Efficient Heuristics for the Hybrid Flow Shop Scheduling Problem with Missing Operations*

Manuel Dios[1], Victor Fernandez-Viagas[1], Jose M. Framinan[1]

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

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

February 4, 2019

Abstract

In this paper, we address the hybrid flowshop scheduling problem for makespan minimisation. More specifically, we are interested in the special case where there are missing operations, i.e. some stages are skipped, a condition inspired in a realistic problem found in a plastic manufacturer. The main contribution of our paper is twofold. On the one hand we carry out a computational analysis to study the hardness of the hybrid flowshop scheduling problem with missing operations as compared to the classical hybrid flowshop problem. On the other hand, we propose a set of heuristics that captures some special features of the missing operations and compare these algorithms with already existing heuristics for the classical hybrid flowshop, and for the hybrid flowshop problem with missing operations. The extensive computational experience carried out shows that our proposal outperforms existing methods for the problem, indicating that it is possible to improve the makespan by interacting with the jobs with missing operations.

1 Introduction

Flowshop scheduling problems have been largely studied in the literature during the last 50 years (Pan et al., 2014). Due to a rising demand of products, both in variety and quantity, it is commonplace that companies increase their capacity by adding new resources (both physical and human) to some stages in the manufacturing process (Ribas et al., 2010). As a result, some processing stages are formed by several machines, and a flowshop layout

*Preprint submitted to Computers & Industrial Engineering. DOI:10.1016/j.cie.2017.10.034

turns into a hybrid flowshop, a manufacturing setting that is gaining importance nowadays (Ruiz and Vazquez-Rodriguez, 2010).

Depending on the characteristics of the machines in every stage, a number of variations in the Hybrid Flowshop Scheduling (HFS) problem have been analysed in the literature: identical machines (the processing time for all machines in a stage is the same), uniform machines (each machine in a stage has a speed parameter associated that controls the processing time), and unrelated machines (the processing times of the different machines in a stage are independent). Note that this problem is also denoted in the literature as "Flexible Flow Line Scheduling", see e.g. Leon and Ramamoorthy (1997) and Kurz and Askin (2003) or simply as "Hybrid Flowshop Problem", see e.g. Ruiz et al. (2008). In this paper, we focus on the HFS problem with identical parallel machines and with the possibility of missing operations, i.e. not every job has to go through all the stages. More specifically, we refer to the problem under study as Hybrid Flowshop Scheduling with Missing Operations (HFSMO) problem.

Regarding the objective function of the HFS problem, the main objectives used in the literature can be classified into those related with the completion time of the jobs, with the delay of jobs, with multiple criteria, or those with an specific problem-dependent criterion (Ribas et al., 2010). Among all objectives, the makespan is clearly the most common. In this paper, we focus on this objective, which would enable us to compare and to discuss our results with existing contributions. Note that the HFSMO problem with makepan objective is denoted as $FHm\big(PM^{(k)}\big)_{k=1}^{m}|skip|C_{\max}$ according to the notation in e.g. Vignier et al. (1999) and Ruiz and Vazquez-Rodriguez (2010). This problem is NP-hard for 2 stages if at least one of them has more than one machine, since it is a generalization of the HFS problem, which is known to be NP-hard with makespan objective (Gupta, 1988) for the same conditions.

The HFSMO problem has not been widely analysed in the literature. Most often, it has been addressed as an ordinary HFS problem assuming that a missing operation

is tantamount to an operation with zero processing time. Nevertheless, this assumption implies that every job has to be processed in every stage even with zero processing times, therefore the completion time of the jobs increases (see e.g. Leisten and Kolbe, 1998 and Sridhar and Rajendran, 1993). Additionally, to the best of our knowledge, no paper in the literature has analysed the empirical hardness of the HFS problem when such a constraint is added. In this paper, first we analyse in detail the space of solutions of the HFSMO problem as compared to the traditional HFS problem. Secondly, we propose new efficient algorithms to solve the HFSMO problem. We focus on improvement heuristics, i.e. heuristics that start from an already built schedule and try to improve it according to a given procedure. The heuristics proposed exploit the specific properties of the missing operations in the HFS setting, and the computational experience carried out shows that they obtain better results for the problem under study and compare favourably to existing heuristics.

The paper is organized as follows: Section 2 contains a description of the HFSMO problem and discusses existing contributions. In Section 3, we conduct a computational analysis of the empirical hardness of both HFS and HFSMO problems. Section 4 describes the heuristics proposed. Section 5 presents the comparison between the new set of heuristics and the existing ones. Finally, in Section 6, we present the conclusions of our work.

