the project “ASSORT” with reference PID2019-108756RB-I00, and by the Junta de Andalucia, Spain under the project “EFECTOS”, with reference US-1264511.

References

Acero-Dominguez, M., & Paternina-Arboleda, C. (2004). Scheduling jobs on a k-stage flexible flow shop using a TOC-based (bottleneck) procedure. In *2004 IEEE systems and information engineering design symposium* (pp. 295–298).

Alaykýran, K., Engin, O., & Döyen, A. (2007). Using ant colony optimization to solve hybrid flow shop scheduling problems. *International Journal of Advanced Manufacturing Technology*, *35*(5–6), 541–550.

Bozorgirad, M., & Logendran, R. (2016). A comparison of local search algorithms with population-based algorithms in hybrid flow shop scheduling problems with realistic characteristics. *International Journal of Advanced Manufacturing Technology*, *83*(5–8), 1135–1151.

Brah, S., & Loo, L. (1999). Heuristics for scheduling in a flow shop with multiple processors. *European Journal of Operational Research*, *113*(1), 113–122.

Engin, O., & Döyen, A. (2004). A new approach to solve hybrid flow shop scheduling problems by artificial immune system. *Future Generation Computer Systems*, *20*(6 Spec. Iss.), 1083–1095.

Fanjul-Peyro, L., & Ruiz, R. (2010). Iterated greedy local search methods for unrelated parallel machine scheduling. *European Journal of Operational Research*, *207*(1), 55–69.

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

Fernandez-Viagas, V., & 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*, 86–96.

Fernandez-Viagas, V., & Framinan, J. (2019). A best-of-breed iterated greedy for the permutation flowshop scheduling problem with makespan objective. *Computers & Operations Research*, *112*.

Fernandez-Viagas, V., & Framinan, J. (2020). Design of a testbed for hybrid flow shop scheduling with identical machines. *Computers & Industrial Engineering*, *141*.

Fernandez-Viagas, V., Molina-Pariente, J. M., & Framinan, J. M. (2018). New efficient constructive heuristics for the hybrid flowshop to minimise makespan: A computational evaluation of heuristics. *Expert Systems with Applications*, *114*, 345–356.

Fernandez-Viagas, V., Molina-Pariente, J., & Framinan, J. (2020). Generalised accelerations for insertion-based heuristics in permutation flowshop scheduling. *European Journal of Operational Research*, *282*(3), 858–872.

Fernandez-Viagas, V., Perez-Gonzalez, P., & Framinan, J. (2019). Efficiency of the solution representations for the hybrid flow shop scheduling problem with makespan objective. *Computers & Operations Research*, *109*, 77–88.

Fernandez-Viagas, V., Ruiz, R., & 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., & 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.

Grabowski, J., & Wodecki, M. (2004). A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion. *Computers & Operations Research*, *31*(11), 1891–1909.

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

Gupta, J. (1988). Two-stage, hybrid flowshop scheduling problem. *Journal of the Operational Research Society*, *39*(4), 359–3641.

Huang, Y.-Y., Pan, Q.-K., Huang, J.-P., Suganthan, P., & Gao, L. (2021). An improved iterated greedy algorithm for the distributed assembly permutation flowshop scheduling problem. *Computers & Industrial Engineering*, *152*.

Kizilay, D., Tasgetiren, M., Pan, Q.-K., & Wang, L. (2015). An iterated greedy algorithm for the hybrid flowshop problem with makespan criterion. In *IEEE symposium series on computational intelligence - CIPLS 2014, proceedings* (pp. 16–23).

Liao, C.-J., Tjandradjaja, E., & Chung, T.-P. (2012). An approach using particle swarm optimization and bottleneck heuristic to solve hybrid flow shop scheduling problem. *Applied Soft Computing*, *12*(6), 1755–1764.

Lin, S.-W., Cheng, C.-Y., Pourhejazy, P., Ying, K.-C., & Lee, C.-H. (2021). New benchmark algorithm for hybrid flowshop scheduling with identical machines. *Expert Systems with Applications*, *183*.

Long, J., Zheng, Z., Gao, X., & Pardalos, P. M. (2018). Scheduling a realistic hybrid flow shop with stage skipping and adjustable processing time in steel plants. *Applied Soft Computing*, *64*, 536–549.

Mao, J.-Y., Pan, Q.-K., Miao, Z.-H., & Gao, L. (2021). An effective multi-start iterated greedy algorithm to minimize makespan for the distributed permutation flowshop scheduling problem with preventive maintenance. *Expert Systems with Applications*, *169*.

