European Journal of Operational Research 299 (2022) 869–882

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Discrete Optimization

Assembly flowshop scheduling problem: Speed-up procedure and computational evaluation

Victor Fernandez-Viagas[a,*], Carla Talens[a], Jose M. Framinan[a,b]

[a] Industrial Management, School of Engineering, University of Seville, Camino de los Descubrimientos s/n, Seville 41092, Spain
[b] Laboratory of Engineering for Environmental Sustainability, University of Seville, Camino de los Descubrimientos s/n, Seville 41092, Spain

ARTICLE INFO

Article history:
Received 3 February 2021
Accepted 3 October 2021
Available online 9 October 2021

Keywords:
Scheduling
Two-stage assembly
Flow shop
Three-stage assembly
Accelerations
Speed-up procedure
Heuristics
Makespan
Computational evaluation
ASP
Multistage assembly

ABSTRACT

In this paper, we address the assembly flowshop scheduling problem, which is a generalisation of two well-known scheduling problems in the literature: the three-stage Assembly Scheduling Problem (ASP) and its variant with two stages denoted as the two-stage ASP. For this problem, we prove several theoretical results which are used to propose a speed-up procedure. This acceleration mechanism can be applied in any insertion-based method for the problem under study and, consequently, also for their special cases. In addition, we propose four efficient constructive heuristics for the problem, based on both Johnson's algorithm and the NEH heuristic. These proposals are compared against 47 algorithms existing in the literature for related problems. The results show the excellent performance of the proposals.

1. Introduction

Assembly scheduling is the branch of the scheduling theory dealing with scenarios where a set of operations must have been completed before the next operation can be initiated. Assembly scheduling is present in many manufacturing sectors, including computer manufacturing, car and motor industries, or plastic industries, among others (see e.g. Allahverdi & Aydilek, 2015; Fattahi, Hassan Hosseini, & Jolai, 2013; Hwang & Lin, 2012; Liao, Lee, & Lee, 2015; Zhang, Zhou, & Liu, 2010 or Sheikh, Komaki, & Kayvanfar, 2018). Its importance for today's manufacturing scenarios has been thus perceived by academics and practitioners, with a high number of contributions produced in the last decades (see Framinan, Perez-Gonzalez, & Fernandez-Viagas, 2019 for a review on this topic). However, as this review attests, most contributions do not consider scheduling decisions for the operations taking place after the assembly, therefore important issues such as the transportation of the assembled product, or scheduling in more complex assembly layouts are seldom addressed

In this paper, we focus on the generic assembly scheduling problem with a first phase, where the components of the product are manufactured in several dedicated machines, and a second phase composed of several operations in series (i.e. adopting a flowshop after the manufacturing of the components of the product), which can model assembly stages with collection, transportation, and/or common operations. This decision problem is labelled in the following as Multi-stage Assembly Scheduling Problem (MASP). The scheduling criterion considered is the minimisation of the maximum completion time of the jobs, or makespan, which is a widespread objective aimed at maximizing machines' utilisation. Furthermore, we assume that the jobs are processed in all machines of the second phase in the same order (permutation constraint). According to the notation proposed in Framinan et al. (2019), this problem can be denoted as $DPm \rightarrow Fm|prmu|C_{\max}$.

The MASP can be seen as a generalization of two well-known assembly scheduling problems: On the one hand, if the number of machines in the second phase is one, MASP can be reduced to the so-called Two-stage Assembly Scheduling Problem (2ASP) or $DP_m \rightarrow 1||C_{\max}$ problem, first introduced by Lee, Cheng, & Lin (1993). On the other hand, if the number of stages in the second phase is two with one machine in each stage, it can be reduced to the three-stage ASP (Koulamas & Kyparisis, 2001), denoted as

* Corresponding author.
E-mail address: vfernandezviagas@us.es (V. Fernandez-Viagas).

https://doi.org/10.1016/j.ejor.2021.10.001

3ASP. In addition, note that, if no assembly is considered (and consequently, the number of dedicated machines in the first stage is one), the problem can be reduced to the well-known Permutation Flowshop Scheduling Problem (FPSP) (Johnson, 1954). Given that the 2ASP is NP-hard for two or more machines in the first phase (Lee et al., 1993), the MASP is also NP-hard when either there are at least two dedicated machines in the first stage, or more than one machine in the second phase. It is thus not surprising that most contributions on the MASP are devoted to proposing approximate algorithms capable of providing good (but not necessarily optimal) solutions with a reasonable computational effort. Since most of these algorithms are based on conducting some type of local search, their ability to quickly assess the quality of the solutions found is a crucial aspect for their efficiency, as in this manner they may explore a large portion of the solution space within a limited computation time. Such speed-up procedures (also known as accelerations) use specific properties of the scheduling problem and therefore they cannot be translated to other problems. In this regard, the pioneering speed-up procedure by Taillard (1990) for the $Fm|prmu|C_{\max}$ problem cannot be applied even for the same layout with different objectives and/or constraints, and, along the years, different speed-ups have been proposed for several scheduling problems (see e.g. Naderi & Ruiz, 2010; Nowicki, 1999; Nowicki & Smutnicki, 1998; Rios-Mercado & Bard, 1998, or Fernandez-Viagas, Molina-Pariente, & Framinan, 2020). However, to the best of our knowledge, no speed-up procedure has been proposed for the MASP problem.

In this paper, we propose a speed-up procedure for the MASP that reduces the complexity of insertion-based local search methods from $O(n^2 \cdot m)$ to $O(n \cdot m)$. Using this acceleration and some theoretical results found for the problem, we propose four new constructive heuristics and compare them against the most efficient so-far heuristics for the problem, as well as against other state-of-the-art heuristics from related scheduling problem. The remainder of the paper is organised as follows: in Section 2 we describe the problem under consideration and analyse the related literature. Some theoretical results required are introduced in Section 3. The speed-up procedure proposed is presented in Section 4, while the constructive heuristics are detailed in Section 5. The computational evaluation is carried out in Section 6 and the conclusions are discussed in Section 7.

2. Problem description and background

In the problem under study, there are n jobs to be scheduled. The jobs consist first on the manufacturing of m_1 components (first phase/stage, denoted as pre-assembly phase), each one carried out on a dedicated machine. These components are subsequently assembled in an assembly phase, thus forming a single unit. This assembly phase, where the single unit is assembled or processed, is composed of a number of serial operations, each one performed in a specific machine. Therefore, the assembly and the previous/subsequent operations (second phase composed of several stages) can be modelled as a flowshop consisting of m_2 machines. It is assumed that the sequence in which the jobs are processed in the flowshop remains the same across all m_2 machines. The processing time of job $j \in \{1, \dots, n\}$ on each machine i is denoted by p_{ij}, with $i \in \{1, \dots, m\}$ $(m = m_1 + m_2)$. Note that the first m_1 processing times correspond to each one of the components of the job, while $p_{m_1+k,j}$ with $k \in \{1, \dots, m_2\}$ corresponds to the processing times of the subsequent operations carried out in the assembly phase. The operation i of job j is denoted $O_{i,j}$. The objective of the scheduling decision problem is to minimize the maximum completion time of the jobs or makespan. The problem under consideration is illustrated by the example in Fig. 1 with five jobs and machines. Each job is composed of two components which are processed on a pre-assembly phase with two dedicated machines (i.e. each machine processing a different one). Once both are completed, they are grouped into a single unit, which is processed in a 3-machine flowshop.

In this setting, a feasible semiactive schedule (see definition in Pinedo, 2012) can be defined by giving the order in which the jobs or components have to be processed in both phases, i.e. $\Pi = (\pi_1, \dots, \pi_k, \dots, \pi_n)$ a sequence of the jobs. In principle, a different order of the jobs could be employed for each stage, however, Potts, Sevast'janov, Strusevich, Van Wassenhove, & Zwaneveld (1995) show that an optimal solution of the 2ASP can be obtained considering only sequences and, given the permutation constraint imposed in the second phase, the sequence resulting from the assembly cannot be altered in the subsequent operations. Therefore, the problem can be expressed as finding a sequence Π with minimal makespan $C_{\max} = \max_{j \in \{1,\dots,n\}} C_{mj}$, being C_{ij} the completion time of job j on machine i. These completion times can be recursively computed (from left to right) using the following equations:

$$C_{i,\pi_k} = \sum_{l=1}^{k} p_{i,\pi_l}, \quad i \in \{1, \dots, m_1\};\ k \in \{1, \dots, n\}. \tag{1}$$

$$C_{m_1+1,\pi_k} = \max\{\max_{i \le m_1}\{C_{i,\pi_k}\}, C_{m_1+1,\pi_{k-1}}\} + p_{m_1+1,\pi_k}, \quad k \in \{1, \dots, n\}. \tag{2}$$

$$C_{m_1+i,\pi_1} = C_{m_1+i-1,\pi_1} + p_{m_1+1,\pi_1}, \quad i \in \{2, \dots, m_2\}. \tag{3}$$

$$C_{m_1+i,\pi_k} = \max\{C_{m_1+i-1,\pi_k}, C_{m_1+i,\pi_{k-1}}\} + p_{m_1+i,\pi_k}, \quad i \in \{2, \dots, m_2\};\ k \in \{2, \dots, n\}. \tag{4}$$

Note that Eq. (1) computes the completion times in the first stage, whereas Eq. (2) determines the completion times on the assembly machine and Eqs. (3) and (4) compute the completion times in the rest of the machines of the second phase. This procedure to obtain a semiactive schedule is usually denoted as *Forward Codification*. In contrast, the inverse procedure to construct a schedule (from right to left) using the inverse sequence $\overline{\Pi} = (\overline{\pi}_1, \dots, \overline{\pi}_n) = (\pi_n, \dots, \pi_1)$ is denoted as *Backward Codification* (for previous uses of this codification in the literature, we refer e.g. to Ribas, Companys, & Tort-Martorell, 2010; Ribas, Companys, & Tort-Martorell, 2013 for the traditional flow shop, or Pan, Wang, Li, & Duan, 2014; Wang, Wang, Liu, & Xu, 2013 for the hybrid variant of the flow shop). Using this procedure, completion times are computed according to Eq. (5) in the assembly phase and Eq. (6) in the pre-assembly phase.

$$\overline{C}_{i,\pi_k} = \max\{\overline{C}_{i+1,\pi_k}, \overline{C}_{i,\pi_{k+1}}\} + p_{i,\pi_k}, i \in \{m, \dots, m_1 + 1\}, k \in \{n, \dots, 1\} \tag{5}$$

$$\overline{C}_{i\pi_k} = \max\{\overline{C}_{m_1+1,\pi_k}, \overline{C}_{i,\pi_{k+1}}\} + p_{i,\pi_k}, i \in \{m_1, \dots, 1\}, k \in \{n, \dots, 1\} \tag{6}$$

with $\overline{C}_{i,\pi_{n+1}} = 0$, $i \in \{1, \dots, m\}$, and $\overline{C}_{m+1,\pi_k} = 0$, $k \in \{1, \dots, n\}$.