2 Problem statement and background

The HFSMO problem with makespan objective can be stated as follows: We consider a set of n jobs, $\mathcal{N} = \{1, ..., n\}$ that have to be processed in s different stages, $\mathcal{S} = \{1, ..., s\}$. Each stage is composed of a set of m_j identical parallel machines, $\mathcal{M}_j = \{1, ..., m_j\}$, i.e. the processing time of the job on all machines within a stage is the same. Thus, the processing time of a job on a specific stage can be denoted as p_{ij}, where i ($i \in \mathcal{N}$) denotes

the job, and j ($j \in \mathcal{S}$) the stage. Every job has the same routing through the stages, but some of them can be skipped, i.e. there may be missing operations. Jobs are processed by exactly one machine at each stage. The objective is to find the sequence of jobs on each stage so the maximum completion time (makespan) is minimised.

As mentioned in Section 1, the general HFS problem –from which the HFSMO problem is a particular case– has been widely analysed in the literature, and several reviews can be found. In Linn and Zhang (1999), the different types of HFS problems are reviewed according to the number of stages: 2-stage, 3-stage and k-stage. Ribas et al. (2010) present a review focusing on the characteristics of the problem, i.e. the type of machines within each stage and the job constraints, discussing the different solution approaches found in the literature. With a similar approach, Ruiz and Vazquez-Rodriguez (2010) review more than 200 works using the classification and nomenclature of Vignier et al. (1999). An evaluation of mixed integer linear programming models for the HFS problem is performed by Naderi et al. (2014), developing and comparing four different mathematical models.

Among the approximate procedures available for the HFS problem with makespan objective, four heuristics are worth to note. The first one is the NEH, first proposed by Nawaz et al. (1983) for the permutation flowshop scheduling problem, PFSP. This heuristic starts by ordering jobs according to the highest sum of processing times across all stages ($\sum_j p_{ij}$) and constructs a solution by selecting all jobs, one by one, by inserting them into the position providing the lowest (partial) makespan (the pseudocode of this heuristic is presented in Section 4). The NEH is first applied to hybrid flowshops by Brah and Loo (1999), and the authors found that it outperforms other usual flowshop heuristics such as CDS1, CDS2, PAM and HO.

A similar heuristic, denoted as Raj, is that by Rajendran (1993), a modification of the NEH that reduces its computational time at the expense of decreasing the quality of the solution. In this case, instead of testing each job in all positions, it is inserted in half of them. Its pseudocode is shown in Section 4.

Finally, the heuristics WT1_NEH(x) and WT2_NEH(x) by Kizilay et al. (2015) are based on the profile-fitting method by McCormick et al. (1989). In these heuristics, a number of sequences are constructed based on a parameter x, and the one obtaining the minimum makespan is sought by reducing as much as possible the waiting time between jobs in each stage. Once a solution is obtained, the NEH heuristic is applied, using the solution found by the previous procedure (instead of using the order obtained by computing the longest processing time of the jobs as initial solution order). Both heuristics are shown to perform better that the NEH for the HFS problem with makespan objective.

Among the works considering explicitly missing operations, Leisten and Kolbe (1998) address missing operations in the classical m-machine flowshop setting. These authors analyse the suitability of using strict permutation schedules against the possibility of allowing for job passing. Their conclusion is that advantages can be found when calculating makespan (or total flowtime) if job passing is allowed. For the same layout, Rajendran and Ziegler (2001) analyse the performance of several dispatching rules and a heuristic using a percentage of missing operations of 0%, 20%, 30% and 40%. In Tseng et al. (2008), the difference between permutation and non permutation schedules in a two-stage HFSMO problem is studied, showing that non-permutation schedules obtain better results without greatly increasing the computation times. They present a heuristic for generating a non permutation schedule from a permutation sequence, obtaining a reduction in the makespan values. The heuristic was tested on a set of instances considering the following probability of missing operations: 0%, 20% and 40%. More recently, Saravanan et al. (2014) solve the k-stage HFSMO problem with makespan objective considering the same percentages of missing operations. They use two different metaheuristics (Simulated Annealing and Particle Swarm Optimization), obtaining that the former outperforms the latter, both in computational time and in the quality of the solutions. Finally, Marichelvam and Prabaharan (2014) propose a new hybrid metaheuristic for the k-stage HFSMO problem with makespan objective named Improved Hybrid Genetic Scatter Search (IHGSS), where

a Genetic Algorithm (GA) and a Scatter Search (SS) are combined. The authors compare its results against the standard versions of the GA and the SS, concluding that the IHGSS outperforms both metaheuristics.

Apart from these contributions specifically devoted to missing operations, some authors also take them into account for more constrainted layouts: In Kurz and Askin (2003), a set of heuristics are proposed for the k-stage hybrid flowshop with sequence-dependent setup times, assuming the possibility that the jobs skip some stages. The authors consider three levels of this skipping probability, 0%(low), 0.05%(medium) and 40%(high). In Naderi et al. (2010), a new dispatching rule and an iterated local search metaheuristic are proposed for the HFSMO problem with sequence-dependent setup times. In their case, two levels of skipping probability are considered, i.e. 10% and 40%.