Naderi, B., Gohari, S., & Yazdani, M. (2014). Hybrid flexible flowshop problems: Models and solution methods. *Applied Mathematical Modelling*, *38*(24), 5767–5780.

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

Naderi, B., Ruiz, R., & Zandieh, M. (2010). Algorithms for a realistic variant of flowshop scheduling. *Computers & Operations Research*, *37*(2), 236–246.

Negenman, E. (2001). Local search algorithms for the multiprocessor flow shop scheduling problem. *European Journal of Operational Research*, *128*(1), 147–158.

Niu, Q., Zhou, T., & Ma, S. (2009). A quantum-inspired immune algorithm for hybrid flow shop with makespan criterion. *Journal of Universal Computer Science*, *15*(4), 765–785.

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

Nowicki, E., & Smutnicki, C. (1998). The flow shop with parallel machines: A tabu search approach. *European Journal of Operational Research*, *106*(2-3), 226–253.

Nowicki, E., & Smutnicki, C. (2006). Some aspects of scatter search in the flow-shop problem. *European Journal of Operational Research*, *169*(2), 654–666.

Öztop, H., Tasgetiren, M. F., Eliiyi, D. T., Pan, Q.-K., & Kandiller, L. (2020). An energy-efficient permutation flowshop scheduling problem. *Expert Systems with Applications*, *150*, Article 113279.

Pan, Q.-K. (2016). An effective co-evolutionary artificial bee colony algorithm for steelmaking-continuous casting scheduling. *European Journal of Operational Research*, *250*(3), 702–714.

Pan, Q.-K., Gao, L., Li, X.-Y., & Gao, K.-Z. (2017). Effective metaheuristics for scheduling a hybrid flowshop with sequence-dependent setup times. *Applied Mathematics and Computation*, *303*, 89–112.

Pan, Q.-K., Wang, L., Li, J.-Q., & Duan, J.-H. (2014). A novel discrete artificial bee colony algorithm for the hybrid flowshop scheduling problem with makespan minimisation. *Omega (United Kingdom)*, *45*, 42–56.

Peng, K., Pan, Q.-K., Gao, L., Zhang, B., & Pang, X. (2018). An improved artificial bee colony algorithm for real-world hybrid flowshop rescheduling in steelmaking-refining-continuous casting process. *Computers & Industrial Engineering*, *122*, 235–250.

Pinedo, M. (1995). *Scheduling: Theory, algorithms and systems*. Prentice Hall.

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

Ribas, I., Companys, R., & Tort-Martorell, X. (2013). A competitive variable neighbourhood search algorithm for the blocking flow shop problem. *European Journal of Industrial Engineering*, *7*(6), 729–754.

Ribas, I., Leisten, R., & Framinan, J. (2010). Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective. *Computers & Operations Research*, *37*(8), 1439–1454.

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

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

Ruiz, R., & Vázquez-Rodríguez, J. (2010). The hybrid flow shop scheduling problem. *European Journal of Operational Research*, *205*(1), 1–18.

Santos, D., Hunsucker, J., & Deal, D. (1996). An evaluation of sequencing heuristics in flow shops with multiple processors. *Computers & Industrial Engineering*, *30*(4), 681–692.

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

Urlings, T., Ruiz, R., & Stützle, T. (2010). Shifting representation search for hybrid flexible flowline problems. *European Journal of Operational Research*, *207*(2), 1086–1095.

Wang, S.-Y., Wang, L., Liu, M., & Xu, Y. (2013). An enhanced estimation of distribution algorithm for solving hybrid flow-shop scheduling problem with identical parallel machines. *International Journal of Advanced Manufacturing Technology*, *68*(9–12), 2043–2056.

Xu, Y., Wang, L., Wang, S., & Liu, M. (2013). An effective shuffled frog-leaping algorithm for solving the hybrid flow-shop scheduling problem with identical parallel machines. *Engineering Optimization*, *45*(12), 1409–1430.

Zhang, B., Pan, Q.-K., Gao, L., Li, X.-Y., Meng, L.-L., & Peng, K.-K. (2019). A multiobjective evolutionary algorithm based on decomposition for hybrid flowshop green scheduling problem. *Computers & Industrial Engineering*, *136*, 325–344.

Zou, W.-Q., Pan, Q.-K., & Tasgetiren, M. (2021). An effective iterated greedy algorithm for solving a multi-compartment AGV scheduling problem in a matrix manufacturing workshop. *Applied Soft Computing*, *99*.