With respect to the previous literature on the topic, in view of the relationship of the problem under consideration with other assembly scheduling problems, it is worth also to analyse the existing solution procedures for related problems, namely the two-stage ASP with a single machine in the assembly stage ($DPm \to 1$), the customer order scheduling problem ($DPm \to 0$), the two-stage ASP with parallel machines in the assembly stage ($DPm \to Pm$) and, finally, the three-stage assembly flow shop scheduling problem ($DPm \to F2$).

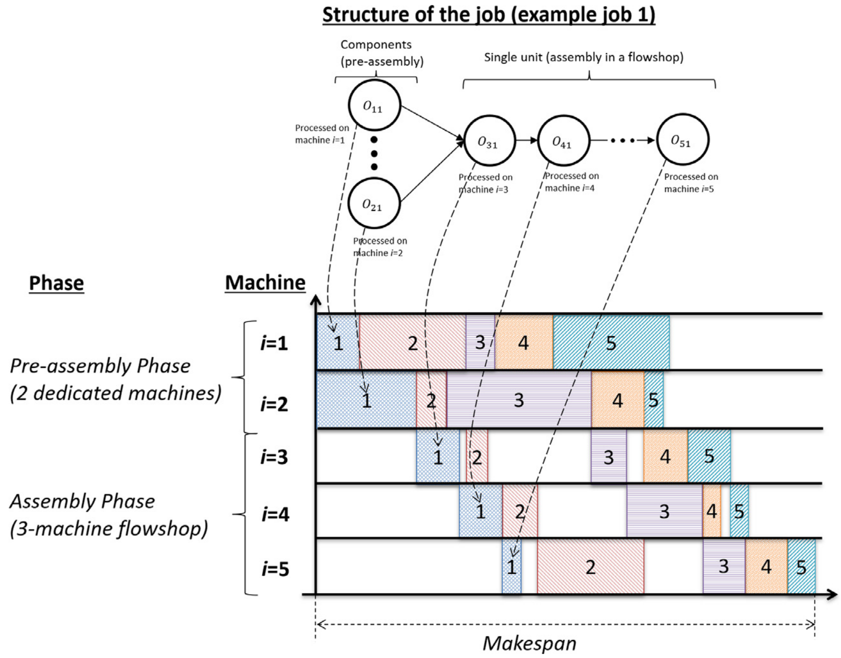

Fig. 1. Example of the problem under consideration with five jobs and machines.

Regarding the $DPm \rightarrow 1$ layout, several heuristics have been proposed to solve the problem with the objective of minimising the makespan ($DPm \rightarrow 1||C_{\max}$). Lee et al. (1993) address the problem with two machines in the first phase and an assembly machine in the second phase and shown that it is strongly NP-hard. These authors develop a Branch and Bound (B&B) procedure for the problem and proposed three heuristics, labelled LCL_1, LCL_2 and LCL_3, based on the characteristics of the problem and applying Johnson's algorithm. Sun, Morizawa, & Nagasawa (2003) design a series of heuristic algorithms, denoted as SMN_1 to SMN_{14}, based on the basic idea of Johnson's algorithm (Johnson, 1954) and compare them against the heuristics proposed in Lee et al. (1993). Lin, Cheng, & Chou (2006) show that the problem is strongly NP-hard even when all the jobs have the same processing time on the second-stage machine and design a heuristic, labelled as H_4, which is compared against LCL_1, LCL_2 and LCL_3 (Lee et al., 1993). In Allahverdi & Al-Anzi (2006), the two-stage ASP with setup times is addressed and the authors proposed two evolutionary algorithms and a simple and efficient algorithm, denoted as AA. Finally, Komaki & Kayvanfar (2015) address the problem with release time of jobs and developed several constructive heuristics, labelled as I_1 to I_{48}. The authors also propose a lower-bound and a metaheuristic algorithm. With respect to $DPm \rightarrow 1||\sum C_j$, the first reference addressing this objective is Tozkapan, Kirca, & Chung (2003), where the authors prove that there exists a sequence that is optimal for the problem and propose two heuristics, labelled $TCK1$ and $TCK2$, to find an upper bound for their B&B algorithm. Al-Anzi & Allahverdi (2006) also address this problem and propose three simple constructive heuristics ($S1$, $S2$ and $S3$) based on the idea of ordering the jobs according to the Shortest Processing Time (SPT) rule, and two additional constructive heuristics, labelled $A1$ and $A2$. Recently, Framinan & Perez-Gonzalez (2017b) develop a constructive heuristic, denoted as FAP, which outperforms the existing constructive heuristics and is based on the problem properties studied by Al-Anzi & Allahverdi (2006). Finally, Lee (2018) proposes six lower bounds and test them in a B&B algorithm. The author also designs four greedy-type constructive heuristics, labelled $G1$, $G2$, $G3$ and $G4$.

Regarding the $DPm \rightarrow 0$ layout, the problem has been mostly addressed with the objective of minimizing the total completion time. Sung & Yoon (1998) propose two constructive heuristics based on the SPT rule. The first one, denoted as $STPT$, schedules the order with the smallest total processing time across all m machines, and the second one, labelled $SMPT$, selects the order with the smallest maximum amount of processing time on any of the m machines. Leung, Li, & Pinedo (2005) propose a constructive heuristic that selects as the next order to be sequenced the one that would be completed the earliest, that is, the order with the Earliest Completion Time (ECT). Based on this idea and including some look-ahead concepts, Framinan & Perez-Gonzalez (2017a) propose a constructive heuristic and two specific local search mechanisms for the problem, $SHIFT_k$ and $SHIFT_{k_{OPT}}$.

There are few references addressing the $DPm \rightarrow Pm$ layout. In Sung & Kim (2008), a heuristic, SAK, applying a processing-time-based pairwise exchange mechanism is designed, while Allahverdi & Al-Anzi (2012) propose a mathematical model and three new metaheuristics, both minimising total completion times. Recently, Talens, Fernandez-Viagas, Perez-Gonzalez, & Framinan (2020) propose for the same objective two new constructive heuristics based on specific knowledge of the problem. The first one, NCH, constructs iteratively a sequence by selecting the most suitable job, and, for the second proposal, the NCH heuristic is embedded into a beam search-based constructive heuristic. The authors also carry out a computational evaluation which shows that the proposals are more efficient than the existing heuristics.

Finally, regarding the $DPm \rightarrow F2$, Koulamas & Kyparisis (2001) propose two heuristics to minimise the makespan, $H0K$ and $H3K$, using Johnson's algorithm, and analyse their worst-case performance ratio. In Komaki, Teymourian, Kayvanfar, & Booyavi (2017), the authors propose also for makespan minimisation a bio-inspired metaheuristic together with a lower bound and four constructive heuristics inspired from the lower bound. These heuristics (denoted as $DR1$, $DR2$, $DR3$ and $DR4$) compute different indices and sort them in a non-decreasing order.

To summarise the state of the art, although there are several approximate algorithms (heuristics and metaheuristics) that have been proposed to solve some related scheduling problems, to the best of our knowledge, the problem under consideration has not been addressed so far in the literature. So, the performance of all aforementioned algorithms is not clear for the problem under consideration. In addition, most of the proposals incorporate generic mechanisms, without taking advantage of the specific characteristics of the problem under study, which could help in the search of finding more efficient algorithms to solve the problem. To tackle these challenges, we propose a speed-up procedure based on specific properties of the problem in order to decrease the complexity of approximate algorithms. Using this procedure and some theoretical results of the problem, we also propose four heuristics which are compared against the heuristics identified from related scheduling problems.

3. Theoretical results

The speed up procedure proposed in this paper is based on the critical path. In this section we provide the definitions and proofs required using concepts from graph theory. To do so, it is convenient to depict a graph model associated to a sequence (solution) of the MASP. More specifically, for a given sequence $\Pi = (\pi_1, \ldots, \pi_n)$ in a MASP, a *direct* graph $G(\Pi) = (V, E)$ can be constructed, with V containing nodes $O_{i'j}$ ($i' \in \{1, \ldots, m_2\}$, $j \in \{1, \ldots, n\}$) representing the completion of the processing of job j on machine $m_1 + i'$, nodes $O_{0,j}$ ($j \in \{1, \ldots, n\}$) representing the completion of the processing of all components of job j, and a source node s. Note that, although the notation for the nodes and operations is the same as in Section 2 ($O_{i'j}$), the range of the indices i and i' is different as we here aggregate all operations previous to the assembly stage in a single node. The edges E in the graph connect some nodes with different weights:

- Each node $O_{i',j}$ ($i' \in \{1, \ldots, m_2 - 1\}$ and $j \in \{1, \ldots, n-1\}$) is connected to node $O_{i',j+1}$ and to node $O_{i'+1,j}$ with weights $p_{m_1+i',\pi_{j+1}}$, and p_{m_1+i+1,π_j}, respectively.
- Each node O_{0j} ($j \in \{1, \ldots, n\}$) is connected to node O_{1j} with weight p_{m_1+1,π_j}.
- The source node s is connected to nodes O_{0j} ($j \in \{1, \ldots, n\}$) with weights $\max_{i\in\{1,\ldots,m_1\}} \sum_{k=1}^{j} p_{i,\pi_k}$.

Fig. 2 shows an example of the graph model for an instance of the MASP with four jobs, and m_1 dedicated machines in the first phase followed by three operations in series in the second phase ($m_2 = 3$). It is clear that $G(\Pi)$ is a Directed Acyclic Graph (DAG in the following, see e.g. Cormen, Leiserson, Rivest, & Stein, 2009), a fact that we will use later since the optimal substructure property holds for finding the longest path in a DAG, but in any graph (i.e. a sub-path of the longest path is a longest path).

Using this representation, the critical path associated to a solution of a given instance of the MASP problem is the longest path from the source to the last operation of the last job. More specifically, given $G(\Pi)$, we define $\mathcal{P}_\Pi(O_{i'j})$ the critical path of $O_{i',j}$ as the longest path in $G(\Pi)$ going from s to $O_{i',j}$. In other words, $\mathcal{P}_\Pi(O_{ij})$ is an ordered set of vertices in E providing the maximum distance between s and $O_{i',j}$. It follows that $\mathcal{P}_\Pi(O_{mn})$ is simply the makespan yield by sequence Π, and that the makespan value (the length of such critical path) can be computed using the following set of equations[1] :

$$c_{i'\pi_k} = \max\{c_{i'-1,\pi_k}, c_{i',\pi_{k-1}}\} + p_{m_1+i',\pi_k} \quad i' \in \{1, \ldots, m_2\};$$

$$k \in \{1, \ldots, n\} \tag{7}$$

with $c_{i'\pi_0} = 0$ ($i' \in \{0, \ldots, m_2\}$) and $c_{0\pi_j} = \max_{i\in\{1,\ldots,m_1\}} \sum_{k=1}^{j} p_{i,\pi_k}$ ($j \in \{1, \ldots, n\}$).

$$C_{max} = c_{m_2\pi_n} \tag{8}$$

[1] Note that $c_{i'j}$ is the length of the path from the source until node $O_{i',j}$. Again, we intentionally use $c_{i'j}$ due to its relationship with the completion times (C_{ij}) defined in Section 2. In fact, it is easy to show that $c_{0j} = \max_{i\in\{1,\ldots,m_1\}}\{C_{i,j}\}$ ($j \in \{1, \ldots, n\}$), while $c_{i'j} = C_{m_1+i',j}$, with $i' \in \{1, \ldots, m_2\}$.