Note that it is also possible to find some contributions on missing operations in problems slightly different to the classical HFS problem, such as in e.g. Behnamian and Fatemi Ghomi (2011), where the case with machine- and resource-dependent processing times is addressed using a genetic algorithm and a variable neighborhood search. In this case, the authors consider the possibility that jobs can skip stages, and assume the same skipping probabilities as in Kurz and Askin (2003).

As a summary, there are many heuristics available for the HFS problem with makespan objective, and a few of them also for the particular case HFSMO. However, some issues remain open:

1. Are HFS and HFSMO problems substantially different? To the best of our knowledge, there is no study assessing whether the additional feature of missing operations make these problems different. If this is the case, developing specific heuristics for the problem could be fully justified. This issue is addressed in Section 3.

2. Is it possible to use the special structure of the HFSMO to develop efficient solution methods for the problem? As suggested by the literature review, the missing operations should not be simply treated as zero processing times operations, so the special

structure of the problem could be exploited. This aspect is dealt with in Section 4.

3 Analysis of the problem

In this section we present an empirical analysis of the structure of solutions of the HFSMO problem as compared to that of the HFS problem. The idea is to analyse how hard is –in statistical terms– to obtain good solution for the problems under study. To do so, a complete enumeration of the space of solutions of the problem is carried out for a high number of instances, and the deviation with respect to the optimal solution for each of them is studied. In this manner, the empirical distribution of the deviation from the optimal makespan can be obtained. This type of analysis has been proved to be very useful for different scheduling problems (see e.g. Taillard, 1990; Armentano and Ronconi, 1999; Framinan et al., 2001, or Perez-Gonzalez and Framinan, 2015). A disadvantage is that this method can only be used for a small number of jobs/stages, given the NP-hard nature of the problem discussed in Section 1.

In the analysis carried out, we use a simple permutation encoding following a FIFO discipline (see e.g. Santos et al., 1996, Naderi et al., 2010 or Pan et al., 2014 for similar encoding). With respect to the instances to be analysed, different parameters have to be determined. The processing times are generated according to the [1,99] uniform distribution, which is the most common distribution used in the literature, both for the PFSP (Taillard, 1993) and for the HFS (Ruiz et al., 2008). Regarding the percentage of missing operations, authors typically use 3 levels, i.e. 0%, 20%, and 40% (see e.g. Tseng et al., 2008 and Saravanan et al., 2014). We add the level 60% to extend the analysis. In order to set the specific number of missing operations for each job, we follow a procedure similar to Rajendran and Ziegler (2001), i.e. we generate all processing times and then make a $x\%$ of the total number of operations to be equal to zero. We avoid that a job has all its operations with processing times equal to zero by limiting the number of missing

Table 1: Hardness analysis parameters

Parameter	**Values**
n	5,6,7,8
p_{ij}	$U\,[1, 99]$
% Missing Operations	0%,20%,40%,60%
S	3,5,10,15
m_j	1,2,$U\,[1, 5]$,$U\,[2, 4]$
Number of Instances	300

operations in a job to be $(s-1)$ at maximum. Regarding the number of stages, we make the analysis for 3, 5, 10 and 15 stages.

Regarding the number of machines per stage, we consider two deterministic and two uniformly distributed cases. In the first case, we assume that there are one or two machines in every stage, respectively (clearly, the first case is the standard flowshop layout). For the third case, a uniform [1,5] distribution is employed to generate the number of machines in each stage, so, it is quite likely to have a bottleneck stage. Finally, the fourth case uses a [2,4] uniform distribution, which is expected to provide a more balanced shop. In order to ensure the statistical significance of the analysis we use 300 instances for each problem size, thus processing a total of 76,800 instances for each type of analysis. A summary of the parameters employed is shown in Table 1.

The results of the complete enumeration (Figures 1- 5) are shown as an empirical probability distribution in Figures 1-5, after analysing the objective function value of each solution in each instance. In these figures, the x-axis represents percentual distance to the optimal value of the objective function, while the y-axis represents (empirical) probability that a solution is within the corresponding percentual distance (non accumulative). In other words, the charts give the empirical distribution function of the random variable 'percentual distance to the optimal value of the objective function'. According to the results, the following conclusions can be stated:

- Number of Jobs: There is an increasing trend in the hardness as the number of jobs

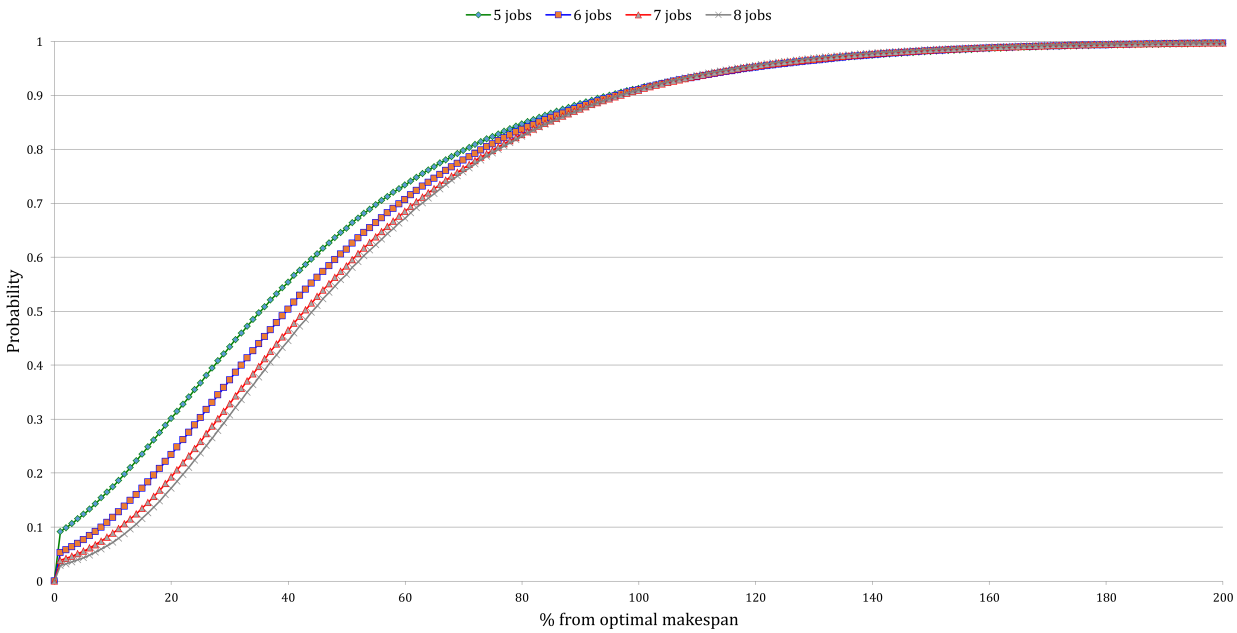

Figure 1: Cummulative distribution function for $m_j = U[1, 5]$ (jobs)

increases. This pattern is quite similar for any number of machines per stage. For instance, in Figure 1 we see the case of an hybrid flowshop with $m_j = U[1, 5]$, where the probability to find solutions closer to the optimal solution is always lower for the case with more jobs (the same pattern has been found for different number of machines per stage).

- % Missing Operations: The pattern followed by the empirical hardness is similar for $m_j = 1$ and $m_j = U[1, 5]$. In these cases, the highest hardness is achieved for the problems without missing operations, and the tendency is that the higher the percentage of missing operations, the less harder the problem is. Moreover, there is a difference in the hardness of $m_j = 1$ and $m_j = U[1, 5]$, being the latter harder than the former (see Figures 2 and 3).

  For $m_j = 2$ and $m_j = U[2, 4]$, the tendency is the opposite (see Figures 4 and 5), i.e. the higher the percentage of missing operations, the harder the problem is. However, for $m_j = 2$ and $m_j = U[2, 4]$ the difference in hardness is not as pronounced as in the previous two cases. The most significant difference is that, for $m_j = 2$, there is