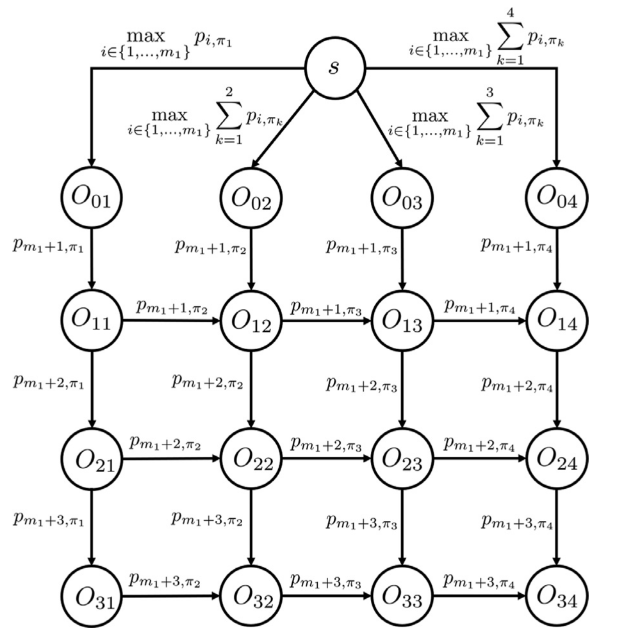

Fig. 2. Example of graph model for a MASP instance.

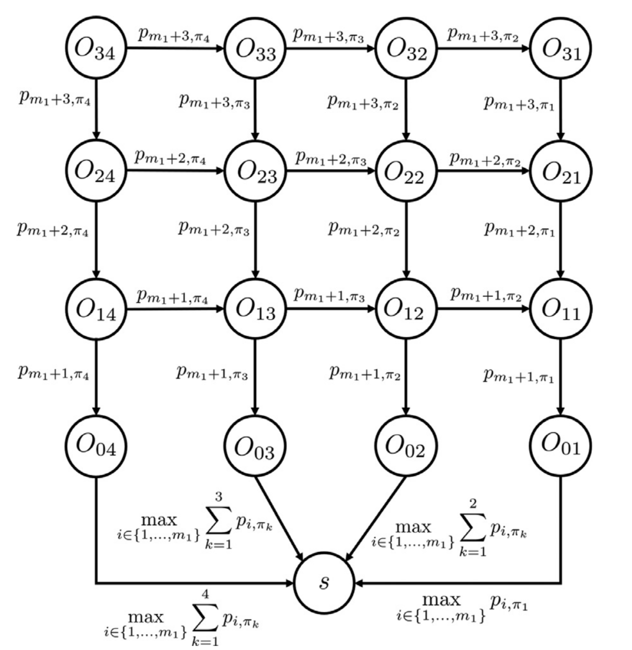

Fig. 3. Example of the reverse graph model for the MASP instance.

Along with the direct graph associated to a sequence in a MASP instance, we can also define $\bar{G}(\Pi)$ the *reverse* graph, which simply consists of transposing the direct graph and changing the direction of the edges. Fig. 3 provides the reverse graph of the MASP instance in Fig. 2. The reverse graph is also a DAG, so the optimal substructure property holds. Similarly, $\bar{\mathcal{P}}_\Pi(O_{i'j})$ can be defined as the longest path in the reverse graph from operation $O_{i',j}$ to the source. Its length can be computed using the following set of

```
Procedure InstancesGenerator()
    n := {50, 100, 150, 200, 250, 300};
    m1 := {2, 4, 6, 8};
    m2 := {1, 2, 5, 10, 15, 20};
    γ := {1, 1.2, 1.4, 1.6, 1.8, 2.0, 2.2, 2.4, 2.6, 2.8, 3.0};
    for j ∈ n do
        for i1 ∈ m1 do
            for i2 ∈ m2 do
                for l ∈ γ do
                    for k = 1 to 10 do
                        Generate processing times in the pre-assembly phase following a uniform distribution [1, 99];
                        Generate processing times in the assembly phase following a uniform distribution [1, 100 · l − 1];
                        C_max^NEH : Makespan obtained solving the problem with the NEH procedure;
                        C_max^IG : Makespan obtained solving the problem with the IG procedure;
                        ARPD1_lk := (C_max^NEH − C_max^IG) / C_max^IG · 100
                    end
                end
                Select the ten instances with lowest value of ARPD1_lk, l ∈ γ, k ∈ {1, . . . , 10};
            end
        end
    end
end
```

Fig. 4. Procedure to generate instances.

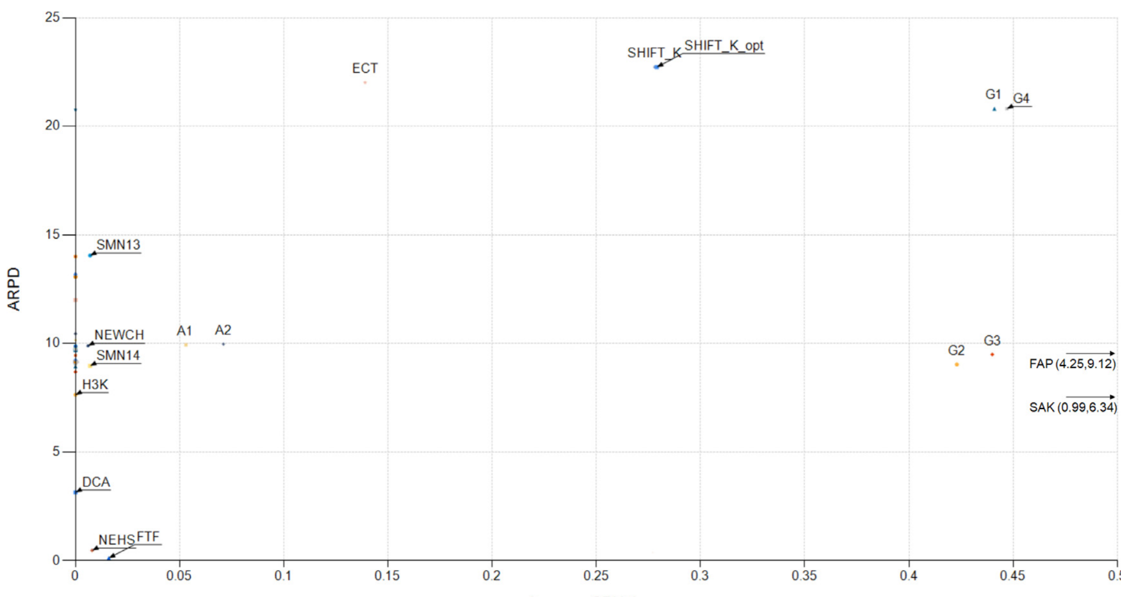

Fig. 5. Computational evaluation of heuristics. Average CPU time versus ARPD.

equations:

$$\bar{c}_{i'j} = \max\{\bar{c}_{i'+1,j}, \bar{c}_{i',j+1}\} + p_{m_1+i',\pi_j} \quad i' \in \{m_2, \ldots, 0\}; \quad j \in \{n, \ldots, 1\} \tag{9}$$

with $\bar{c}_{m_2+1,j} = 0$ $(j \in \{1, \ldots, n\})$ and $\bar{c}_{i',n+1} = 0$ $(i' \in \{0, \ldots, m_2\})$. Furthermore, $\bar{c}_s = \max_{j\in\{1,\ldots,n\}} \left(\bar{c}_{0j} + \sum_{k=1}^{j} p_{i',\pi_k}\right)$.

Obviously, the length of $\mathcal{P}_\Pi(O_{i'j})$ and $\bar{\mathcal{P}}_\Pi(O_{i'j})$ is the same (i.e. the makespan of the sequence).

Let us assume that, for a given partial sequence $\Pi = (\pi_1, \ldots, \pi_{k-1})$ of size $k-1$, the corresponding $c_{i'j}$ and $\bar{c}_{i'j}$ have been computed and that a job σ is to be inserted in position l ($l \in \{1, \ldots, k\}$). Then we can construct a graph for the sequence $(\pi_1, \ldots, \pi_{l-1}, \sigma, \pi_l, \ldots, \pi_{k-1})$ (we label it *augmented* graph to distinguish it from the graph obtained before the insertion). Clearly, c_{i',π_l-1} in the augmented graph are the same than in the original graph and $\bar{c}_{i'\pi_l}$ are the same than in the original reverse graph for all i'. With the values of c_{i',π_l-1} in the original graph we can compute $c^{\sigma}_{i'l}$ in the augmented graph using Eq. (8), i.e. we compute the longest path from s to operations $O_{i'l}$ taking into account that σ has been inserted in position l). Then the makespan (i.e. the longest path to the last operation of the last job in the direct graph) would be given by $\max_{i' \in \{0,\dots,m_2\}}\{c^{\sigma}_{i'l} + \bar{c}_{i'\pi_l}\}$ (see Eq. (10)), and thus the minimum makespan that can be obtained by inserting job σ in position $l \in \{1,\dots,k\}$ would be given by $\min_{l\in\{1,\dots,k\}}\left\{\max_{i' \in \{0,\dots,m_2\}}\{c^{\sigma}_{i'l} + \bar{c}_{i'\pi_l}\}\right\}$. Therefore, k^* the best position to insert job σ in a partial sequence is given by Eq. (11).

$$C_{max} = \max_{i' \in \{0,\dots,m_2\}}\{c^{\sigma}_{i'l} + \bar{c}_{i'\pi_l}\} \tag{10}$$

$$k^* = \arg\min_{l\in\{1,\dots,k\}}\left\{\max_{i' \in \{0,\dots,m_2\}}\{c^{\sigma}_{i'l} + \bar{c}_{i'\pi_l}\}\right\} \tag{11}$$

4. Proposed speed-up procedure

Equipped with the previous theoretical results, we propose in this section a speed-up procedure to accelerate the calculation of the makespan in insertion-based mechanisms. More specifically, this procedure can be applied in the insertion of any job in the best position of a partial sequence. Traditionally, in order to calculate the position which minimises the makespan, the job is tested in each position k (with $k \in \{1,\dots,n\}$) and its makespan value evaluated. To calculate the makespan, the completion time of each job π_l, $\Pi = (\pi_1,\dots,\pi_l,\dots,\pi_n)$, has to be obtained on each machine i (with $i \in \{1,\dots,m\}$). This traditional procedure has a complexity $O(n^2 \cdot m)$. This time complexity is reduced by the proposed procedure to $O(n \cdot m)$, by applying previous theoretical results[2]. A detailed explanation of this procedure is as follows:

STEP 1. Calculate completion times, C_{ij} using the forward codification (Eqs. (1)–(4)).