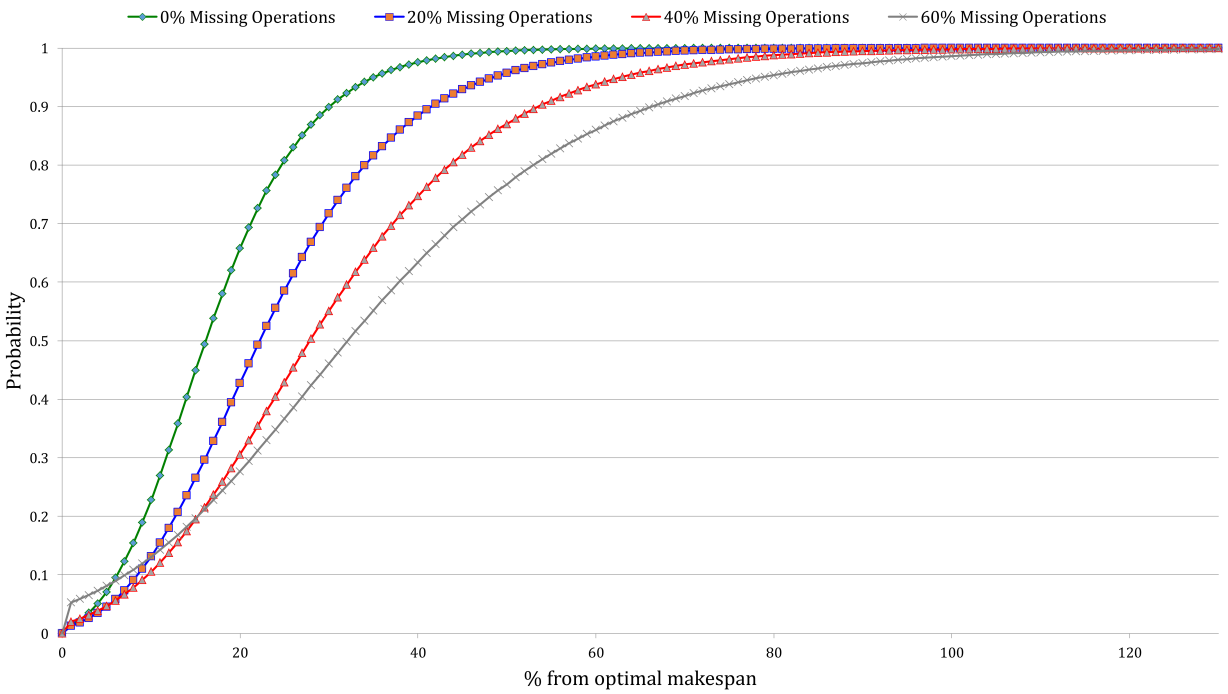

Figure 2: Cummulative distribution function for $m_j = 1$ (missing operations)

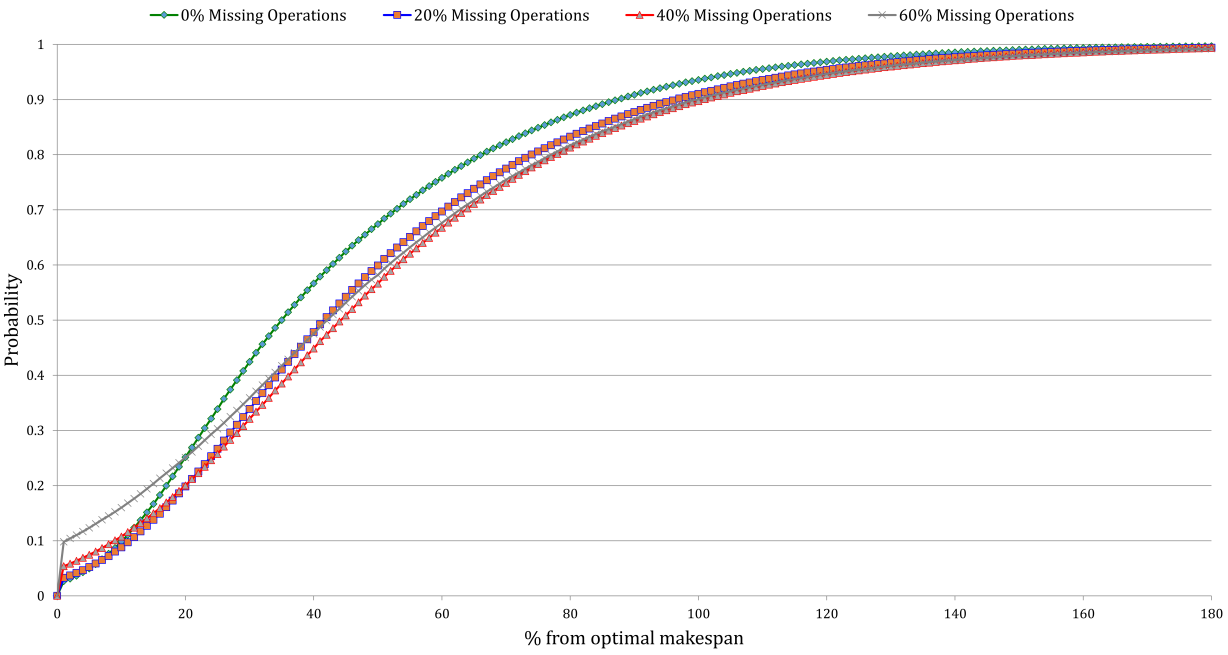

Figure 3: Cummulative distribution function for $m_j = U\,[1, 5]$ (missing operations)

a slightly smaller difference in hardness between problems with 0%, 20% and 40% of missing operations (see Figures 4 and 5). A possible explanation of such different

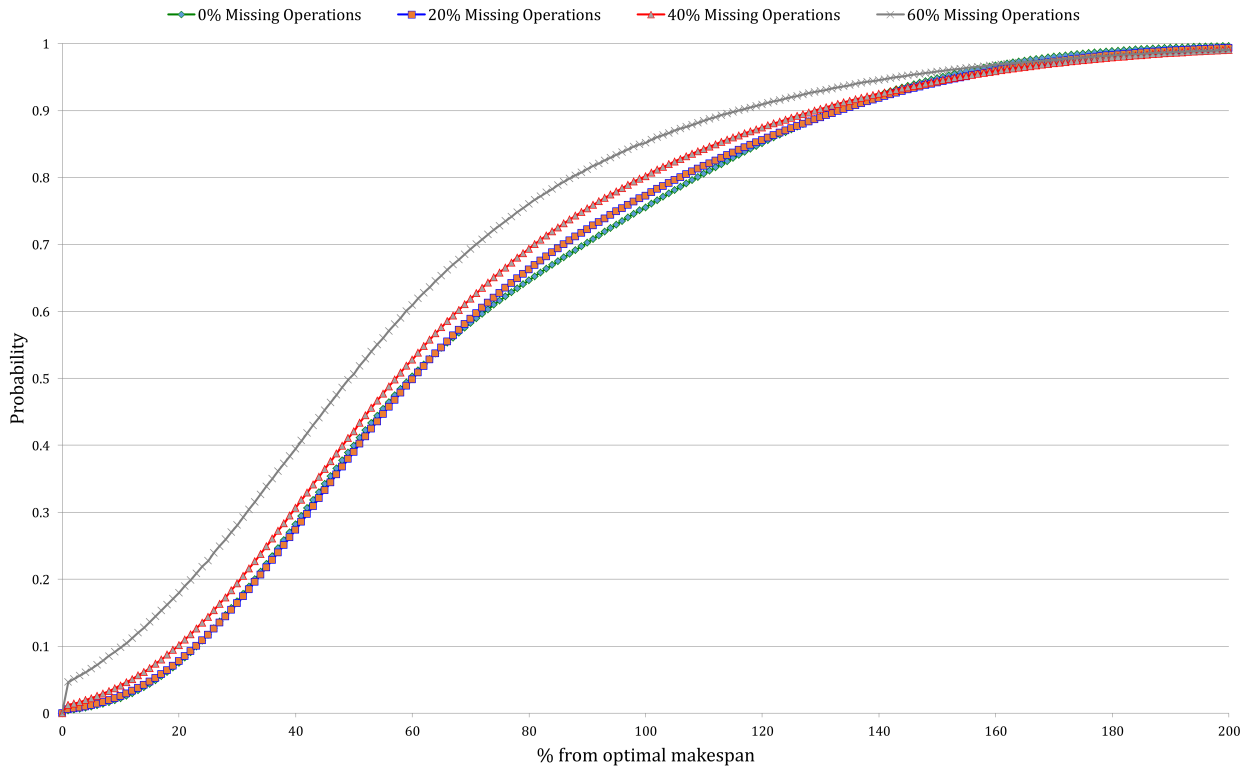

Figure 4: Cummulative distribution function for $m_j = 2$ (missing operations)

behaviour is that, in the first two cases, the possibility of very highly loaded stages in the shop is much higher than in the other two cases, as processing times remain the same for all of them. More specifically, for $m_j = 1$ there is a high probability that any of the single-machine stages becomes very loaded, while in $m_j = [1, 5]$ it is foreseeable that those stages with a single machine, or even those with two, become the most loaded stages. Therefore, the existence of missing operations is an opportunity to alleviate the load of these stages. On the other hand, in the last two cases, the hardness increases with the number of missing operations because the existence of missing operations can cause an over-loading in some stages.

As a conclusion of the analysis carried out in the section, we have shown that, in general, the HFSMO problem is different than the classical HFS problem, so it makes sense to use and develop heuristics specifically designed for this particular case. This is carried out in the next section.