STEP 2. Calculate completion times, $\bar{C}_{ij}$ using the backward codification (Eqs. (5), and (6)).

STEP 3. For each position $k \in \{1,\dots,n\}$

STEP 3.1. Calculate the completion time of the new job in the first phase, when is inserted in position k:

$$C^{\sigma}_{ik} = C_{i,\pi_{k-1}} + p_{i,\sigma}, i \in \{1,\dots,m_1\} \tag{12}$$

STEP 3.2. Calculate the completion time of the new job in the first machine of the second phase, when is inserted in position k:

$$C^{\sigma}_{m_1+1,k} = \max\{\max_{i\in\{1,\dots,m_1\}}\{C^{\sigma}_{i,k}\}, C_{m_1+1,\pi_{k-1}}\} + p_{m_1+1,\sigma} \tag{13}$$

STEP 3.3. Calculate the completion time of the new job in the other machines of the second phase, when is inserted in position k:

$$C^{\sigma}_{i,k} = \max\{C^{\sigma}_{i-1,k}, C_{i,\pi_{k-1}}\} + p_{i,\sigma}, i \in \{m_1+2,\dots,m\} \tag{14}$$

STEP 3.4. For each machine $i \in \{1,\dots,m_1+m_2\}$

STEP 3.4.1.

$$C_{max,i} = C^{\sigma}_{ik} + \bar{C}_{i\pi_k} \tag{15}$$

STEP 3.5.

$$C^{k}_{max} = \max_{i\in\{1,\dots,m_1+m_2\}}\{C_{max,i}\} \tag{16}$$

STEP 4.

$$C_{max} = \min_{k\in\{1,\dots,n\}}\{C^{k}_{max}\} \tag{17}$$

STEP 5. Insert job σ in position $k^* = \arg\min_{k\in\{1,\dots,n\}}\{C^{k}_{\max}\}$.

[2] Note that, in this procedure, we explicitly compute the operations in each machine in the first phase. For a complete proof of these results using solely concepts from the specific scheduling problem, we refer to the on-line materials or http://grupo.us.es/oindustrial/en/research/results.

5. Proposed constructive heuristics

In this section, we propose four constructive heuristics for the problem, the first two being based on Johnson's algorithm. The first one constructs a complete sequence by reducing the problem to a 2-machine flow shop scheduling problem (detailed in Section 5.1), while the second divides the problem into several subproblems and solves them using Johnson's algorithm (detailed in Section 5.2). Regarding the NEH-based proposals, a simple NEH considering the speed-up procedure in the previous section is explained in Section 5.3. Finally, a proposal modifying the job to be inserted in the NEH algorithm is explained in Section 5.4.

5.1. Johnson-based constructive heuristic: JbH_{CB}

Recently, Fernandez-Viagas & Framinan (2017) show that, under certain conditions, the flow shop scheduling problem can be reduced to scheduling the jobs in the most saturated machine, or bottleneck. The proposed Johnson-based constructive heuristic, denoted as JbH_{CB}, takes advantage of this fact by reducing the flow shop in the second phase of our problem to a single machine. In this case, the problem under consideration could be reduced to the $DPm \to 1||C_{\max}$ problem. This reduced problem is solved by obtaining a sequence using Johnson's algorithm and then evaluating this sequence in the original problem. In order to construct a 2-machine flowshop instance suitable for Johnson's algorithm, the processing times of each job on such first machine are its sum of processing times in the first phase divided by the number of jobs, and those for the second machine are its processing times on the most saturated machine in the second phase. A detailed pseudo-code of this algorithm is shown in Appendix A (Fig. 6).

5.2. Divide-and-Conquer algorithm: DCA

The second Johnson-based proposal is a Divide-and-Conquer Algorithm (denoted as *DCA*), which reduces the $DPm \to Fm|prmu|C_{\max}$ problem to several 2-machine flowshop scheduling problems and solves them using Johnson's algorithm. More specifically, the algorithm firstly reduces the $DPm \to Fm|prmu|C_{\max}$ problem to a $Fm|prmu|C_{\max}$ with m_2+1 machines. The processing time p'_{1j} of each job j in the first machine is the maximum completion time in the first phase multiplied by a parameter a, ie. $p'_{1j} = \max_{i\in\{1,\dots,m_1\}}\{a \cdot p_{ij}\}$. The processing times in the other m_2 machines are the original processing times in the second phase, i.e. $p'_{i+1,j} = p_{m_1+i,j}$, with $i \in \{1,\dots,m_2\}$. Once a flow shop is obtained, following a similar procedure as in Campbell, Dudek, & Smith (1970), m_2 2-machine flowshop subproblems are generated with the following processing times:

$$p''_{1j} = \sum_{i=1}^{k}(k+1-i+(m_2+1)\cdot b)p'_{ij} \tag{18}$$

$$p''_{2j} = \sum_{i=m_2+1-k}^{m_2+1}(i-m_2+1+k+(m_2+1)\cdot b)p'_{ij} \tag{19}$$

Following a similar reasoning than in Dannenbring (1977), the weight of the processing times with respect to the machines is a valley, i.e. the first (last) machines have a higher (lower) weight in p''_{1j} and a lower (higher) one in p''_{2j}. Finally, a parameter b is introduced to calibrate this weight. A detailed pseudo-code of the algorithm is shown in Appendix A (Fig. 7).

5.3. NEH with the speed-up procedure: NEHS

In this section, two NEH-based constructive heuristics are proposed. Since the paper by Nawaz, Enscore, & Ham (1983), the NEH

Procedure JbH_{CB}
- **for** $j = 1$ **to** n **do**
  - $p'_{1j} = 0$;
  - **for** $i = 1$ **to** m_1 **do**
    - $p'_{1j} = p'_{1j} + p_{ij}$;
  - **end**
  - $p'_{1j} = p'_{1j}/n$;
- **end**
- Determine the most saturated machine in the second phase (denoted by i^*);
- **for** $j = 1$ **to** n **do**
  - $p'_{2j} = p_{i^*j}$;
- **end**
- Π := sequence obtained by applying Johnson's algorithm to the reduced two-machine flow shop problem using p'_{ij} (with $i \in \{1,2\}$) as processing times. Let OF denote the value of the objective function of such sequence for the $DPm \rightarrow Fm|prmu|C_{max}$ problem;

end

Fig. 6. Pseudocode of JbH_{CB}.

Procedure DCA
- **for** $j = 1$ **to** n **do**
  - $max = p_{ij}$;
  - **for** $i = 1$ **to** m_1 **do**
    - **if** $p_{ij} > max$ **then**
      - $max = p_{ij}$;
    - **end**
  - **end**
  - $p'_{1j} = a \cdot max$;
- **end**
- **for** $j = 1$ **to** n **do**
  - **for** $i = 1$ **to** m_1 **do**
    - $p'_{i+1,j} = p_{m_1+i,j}$;
  - **end**
- **end**
- **for** $k = 1$ **to** m_2 **do**
  - **for** $j = 1$ **to** n **do**
    - $p''_{1j} = \sum_{i=1}^{k}(k + 1 - i + (m_2 + 1) \cdot b) \cdot p'_{ij}$;
    - $p''_{2j} = \sum_{i=m_2+1-k}^{m_2+1}(i - m_2 + 1 + k + (m_2 + 1) \cdot b) \cdot p'_{ij}$;
  - **end**
  - Π^k := sequence obtained by applying Johnson's algorithm to the reduced kth two-machine flow shop problem using p'_{ij} (with $i \in \{1,2\}$) as the processing times. Let OF^k denote the value of the objective function of such sequence;
  - **if** $k = 1$ **then**
    - $OF^b := OF^k$;
  - **else if** $OF^k < OF^b$ **then**
    - $OF^b := OF^k$;
  - **end**
- **end**

end

Fig. 7. Pseudocode of *DCA*.

heuristic has become a cornerstone heuristic to solve scheduling problems. In its original version, this heuristic firstly sorts the jobs in non-increasing sum of processing times. Following this order, each job is inserted in the best position of an initially empty partial sequence, according to a certain objective function. This procedure is repeated until there is no more jobs in the initial sequence. Although initially proposed to minimise the makespan in a permutation flow shop scheduling problem, the NEH algorithm has been successfully adapted for several different scheduling problems (see e.g. Companys, Ribas, & Mateo, 2010; Vázquez-Rodríguez & Ochoa, 2011) and in fact, some extensions are state-of-the-art for several scheduling problems (see e.g. Chen, Yuan, Ng, & Cheng, 2021; Naderi & Ruiz, 2010).

Our first proposal, denoted as *NEHS*, is an adaptation of the traditional NEH algorithm with the incorporation of the proposed speed-up procedure (detailed in Section 4). More specifically, the jobs are initially sorted ($\alpha = (\alpha_1, \ldots, \alpha_n)$) in non-increasing sum of their processing times in all machines of the shop (i.e. $i \in$

Procedure *NEHS()*

α := Jobs ordered by non-increasing sum of processing times $p_j = \sum_{i=1}^{m} p_{ij}$, where $\alpha := (\alpha_1, ..., \alpha_i, ..., \alpha_n)$;
$\Pi := (\alpha_1)$;
for $k_1 = 2$ **to** n **do**
  Calculate completion times, C_{ij} using the forward codification (Equations 1, 2, 3, and 4);
  Calculate completion times, $\overline{C}_{ij}$ using the backward codification (Equations 5, and 6);
  for $k_2 = 2$ **to** k_1 **do**
    Calculate $C_{ik_2}^{\alpha_{k_1}}$, $i \in \{1, m\}$;
    $C_{max}^{k_2} = \max_{i \in \{1,...,m\}}\{C_{ik_2}^{\alpha_{k_1}} + \overline{C}_{i\pi_{k_2}}\}$;
  end
  Π := sequence obtained by inserting α_{k_1} in position k_2 with lowest $C_{max}^{k_2}$;
end
end

Fig. 8. Pseudo code of NEHS.

$\{1, \ldots, m_1 + m_2\}$). The first job of this order (α_1) forms a partial sequence, denoted as Π, initially composed of a single job (i.e. $\Pi = (\alpha_1)$). Then, in iteration k_1 (with $k_1 \in \{2, \ldots, n\}$), the following steps are repeated: i) Calculate the completion times (C_{ij}) using the forward codification; ii) Calculate completion times ($\overline{C}_{ij}$) using the backward codification; iii) Insert job α_{k_1} in the position of Π which minimises the makespan, applying the speed-up procedure explained in Section 4. A detailed pseudo-code of the algorithm is shown in Appendix A (Fig. 8).

5.4. Constructive heuristic FTF

In this section, we explain the second NEH-based constructive heuristic, denoted as *FTF* heuristic. As mentioned above, in the NEH algorithm, the job to be inserted in each iteration is taken iteratively from an initial sequence. Many different initial orders has been tested by researchers along the years (see e.g. Dong, Huang, & Chen, 2008; Kalczynski & Kamburowski, 2008; Kalczynski & Kamburowski, 2009; Liu, Jin, & Price, 2017), having a great influence in the performance of the algorithm. We try to take advantage of this influence by varying the job to be inserted in each iteration. More specifically, the *FTF* algorithm also starts with both the sequence α, sorting the jobs in non-increasing sum of their processing times, and the partial sequence $\Pi = (\alpha_1)$. Then, in each iteration k_1, the following two options are tested:

1. Insert job α_{k_1} in the best position of Π and denote the new partial sequence as Π^{aux}. After that, insert job α_{k_1+1} in the best position of Π^{aux} and denote Π^1 the new partial sequence.
2. Insert job α_{k_1+1} in the best position of Π and denote the new partial sequence as Π^{aux}. After that, insert job α_{k_1} in the best position of Π^{aux} and denote Π^2 the new partial sequence.

Once both options are evaluated and the best sequence among Π^1 and Π^2 is stored for the following iteration, replacing partial sequence Π. In addition, index k_1 is increased two units, as two jobs are inserted in each iteration. The procedure continues until there is no more jobs in α. Obviously, in case k_1 is equal to *n*, only α_{k_1} is tested in every position. A detailed pseudo-code of the algorithm is shown in Appendix A (Fig. 9).

6. Computational evaluation

In this section, we test the performance of the proposals by comparing them against the most promising approximate algorithms of the related literature. More specifically, a total of 51 algorithms are implemented and compared under the same computer conditions on an extensive set of instances. All experimentations carried out in the paper have been run on a cluster of computers Intel Core i7-8700 with 3.2 GHz and 8 GB RAM. To deal with this issue, the procedure to generate the test instances is detailed in Section 6.1. Parameters *a* and *b* used in heuristic *DCA* are calibrated in Section 6.2, while the all re-implemented heuristics of the related literature are enumerated in Section 6.3. Finally, the computational results are shown in Section 6.4.

6.1. Instances generation

In this section, we explain the procedure adopted to generate the instances. We follow a similar procedures as in Taillard (1993), Vallada, Ruiz, & Framinan (2015) and Fernandez-Viagas & Framinan (2020). In our case, a total of 1200 instances are generated to test the proposals, varying the number of jobs, and machines in the following manner:

- Number of jobs: $n \in \{50, 100, 150, 200, 250, 300\}$
- Number of machines in the pre-assembly phase: $m_1 \in \{2, 4, 6, 8\}$
- Number of machines/stages in the assembly phase: $m_2 \in \{1, 2, 5, 10, 20\}$

For each combination of these parameters, 110 instances are generated and 10 are chosen to compose the propose benchmark. In order to select the instances, we have to deal with the balance between the assembly and pre-assembly phases, since one is composed of dedicated parallel machines, while the other one is a flow shop. Under this situation, the balance to generate the processing times is not trivial. In this work, we introduce a parameter γ to generate the distribution of the processing times in the second phase. Thereby, the processing times in the dedicated machines (pre-assembly) stage are generated according to a uniform distribution [1,99], while in the assembly phase they are generated following a uniform distribution $[1, 100 \cdot \gamma - 1]$, with $\gamma \in \{1, 1.2, 1.4, 1.6, 1.8, 2.0, 2.2, 2.4, 2.6, 2.8, 3.0\}$. Next, 10 instances are generated for each value of γ, which results in 110 instances for each combination of *n*, m_1, and m_2. To determine the best ten instances for testing the algorithms in each combination of the parameters, we select the hardest one as done in Taillard (1993) and Vallada et al. (2015). To do so, we first compare in each instance the makespan value obtained by a reference constructive heuristic (the NEH algorithm proposed by Nawaz et al., 1983) and a reference metaheuristic (the iterated greedy IG proposed by Ruiz & Stützle, 2007). Then, the ten instances with lowest percentage difference between the two algorithms have been selected (a similar procedure is followed by Fernandez-Viagas & Framinan, 2020 to determine the hardest instances). Let β_1 denote the benchmark generated using this procedure. More specifically, considering the

```
Procedure FTF
    α := Jobs ordered by non-increasing due dates where α = {α_1, ..., α_i, ..., α_n};
    Π := {α_1};
    k_1 := 2;
    while k_1 ≤ n do
        Calculate completion times in Π, C_ij, using the forward codification (Equations 1, 2, 3, and 4);
        Calculate completion times in Π, C̄_ij, using the backward codification (Equations 5, and 6);
        for k_2 = 2 to k_1 do
            Calculate C^{α_k1}_{ik2}, i ∈ {1, m}, using Π and C_ij;
            C^{k2}_{max} = max_{i∈{1,...,m}} {C^{α_k1}_{ik2} + C̄_{iπ_k2}};
        end
        Π^aux := sequence obtained by inserting α_k1 in position k_2 with lowest C^{k2}_{max};
        if k_1 < n then
            Calculate completion times in Π^aux, C^1_ij, using the forward codification (Equations 1, 2, 3,
            and 4);
            Calculate completion times in Π^aux, C̄^1_ij, using the backward codification (Equations 5, and
            6);
            for k_2 = 2 to k_1 + 1 do
                Calculate C^{α_k1+1}_{ik2}, i ∈ {1, m}, using Π^aux and C^1_ij;
                C^{k2}_{max} = max_{i∈{1,...,m}} {C^{α_k1+1}_{ik2} + C̄^1_{iπ_k2}};
            end
            Π^1 := sequence obtained by inserting α_{k1+1} in position k_2 of Π^aux with lowest C^{k2}_{max}. Let k'
            denote such position and C^1_max such makespan;
            for k_2 = 2 to k_1 do
                Calculate C^{α_k1+1}_{ik2}, i ∈ {1, m}, using Π and C_ij;
                C^{k2}_{max} = max_{i∈{1,...,m}} {C^{α_k1+1}_{ik2} + C̄_{iπ_k2}};
            end
            Π^aux := sequence obtained by inserting α_{k1+1} in position k_2 with lowest C^{k2}_{max};
            Calculate completion times in Π^aux, C^2_ij, using the forward codification (Equations 1, 2, 3,
            and 4);
            Calculate completion times in Π^aux, C̄^2_ij, using the backward codification (Equations 5, and
            6);
            for k_2 = 2 to k_1 + 1 do
                Calculate C^{α_k1}_{ik2}, i ∈ {1, m}, using Π^aux and C^2_ij;
                C^{k2}_{max} = max_{i∈{1,...,m}} {C^{α_k1}_{ik2} + C̄^2_{iπ_k2}};
            end
            Π^2 := sequence obtained by inserting α_k1 in position k_2 of Π^aux with lowest C^{k2}_{max}. Let k''
            denote such position and C^2_max such makespan;
            if C^1_max < C^2_max then
                C_max = C^1_max; Π := Π^1;
            else
                C_max = C^1_max; Π := Π^2;
            end
        end
        k_1 = k_1 + 2;
    end
end
```

Fig. 9. Pseudo code of FTF.

ARPD1 as the Average Percentage Deviation between the NEH and the IG, the procedure to generate β_1 is detailed as follows in Fig. 4.

In addition, a different set of instances, β_2, is generated to fit the parameters of the proposals to avoid an overcalibration. β_2 consider the same levels of the parameters $n \in \{50, 100, 150, 200, 250\}$, $m_1 \in \{2, 4, 6, 8\}$, and $m_2 \in \{5, 10, 20\}$, plus parameter $\gamma = \{1, 1.25, 1.5, 1.75, 2, 2.25, 2.5\}$. For each combination of these four parameters, 10 instances are generated, using the same uniform distributions for the processing times as in the previous case.

6.2. Experimental parameter tuning

Among the four new proposals included in this paper, two parameters have to be calibrated only for the *DCA* heuristic, namely *a* and *b*. After some preliminary tests, we select the following levels for the calibration: $a \in \{1, 1.25, 1.5, 1.75, 2, 2.5, 3\}$ and $b \in \{0.025, 0.05, 0.1, 0.15, 0.20, 0.25\}$. The calibration has been performed using the Average Relative Percentage Deviation (ARPD2, Eq. 20) for each instance of benchmark β_2, where *Best* is the best value found for an instance. A non-parametric Kruskal-Wallis analysis performed separately for both parameters reveals that there are statistically significant differences between the levels of the parameters (all p-values found equals to 0.000). In addition, the best combination of parameters is found for $a = 2$ and $b = 0.15$, which are used in the subsequent experiments.

$$ARPD2 := \frac{C_{max} - Best}{Best} \cdot 100 \qquad (20)$$

6.3. Implemented heuristics

We have identified a number of heuristics from related problems that are to be compared against our proposals. These are:

- Heuristics for the $DPm \rightarrow 1||C_{\max}$ problem:
  - LCL_1, LCL_2 and LCL_3 (Lee et al., 1993): Let $p'_{1j} = \max_{i\in\{1,\cdots,m_1\}} p_{ij}$ and $p'_{2j} = \sum_{i=m_1+1}^{m} p_{ij}$. LCL_1 applies Johnson's algorithm to the job instance with job j defined by p'_{1j} and p'_{2j}. LCL_2 identifies the machine k with the maximum workload in the first phase ($\max_{i\in\{1,\cdots,m_1\}}\{\sum_{j=1}^{n} p_{ij}\}$), so that $p'_{1j} = p_{kj}$. Then, it applies Johnson's algorithm as in LCL_1, using the same p'_{2j}. Finally, LCL_3 applies Johnson's algorithm with $p'_{1j} = \sum_{i=1}^{m_1} \frac{p_{ij}}{m_1}$ and the previous p'_{2j}.
  - H_4 (Lin et al., 2006): Let $p'_j = \sum_{i=1}^{m_1} p_{ij}/p'_{2j}$ (p'_{2j} is computed as in $LCL1$), this heuristic arranges the jobs in non-decreasing order of p'_j.
  - SMN_{13} and SMN_{14} (Sun et al., 2003): These heuristics are adapted by computing $\sum_{i=m_1+1}^{m} p_{ij}$ each time the processing time in the second phase is considered. The steps of these heuristics can be consulted in the referred paper.
  - AA (Allahverdi & Al-Anzi, 2006): This algorithm inserts, step by step, an unscheduled job in an initially empty partial sequence. Among the unscheduled jobs, it first selects the job with minimum value of the maximum processing times in the first phase (Step 1a), and second the job with the minimum processing time in the second phase (Step 1b). If the value obtained from Step 1a is less than or equal to that obtained from Step 1b, it places the corresponding job in the next earliest available position in the sequence, otherwise it places the corresponding job in the next latest available position in the sequence.
  - 14 heuristics provided by the combination of different dispatching rules (Komaki & Kayvanfar, 2015): From the initial 16 dispatching rules proposed (I_1 to I_{16}), 7 (I_3, I_4, I_8, I_9, I_{10}, I_{11} and I_{12}) can be adapted and applied to the problem under study.
    * Non-decreasing order of $I_3 = \max_{i\in\{1,\cdots,m_1\}} p_{ij}$.
    * Non-decreasing order of $I_4 = \sum_{i=1}^{m_1} \frac{p_{ij}}{m_1}$.
    * Non-decreasing order of $I_8 = p_{i*j}$, where $i*$ is the most loaded machine in the first phase.
    * Non-increasing order of $I_9 = \sum_{i=m_1+1}^{m} p_{ij}$.
    * Non-increasing order of $I_{10} = \max_{i\in\{1,\cdots,m_1\}} p_{ij} + \sum_{i=m_1+1}^{m} p_{ij}$.
    * Non-increasing order of $I_{11} = \sum_{i=1}^{m_1} \frac{p_{ij}}{m_1} + \sum_{i=m_1+1}^{m} p_{ij}$.
    * Non-increasing order of $I_{12} = p_{i*,j} + \sum_{i=m_1+1}^{m} p_{ij}$, where p_{i*j} is the same as in I_8.