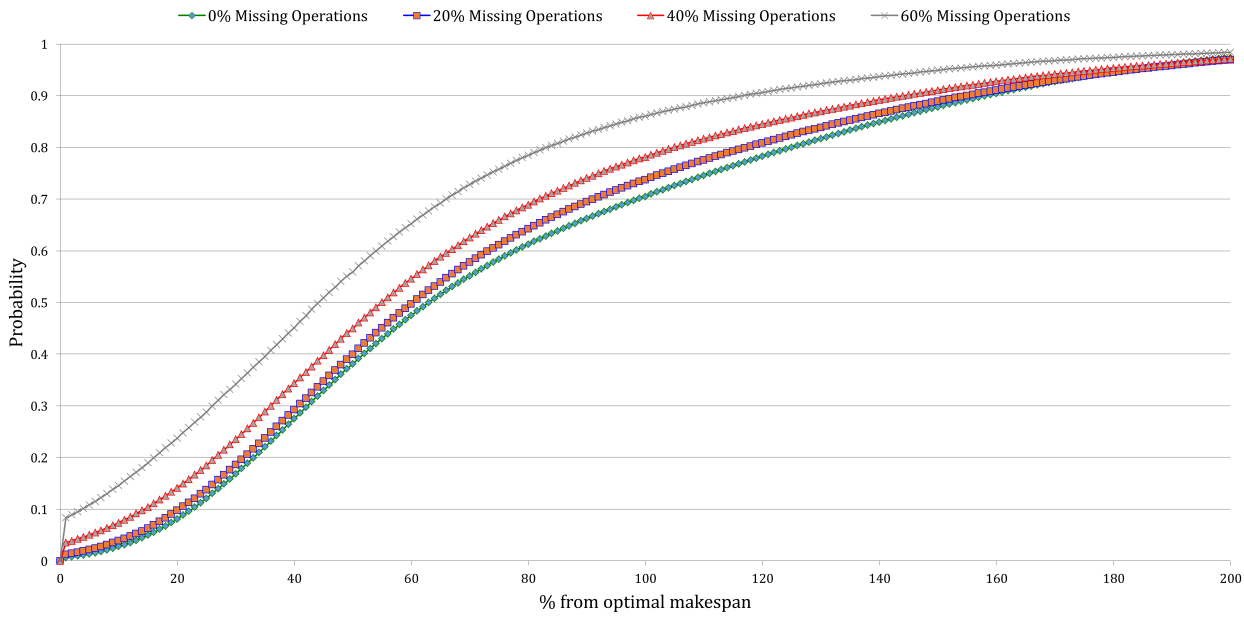

Figure 5: Cummulative distribution function for $m_j = U\,[2, 4]$ (missing operations)

4 Heuristics Proposal

The aim of this section is to develop appropriate procedures for the HFSMO problem that take advantage of the special features of the missing operations. To do so, we:

1. Propose a number of modifications of well-known dispatching rules employed for the HFS problem.

2. Propose four sets of improvement heuristics specifically conceived for the HFSMO problem.

Regarding the dispatching rules, the following extensions are proposed:

- SPTswm (SPT - stages without missing operations) and LPTswm (LPT - stages without missing operations). The jobs are sorted according to the ascending (descending) sum of processing times divided by the number of stages without missing operations. This approach attempts to remove the effect of zero-processing times.

- SPTw (SPT - weighted) and LPTw (LPT - weighted). These two rules first apply a weight to every stage depending on the number of machines in each stage and then

sort the jobs according to the ascending or descending weighted sum of processing times. The idea is to take into account the processing capacity of each stage, favouring those jobs with higher processing times in stages with more capacity and those with shorter processing times, respectively.

- SPTwswm (SPT - weighted stages without missing operations) and LPTwswm (LPT - weighted stages without missing operations). The previous dispatching rules are mixed, so we apply the factors referring to the number of stages without missing operations and to the number of machines per stage.

- A backward version of all the above rules (as done e.g. in Pan et al., 2014), where jobs are assigned to machines starting from the last stage instead of starting from the first and then they are assigned to the preceding stages by computing their backward release times. To denote the backward version of the dispatching rules we add a b at the beginning of the name, e.g. bSPTswm is the backward version of the SPTswm rule.

Regarding improvement heuristics, we propose the following DFF_N(a), DFF_R(a), DFF_N(100)_RI, and DFF_SD. Firstly, DFF_N(a) and DFF_R(a) try to improve the solutions obtained by the NEH and Raj heuristics by searching in the neighbourhood of jobs with missing operations. More specifically, let Π_{NEH} and Π_{Raj} be the sequences obtained by the NEH heuristic (see pseudocode in Figure 6) and Rajendran heuristic (see pseudocode in Figure 7) respectively. DFF_N(a) uses Π_{NEH} as its initial solution whereas DFF_R(a) uses Π_{Raj}. Additionally, let q^j be the number of missing operations in stage j and let $\Delta^j := \{\delta_1^j, \ldots, \delta_{q^j}^j\}$ be the set of jobs containing missing operations in stage j. Then, the proposed heuristics first compute Δ^j for each stage j and the initial sequences of jobs, denoted as Π. After that, each job δ_i^j is tested in both the first and last $a/2$ positions of iteration sequence (denoted as Π), updating the solution if the best makespan is improved. A detailed procedure of the heuristic DFF_N(a) (DFF_R(a)) is explained

below:

Step 1 Let Π_{NEH} be the initial solution of DFF_N(a), i.e. Π, (or equivalently, let Π_{Raj} be the initial solution of DFF_R(a))

Step 2 Perform Steps 3-6 for all stages ($j = \{1, \ldots, s\}$)

Step 3 Perform Steps 4-6 for all jobs ($i = \{1, \ldots, q^j\}$)

Step 4 Remove job δ_i^j from Δ^j.

Step 5 Insert job δ_i^j in the first and last $a/2$ positions of sequence Π and denote the new sequence of jobs as Π^k, $\forall k = \{1, ..., a\}$.