  After combining I_3, I_4 and I_8 with I_9, I_{10}, I_{11} and I_{12}, we obtain 12 additional dispatching rules (I_{17} to I_{28}) to which the Johnson's algorithm is applied. Note that the indicator of I_3 is the same as $DR1$, and $I_{17} = LCL_1$, $I_{18} = LCL_3$ and $I_{19} = LCL_2$.
- Heuristics for the $DPm \rightarrow 1||\sum C_j$ problem:
  - $TCK1$ and $TCK2$ (Tozkapan et al., 2003): TCK_1 obtains $m_1 + 1$ different sequences by applying the SPT in each machine of the first phase and in the machine of the second phase. To adapt this heuristic to our problem, the SPT rule obtained in the second phase is computed by using the sum of processing times in each machine of that second phase. Then, the sequence with the lowest $C_{\max}$ is selected. TCK_2 computes three indices for each job that have been adapted to our problem: $MPT_j = \min\{p_{1j}, p_{2j}, \cdots, p_{mj}\}$; $APT_j = \frac{1}{m_1+m_2}\sum_{i=1}^{m} p_{ij}$; and $MXPT_j = \max\{p_{1j}, p_{2j}, \cdots, p_{mj}\}$. Then, three sequences are obtained by sorting the jobs in non decreasing order of these indicators, and the sequence yielding the lowest $C_{\max}$ is selected.
  - $A1$ and $A2$ (Al-Anzi & Allahverdi, 2006): These algorithms construct a sequence by iteratively appending a job at the end of the current partial sequence. For algorithm A_1, the job is chosen so that the next indicator is minimised:

$$A1_j = \max_{i\in\{1,\ldots,m_1\}} \left\{ \sum_{r=1}^{j-1} p_{i[r]} + p_{ij} \right\} \qquad (21)$$

  while for algorithm A_2 the indicator is adapted to our problem by dividing the assembly time by the number of assembly machines, m_2, i.e.:

$$A2_j = \max_{i\in\{1,\ldots,m_1\}} \left\{ \sum_{r=1}^{j-1} p_{i[r]} + p_{ij} \right\} + \sum_{i=m_1+1}^{m} p_{ij} \qquad (22)$$

  - $S1$, $S2$ and $S3$ (Al-Anzi & Allahverdi, 2006): The authors designed several dispatching rules. Heuristic $S1$ sorts the jobs in non decreasing order of $\sum_{i=m_1+1}^{m} p_{ij}$. Heuristic $S2$ is obtained by sorting the jobs in non decreasing order of $\max_{i\in\{1,\ldots,m_1\}}\{p_{ij}\}$ and, finally, heuristic $S3$ sorts the jobs in non decreasing order of $\max_{i\in\{1,\ldots,m_1\}}\{p_{ij}\} + \sum_{i=m_1+1}^{m} p_{ij}$.
  - $G1$, $G2$, $G3$ and $G4$ (Lee, 2018): These heuristics construct a sequence by inserting the job with the smallest value of an indicator. The indicators for each heuristic are $G1_j = C_{[j]} - C_2^*$; $G2_j = C1_j^* - C1_{j-1}^*$; $G3_j = C1_j^* - C_2^*$ and $G4_j = C_{[j]} - C1_j^*$, being $C1_j^*$ the completion time of job j in the first phase and $C2^*$ the completion time of the last-positioned job in the second phase. These indicators have been adapted computing the completion times in the second phase using Eq. (4).
  - FAP (Framinan & Perez-Gonzalez, 2017b): This heuristic computes an indicator taking into account which phase is dominant and an estimation of the completion time of the unscheduled jobs. The indicator has been adapted considering $\sum_{i=m_1+1}^{m} p_{ij}$ instead of $p_\bullet$, which is defined as the sum of the processing times of the unscheduled jobs in the assembly phase.
- Heuristics for the $DPm \rightarrow 0||\sum C_j$ problem:
  - $STPT$ (Sung & Yoon, 1998): A sequence is constructed sorting the jobs in ascending order of their sum of their processing times on the m_1 machines. In our case, $m_1 + m_2$ are considered.
  - $SMPT$ (Sung & Yoon, 1998): A sequence is constructed sorting the jobs in ascending order of their maximum processing time on the m_1 machines. In our case, $m_1 + m_2$ are considered.
  - ECT (Ahmadi, Bagchi, & Roemer, 2005; Leung et al., 2005): In this heuristic, the order with the earliest completion time is selected as the next to be sequenced. The completion time is computed according to Eq. (4).
  - $SHIFT_k$ and $SHIFT_{k_{OPT}}$ (Framinan & Perez-Gonzalez, 2017a): $SHIFT_k$ obtains a partial sequence in an iterative manner using the ECT heuristic. Then, the jobs are iteratively removed from their position and reinserted. The procedure is repeated until the so-obtained partial sequence does not returns a lower objective function value. $SHIFT_{k_{OPT}}$ restarts the

Table 1
Computational results grouped by m_1 and m_2. Average CPU times (ACPU) are given in seconds in last columns.

Heuristic	m_1				m_2					ARPD2	ACPU
	2	4	6	8	1	2	5	10	20		
NEHS	0.377	0.529	0.527	0.465	0.336	0.160	0.364	0.683	0.832	**0.475**	**0.008**
FTF	0.076	0.169	0.146	0.115	0.302	0.050	0.043	0.043	0.196	**0.127**	**0.016**
JBH_{CB}	9.423	8.934	9.131	8.811	1.242	4.916	10.786	14.547	13.920	9.075	0.000
DCA	2.867	3.114	3.288	3.332	0.385	0.804	2.951	5.867	5.757	**3.150**	**0.000**
LCL_1	9.336	9.199	8.875	8.831	1.073	4.865	10.721	15.017	13.664	9.060	0.000
LCL_2	8.979	9.802	9.949	9.699	1.679	5.985	11.359	15.055	13.996	9.607	0.000
LCL_3	9.407	9.019	9.095	8.877	1.174	5.020	10.731	14.636	13.976	9.100	0.000
H4	9.208	8.972	8.904	8.723	1.228	4.754	10.576	14.533	13.705	8.952	0.000
SMN13	17.323	13.246	15.430	11.638	8.487	23.645	10.723	14.926	14.326	14.409	0.007
SMN14	9.321	9.123	8.892	8.557	1.132	4.939	10.494	14.704	13.634	8.973	0.007
AA	9.130	9.149	9.292	9.034	1.023	4.638	10.671	15.201	14.261	9.151	0.001
I4	9.419	9.050	9.203	8.946	1.265	5.042	10.865	14.691	13.948	9.155	0.000
I8	9.865	9.844	9.956	9.699	1.680	6.092	11.612	15.486	14.374	9.840	0.000
I9	9.838	9.851	10.012	9.764	1.807	6.181	11.457	15.635	14.285	9.866	0.000
I10	9.456	9.190	9.385	8.898	1.388	4.860	10.849	14.975	14.117	9.232	0.000
I11	9.512	9.209	9.436	8.891	1.440	4.837	10.851	15.071	14.138	9.262	0.000
I12	9.676	9.553	9.642	9.235	1.577	5.338	11.258	15.402	14.087	9.527	0.000
I20	9.391	9.281	8.967	9.032	1.277	5.082	10.728	15.017	13.773	9.168	0.000
I21	9.556	9.537	9.560	9.158	1.506	5.181	11.187	15.276	14.148	9.453	0.000
I22	9.862	9.859	9.948	9.689	1.694	6.092	11.603	15.486	14.362	9.840	0.000
I23	9.397	9.233	8.872	8.899	1.116	5.005	10.728	15.017	13.673	9.100	0.000
I24	9.419	9.050	9.203	8.946	1.265	5.042	10.865	14.691	13.948	9.155	0.000
I25	9.864	9.858	9.944	9.679	1.678	6.081	11.612	15.486	14.365	9.836	0.000
I26	9.398	9.271	8.952	9.037	1.275	5.057	10.728	15.017	13.783	9.164	0.000
I27	9.419	9.050	9.203	8.946	1.265	5.042	10.865	14.691	13.948	9.155	0.000
I28	9.865	9.844	9.956	9.699	1.680	6.092	11.612	15.486	14.374	9.841	0.000
TCK1	13.251	14.170	14.315	14.296	8.494	11.808	15.029	17.664	17.051	14.008	0.000
TCK2	10.225	10.072	10.418	9.896	2.004	6.277	12.196	15.865	14.458	10.153	0.000
A1	10.018	9.981	10.013	9.758	1.749	6.184	11.706	15.722	14.383	9.942	0.053
A2	10.200	9.880	9.932	9.889	1.860	6.136	11.810	15.827	14.283	9.975	0.071
S1	13.251	14.170	14.315	14.296	8.494	11.808	15.029	17.664	17.051	14.008	0.000
S2	9.391	9.878	9.595	9.939	1.730	5.641	11.659	15.456	14.058	9.701	0.000
S3	11.741	13.153	13.512	13.879	6.904	10.455	14.441	17.303	16.291	13.071	0.000
G1	20.673	20.914	20.921	20.714	6.796	15.545	25.416	30.420	25.922	20.806	0.441
G2	9.396	9.227	8.969	8.563	1.189	4.873	10.721	14.702	13.747	9.039	0.423
G3	9.640	9.410	9.575	9.369	1.135	5.529	11.467	15.424	13.979	9.499	0.440
G4	20.673	20.914	20.921	20.714	6.796	15.545	25.416	30.420	25.922	20.806	0.447
FAP	9.321	9.139	9.052	8.970	1.351	4.975	10.627	14.801	13.903	9.120	4.248
STPT	10.603	10.463	10.664	10.082	2.717	6.692	12.241	16.062	14.587	10.453	0.000
SMPT	12.480	13.403	13.541	13.484	5.976	11.666	15.029	17.664	15.832	13.227	0.000
ECT	21.635	21.975	22.296	22.164	7.993	18.044	27.271	30.506	26.338	22.018	0.139
$SHIFT_k$	22.340	22.833	22.942	22.797	7.909	18.763	28.549	31.647	26.843	22.728	0.278
$SHIFT_{k_{OPT}}$	22.340	22.833	22.942	22.797	7.909	18.763	28.549	31.647	26.843	22.728	0.279
SAK	6.462	6.445	6.400	6.035	0.595	2.323	7.076	11.051	10.663	6.336	0.989
NCH	10.076	10.135	9.949	9.427	1.773	5.880	11.741	15.835	14.286	9.897	0.006
DR1	9.391	9.281	8.967	9.032	1.277	5.082	10.728	15.017	13.773	9.168	0.000
DR2	10.890	11.987	12.341	12.802	8.926	12.326	12.271	13.379	13.122	12.005	0.000
DR3	20.610	20.789	21.065	20.612	11.337	25.435	22.281	23.768	21.058	20.769	0.000
DR4	11.514	13.129	13.844	14.298	7.818	11.961	14.668	16.229	15.344	13.196	0.000
H0K	8.380	8.613	8.819	8.969	2.948	10.049	10.085	11.021	9.404	8.695	0.000
H3K	7.019	7.638	7.871	8.029	2.141	6.187	7.981	10.804	11.104	7.639	0.000

reinsertion phase whenever a better sequence is found and repeats the process until no further improvement is found.

- Heuristics for the $DPm \rightarrow Pm||\sum C_j$ problem:
  - *SAK* (Sung & Kim, 2008): This heuristic sorts the jobs in non decreasing order of $psum_j = \sum_{i=1}^{m_1} p_{ij} + p_{m_1+1,j}$. Set k_1=1 and $k_2 = k_1$+1, it exchanges the k_1th job and the k_2th job. If the objective function value is improved, it keeps the exchange. If not, $k_2 = k_2$+1. In order to adapt it to our problem, the indicator is computed as $psum_j = \sum_{i=1}^{m} p_{ij}$.
  - *NCH* (Talens et al., 2020): This heuristic constructs iteratively a sequence by selecting one job among the unscheduled jobs and adding it at the end of the partial sequence. The job with the minimum value of the indicator $PHI_j = a \cdot IT_j + C_{m_1+1,j}$ is selected. The indicator has been adapted by computing $C_{m,j}$ as $\sum_{i=m_1+1}^{\lfloor m_1+m_2/2 \rfloor} p_{ij}/m_1$.
- Heuristics for the $DPm \rightarrow F2||C_{\max}$ problem:
  - Dispatching rules *DR*1, *DR*2, *DR*3 and *DR*4 (Komaki et al., 2017): *DR*1 and *DR*4 sort the jobs in non-decreasing order of $\max_{i \in \{1,\cdots,m_1\}} p_{ij}$ and $\max_{i \in \{1,\cdots,m_1\}} p_{ij} + \sum_{i=m_1+1}^{m} p_{ij}$, respectively. To adapt the dispatching rules *DR*2 and *DR*3, *DR*2 sorts the jobs according to non-decreasing $\sum_{i=m_1+1}^{\lfloor m_1+m_2/2 \rfloor} p_{ij}$ and *DR*3 according to non-decreasing $\sum_{i=\lfloor m_1+m_2/2 \rfloor+1}^{m} p_{ij}$.
  - Heuristic *H0K* (Koulamas & Kyparisis, 2001) applies Johnson's algorithm to the problem by solving the 2-machine flowshop problem where the processing times of the first machine are the average processing times of the first $\lfloor m_2/2 \rfloor$ machines in the assembly phase, and those of the second machine are the last $\lceil m_2/2 \rceil$ machines.
  - Heuristic *H3K* (Koulamas & Kyparisis, 2001) transforms the problem into a 3-machines flow shop and then applies the algorithm proposed by Röck & Schmidt (1983). The processing times of the first machine in the transformed problem

Table 2
Holm's procedure.

Hypothesis	p-value	γ_i	Wilcoxon	$\alpha/(2-\gamma_i+1)$	Holm's procedure
H_1: DCA=H3K	0.000	1	R	0.025	R
H_2: FTF=SAK	0.000	2	R	0.050	R

are the processing times in the pre-assembly phase. For the second machines, the average processing times of the first $\lfloor m_2/2 \rfloor$ machines in the assembly phase are considered. Finally, for the third machines, the last $\lceil m_2/2 \rceil$ processing times are considered.

All these heuristics are compared against the heuristics proposed in Section 5 (JbH_{CB}, DCA, NEHS, and FTF).

6.4. Comparison of heuristics

The comparison among the implemented algorithms is performed using Benchmark β_1, under the same computer conditions (same programming lenguage, C#, same common functions and libraries and same computer). Two common indicators are used to establish the most efficient algorithms for the problem under consideration. Firstly, the *ARPD*2 indicator is used as a measure of the quality of the solutions of each approximate algorithm (with *Best* as the best solution found in the instance). Secondly, the Average CPU time (also denoted as *ACPU*) is used to measure the computational effort required by each algorithm. The computational results are shown in Table 1 with regard to parameters m_1 and m_2. A summary of the average computational results is shown in Fig. 5. In terms of the quality of the solutions, the best result is clearly found by the FTF heuristic with an *ARPD*2 value of 0.127 requiring 0.016 seconds in average. This heuristic clearly outperforms the best heuristic from the literature, which is the SAK heuristic. This latter algorithm has an *ARPD*2 value of 6.336 found in 0.989 seconds. Using the ACPU and *ARPD*2 indicators, we show the set of non-dominated heuristics in bold in Table 1 (Pareto set of heuristics). This set is formed exclusively by the proposed heuristics DCA (0.000, 3.150), NEHS (0.008, 0.475) and FTF (0.016, 0.127), which clearly outperform any other implemented heuristic from the related literature. This hypothesis is confirmed by performing a Holm's procedure (Holm, 1979) comparing the heuristics against the closest ones from the literature (i.e. hypotheses DCA=H3K and FTF=SAK), and using a non-parametric Wilcoxon signed-rank test with a 0.95 confidence level. The statistical results are shown in Table 2, yielding a p-value equal to 0.000 in both cases.

In view of the results, the following comments can be done:

- Regarding the two-stage assembly scheduling problem ($DPm \rightarrow 1||C_{\max}$), several heuristics have been proposed in the literature up-to-now: LCL_1, LCL_2, LCL_3, H_4, SMN_{13}, SMN_{14}, I_3, I_4, I_8, I_9, I_{10}, I_{11}, I_{12}, $DR1$, $DR4$, and heuristics from I_{20} to I_{28}. This problem is found for $m_2 = 1$ in the proposed benchmark, i.e. in 240 instances. For this case, the average results, ARPD2, are shown in the sixth column of Table 1. Among all these heuristics developed specifically for that problem, the best ARPD2 is found by LCL_1 with $ARPD2 = 1.073$ and 0.000 average CPU times. In addition, SAK obtains an ARPD2 value of 0.595 in 0.566 seconds. Both heuristics are clearly outperformed by the new proposals. Thereby, DCA, NEHS, and FTF obtain an ARPD2 value of 0.385, 0.336 and 0.302, respectively. The ACPU required is 0.000, 0.004, and 0.007 seconds respectively. A non-parametric Wilcoxon signed-rank test has confirmed this assumption rejecting that $LCL_1 = DCA$ and $SAK = FTF$ with p-values equal to 0.000 in both cases.
- Similarly to the three stage assembly scheduling problem ($DPm \rightarrow F2||C_{\max}$), the best heuristic among the specific ones developed for the problem (i.e. $DR1$, $DR2$, $DR3$, $DR4$, $H0K$, and $H3K$) is $DR1$ with $ARPD2 = 5.082$ and $ACPU = 0.000$. Furthermore, it is worth noting the excellent performance found by the adaptation of the *AA* heuristic to the problem, with $ARPD2 = 4.865$ and $ACPU = 0.000$. Nevertheless, both heuristics are outperformed by the *DCA* heuristic ($ARPD2 = 0.804$) also requiring $ACPU = 0.000$. This is confirmed by a p-values equal to 0.000 in a Wilcoxon signed-rank test.
- Almost all adapted approximate algorithms perform relatively well for the problem under consideration with a low number of assemblies machines (mainly in case $m_2 = 1$). However, their performance highly worsen as m_2 increases. In fact, there is only one adapted heuristic with ARPD2 lower than ten (H0K) for $m_2 = 20$ and no one with ARPD2 lower than 9. In contrast, the NEHS and FTF proposals perform very well regardless the value of the parameter (NEHS with an ARPD2 lower than 0.900 and FTF lower than 0.350 for any m_2 value).
- The influence of parameter m_1 in the problem is much lower than that of m_2. Most of the implemented algorithms have a variation in the *ARPD*2 lower than 1.000 when m_1 is changed. In this regard, the best behaviour with the increase in m_1 is found by the SMN13 heuristic, which reduces the ARPD2 from 17.323 (in $m_1 = 2$) to 11.638 (in $m_1 = 8$).

7. Conclusions

In this paper, we have addressed the multi-stage assembly flow shop scheduling problem. This problem is a generalisation of both 2ASP and 3ASP. For the general problem, we have proposed a number of theoretical results that allow us to develop an efficient method to speed up approximate algorithms based on inserting jobs into a partial sequence and, as a consequence, this result can also be applied to any of its special cases. In addition, four constructive heuristics have been proposed to find high-quality approximate solutions for the problem under consideration. The first two proposals are based on Johnson's algorithm, while the last two algorithms are based on the NEH heuristic incorporating the new speed-up procedure. An extensive evaluation has been performed on a new hard set of instances specifically designed for the problem under consideration. In this evaluation, a total of 51 approximate algorithms have been again re-implemented and compared under the same conditions.

The computational evaluation has shown the excellent performance of the proposals to solve the problem with respect both to the quality of the solutions and to the computational effort required. More specifically, the set of efficient (non-dominated) heuristics for the multi-stage assembly flow shop scheduling problem and also for its small variants (i.e. $DPm \rightarrow 1||C_{\max}$ and $DPm \rightarrow F2||C_{\max}$) is formed exclusively by the new proposals DCA, NEHS, and FTF. Therefore, they can be considered as the new state-of-the-art constructive heuristics for these problems.

Acknowledgment

The authors wish to thank the referees for their comments on the earlier versions of the manuscript. 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 under the projects “DEMAND”, “IBSOS and “EFECTOS”, with references P18-FR-1149, 5835 and US-1264511, respectively.

Appendix A

In this appendix, we show the pseudocode of all our proposals. Thereby, Figs. 6–9 indicate the pseudocode for JbH_{CB}, *DCA*, *NEHS* and *FTF*, respectively.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.ejor.2021.10.001

References

Ahmadi, R., Bagchi, U., & Roemer, T. A. (2005). Coordinated scheduling of customer orders for quick response. *Naval Research Logistics, 52*(6), 493–512. https://doi.org/10.1002/nav.20092.

Al-Anzi, F. S., & Allahverdi, A. (2006). A hybrid tabu search heuristic for the two-stage assembly scheduling problem. *International Journal of Operations Research, 3*(2), 109–119.

Allahverdi, A., & Al-Anzi, F. S. (2006). Evolutionary heuristics and an algorithm for the two-stage assembly scheduling problem to minimize makespan with setup times. *International Journal of Production Research, 44*(22), 4713–4735. https://doi.org/10.1080/00207540600621029.

Allahverdi, A., & Al-Anzi, F. S. (2012). A new heuristic for the queries scheduling problem on distributed database systems to minimize mean completion time. In *Proceedings of the 21st international conference on software engineering and data engineering, SEDE 2012* (pp. 93–97).

Allahverdi, A., & Aydilek, H. (2015). The two stage assembly flowshop scheduling problem to minimize total tardiness. *Journal of Intelligent Manufacturing, 26*(2), 225–237. https://doi.org/10.1007/s10845-013-0775-5.

Campbell, H. G., Dudek, R. A., & Smith, M. L. (1970). Heuristic algorithm for the *n* job, *m* machine sequencing problem. *Management Science, 16*(10), 630–637.

Chen, R., Yuan, J., Ng, C. T., & Cheng, T. C. E. (2021). Single-machine hierarchical scheduling with release dates and preemption to minimize the total completion time and a regular criterion. *European Journal of Operational Research*. https://doi.org/10.1016/j.ejor.2020.12.006.

Companys, R., Ribas, I., & Mateo, M. (2010). Improvement tools for NEH based heuristics on permutation and blocking flow shop scheduling problems. In *IFIP Advances in information and communication technology, 338 AICT* (pp. 33–40). https://doi.org/10.1007/978-3-642-16358-6_5.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to algorithms* (3rd ed.). The MIT Press.

Dannenbring, D. G. (1977). An evaluation of flow shop sequencing heuristics. *Management Science, 23*(11), 1174–1182. https://doi.org/10.1287/mnsc.23.11.1174.

Dong, X., Huang, H., & Chen, P. (2008). An improved NEH-based heuristic for the permutation flowshop problem. *Computers & Operations Research, 35*(12), 3962–3968. http://www.sciencedirect.com/science/article/pii/S0305054807001116

Fattahi, P., Hassan Hosseini, S. M., & Jolai, F. (2013). Some heuristics for the hybrid flow shop scheduling problem with setup and assembly operations. *International Journal of Industrial Engineering Computations, 4*(3), 393–416. https://doi.org/10.5267/j.ijiec.2013.03.004.

Fernandez-Viagas, V., & Framinan, J. M. (2017). Reduction of permutation flowshop problems to single machine problems using machine dominance relations. *Computers and Operations Research, 77*, 96–110. https://doi.org/10.1016/j.cor.2016.07.009.

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

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

Framinan, J. M., & Perez-Gonzalez, P. (2017a). New approximate algorithms for the customer order scheduling problem with total completion time objective. *Computers and Operations Research, 78*, 181–192. https://doi.org/10.1016/j.cor.2016.09.010.

Framinan, J. M., & Perez-Gonzalez, P. (2017b). The 2-stage assembly flowshop scheduling problem with total completion time: Efficient constructive heuristic and metaheuristic. *Computers and Operations Research, 88*, 237–246. https://doi.org/10.1016/j.cor.2017.07.012.

Framinan, J. M., Perez-Gonzalez, P., & Fernandez-Viagas, V. (2019). Deterministic assembly scheduling problems: A review and classification of concurrent-type scheduling models and solution procedures. *European Journal of Operational Research, 273*(2), 401–417. https://doi.org/10.1016/j.ejor.2018.04.033.

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

Hwang, F. J., & Lin, B. M. T. (2012). Two-stage assembly-type flowshop batch scheduling problem subject to a fixed job sequence. *Journal of the Operational Research Society, 63*(6), 839–845.

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

Kalczynski, P. J., & Kamburowski, J. (2008). An improved NEH heuristic to minimize makespan in permutation flow shops. *Computers & Operations Research, 35*(9), 3001–3008. http://www.sciencedirect.com/science/article/pii/S0305054807000172

Kalczynski, P. J., & Kamburowski, J. (2009). An empirical analysis of the optimality rate of flow shop heuristics. *European Journal of Operational Research, 198*(1), 93–101. https://doi.org/10.1016/j.ejor.2008.08.021. http://www.sciencedirect.com/science/article/pii/S037722170800739X

Komaki, G. M., & Kayvanfar, V. (2015). Grey Wolf Optimizer algorithm for the two-stage assembly flow shop scheduling problem with release time. *Journal of Computational Science, 8*, 109–120. https://doi.org/10.1016/j.jocs.2015.03.011.

Komaki, G. M., Teymourian, E., Kayvanfar, V., & Booyavi, Z. (2017). Improved discrete cuckoo optimization algorithm for the three-stage assembly flowshop scheduling problem. *Computers and Industrial Engineering, 105*, 158–173. https://doi.org/10.1016/j.cie.2017.01.006.

Koulamas, C., & Kyparisis, G. (2001). The three-stage assembly flowshop scheduling problem. *Computers and Operations Research, 28*(7), 689–704. https://doi.org/10.1016/S0305-0548(00)00004-6.

Lee, C.-Y., Cheng, T. C. E., & Lin, B. M. T. (1993). Minimizing the makespan in the 3-machine assembly-type flowshop scheduling problem. *Management Science, 39*(5), 616–625. https://doi.org/10.1287/mnsc.39.5.616.

Lee, I. S. (2018). Minimizing total completion time in the assembly scheduling problem. *Computers and Industrial Engineering, 122*, 211–218. https://doi.org/10.1016/j.cie.2018.06.001.

Leung, J. Y. T., Li, H., & Pinedo, M. (2005). Order scheduling in an environment with dedicated resources in parallel. *Journal of Scheduling, 8*(5), 355–386. https://doi.org/10.1007/s10951-005-2860-x.

Liao, C.-J., Lee, C.-H., & Lee, H. C. (2015). An efficient heuristic for a two-stage assembly scheduling problem with batch setup times to minimize makespan. *Computers and Industrial Engineering, 88*, 317–325. https://doi.org/10.1016/j.cie.2015.07.018. 4113

Lin, B. M. T., Cheng, T. C. E., & Chou, A. S. C. (2006). Scheduling in an assembly-type production chain with batch transfer. *Omega, 35*(2), 143–151. https://doi.org/10.1016/j.omega.2005.04.004.

Liu, W., Jin, Y., & Price, M. (2017). A new improved NEH heuristic for permutation flowshop scheduling problems. *International Journal of Production Economics, 193*, 21–30. https://doi.org/10.1016/j.ijpe.2017.06.026.

Naderi, B., & Ruiz, R. (2010). The distributed permutation flowshop scheduling problem. *Computers & Operations Research, 37*(4), 754–768. http://www.scopus.com/inward/record.url?eid=2-s2.0-70350752447&partnerID=40&md5=668c5360d210fc7c4da0d99874bc9c94

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

Nowicki, E. (1999). The permutation flow shop with buffers: A tabu search approach. *European Journal of Operational Research, 116*(1), 205–219. https://doi.org/10.1016/S0377-2217(98)00017-4.

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. https://www.scopus.com/inward/record.uri?eid=2-s2.0-0032050653&partnerID=40&md5=8d49189a360b07184cf08d3ebfcc67eb

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. https://doi.org/10.1016/j.omega.2013.12.004.

Pinedo, M. (2012). *Scheduling: Theory, algorithms and systems*. Springer.

Potts, C. N., Sevast'janov, S. V., Strusevich, V. A., Van Wassenhove, L. N., & Zwaneveld, C. M. (1995). The two-stage assembly scheduling problem: Complexity and approximation. *Computers and Operations Research, 43*(2), 346–355.

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. http://www.sciencedirect.com/science/article/pii/S030505481000050X

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. https://doi.org/10.1504/EJIE.2013.058392. http://www.scopus.com/inward/record.url?eid=2-s2.0-84891367893&partnerID=40&md5=dcc193d7a615114fb7316b9354a57d89

Rios-Mercado, R. Z., & Bard, J. F. (1998). Heuristics for the flow line problem with setup costs. *European Journal of Operational Research, 110*(1), 76–98. http://www.scopus.com/inward/record.url?eid=2-s2.0-0032189491&partnerID=40&md5=e8648b37d7ae6a878baabc93854ebfde

Röck, H., & Schmidt, G. (1983). Machine aggregation heuristics in shop scheduling. *Methods of Operations Research, 45*, 303–314.

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.

Sheikh, S., Komaki, G. M., & Kayvanfar, V. (2018). Multi objective two-stage assembly flow shop with release time. *Computers and Industrial Engineering, 124*(3), 276–292. https://doi.org/10.1016/j.cie.2018.07.023.

Sun, X., Morizawa, K., & Nagasawa, H. (2003). Powerful heuristics to minimize makespan in fixed, 3-machine, assembly-type flowshop scheduling. *European Journal of Operational Research, 146*(3), 498–516. https://doi.org/10.1016/S0377-2217(02)00245-X.

Sung, C. S., & Kim, H. A. (2008). A two-stage multiple-machine assembly scheduling problem for minimizing sum of completion times. *International Journal of Production Economics, 113*(2), 1038–1048. https://doi.org/10.1016/j.ijpe.2007.12.007.

Sung, C. S., & Yoon, S. H. (1998). Minimizing total weighted completion time at a pre-assembly stage composed of two feeding machines. *International Journal of Production Economics, 54*(3), 247–255. https://doi.org/10.1016/S0925-5273(97)00151-5.

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

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

Talens, C., Fernandez-Viagas, V., Perez-Gonzalez, P., & Framinan, J. M. (2020). New efficient constructive heuristics for the two-stage multi-machine assembly scheduling problem. *Computers and Industrial Engineering, 140*, Article 106223. https://doi.org/10.1016/j.cie.2019.106223.

Tozkapan, A., Kirca, O., & Chung, C. S. (2003). A branch and bound algorithm to minimize the total weighted flowtime for the two-stage assembly scheduling problem. *Computers and Operations Research, 30*(2), 309–320. https://doi.org/10.1016/S0305-0548(01)00098-3.

Vallada, E., Ruiz, R., & Framinan, J. M. (2015). New hard benchmark for flowshop scheduling problems minimising makespan. *European Journal of Operational Research, 240*, 666–677. http://www.scopus.com/inward/record.url?eid=2-s2.0-84905527090&partnerID=40&md5=0fbd129d7b07bd21fc567ef759e8135d

Vázquez-Rodríguez, J. A., & 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. https://doi.org/10.1057/jors.2010.132.

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. https://doi.org/10.1007/s00170-013-4819-y.

Zhang, Y., Zhou, Z., & Liu, J. (2010). The production scheduling problem in a multi-page invoice printing system. *Computers and Operations Research, 37*(10), 1814–1821. https://doi.org/10.1016/j.cor.2010.01.014.