Step 6 Compute the objective function for each Π^k, $\forall k = \{1, ..., a\}$. If the best solution has been improved, Π is updated as the best sequence among Π^k, $\forall k = \{1, ..., a\}$.

%(NEH Heuristic)

Generate a job sequence $\Pi := [\pi_1, \pi_2, \cdots, \pi_n]$ using dispatching rule LPT

for $i = 1$ **to** n **do**

Take job π_i from Π

Move job π_i to all possible positions in Π and calculate makespan

Select the sequence with minimum makespan

end

Return Π

Figure 6: NEH heuristic pseudocode

%(Rajendran Heuristic)

Generate a job sequence $\Pi := [\pi_1, \pi_2, \cdots, \pi_n]$ using dispatching rule LPT

Place first job in the partial sequence Π^*, i.e. $\Pi^* = (\pi_1)$

Number of jobs in π^* equal to 1 ($n^* = 1$)

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

Calculate $lb = |(n^* + 1)/2|$ and $ub = (n^* + 1)$

Remove next job from Π

Insert the removed job in Π^* in the $lb < \rho < ub$ positions

Evaluate makespan of partial sequences

Select sequence with minimum makespan and update Π^*

$n^* = n^* + 1$

end

Return Π

Figure 7: Rajendran heuristic pseudocode

In order to explain the DFF_N procedure with a small example, let us assume a hybrid flowshop composed of three stages with two machines in the first stage, one machine in the second stage and two machines in the last stage. A set of four jobs whose processing times are depicted in Figure 8 is to be scheduled. As we can see, missing operations appear both in second and third stages. In this example we can see how a reduction in the makespan can be achieved by applying the proposed heuristic. In the left part of Figure 8, it can be seen the result of applying the NEH heuristic to the problem, obtaining a makespan value of 17 for sequence $\Pi = (2, 1, 4, 3)$. In the right part, it can be seen that an improvement of the makespan can be obtained by applying one loop of the new heuristic. In this example we move the job with a missing operation in the last stage (job 3), and try to insert it in the four possible positions, obtaining makespan values of 19,18,17 and 16. Therefore, we select the last possible sequence $\Pi = (2, 1, 3, 4)$ with an improved makespan.

Finally, regarding the DFF_N(100)_RI, and DFF_SD proposals, they are two modi-

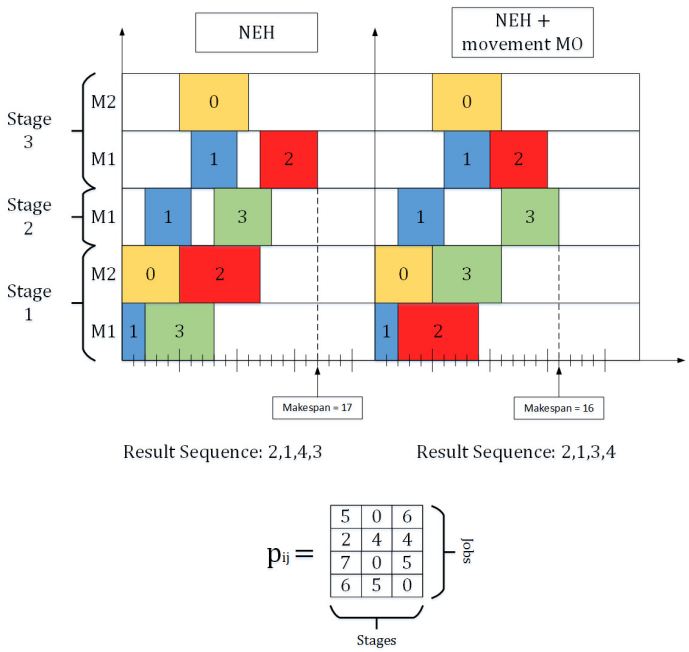

Figure 8: Example of DFF_N

fications of DFF_N(a). On the one hand, DFF_N(100) with Random Insertions (denoted as DFF_N(100)_RI) works in the same manner as DFF_N(100), but for each insertion of a job containing missing operations there is a small probability (10%) of selecting a job without missing operations to be inserted. With this proposal we want to prove whether a better solution can be obtained by considering exclusively jobs with missing operations, or if we can obtain a better solution by including also jobs without missing operations. On the other hand, we also propose the DFF Stage Dependent heuristic (DFF_SD), which is similar to DFF_N(100), but instead of inserting the jobs in all possible positions, insertions are dependent on the stage where the missing operations appear, i.e. if a job contains missing operations in the first $\frac{s}{2}$ stages, the insertions are only made in the first $\frac{n}{2}$ positions of the sequence, and if the job contains the missing operations in the second $\frac{s}{2}$ stages, the insertions are made in the second $\frac{n}{2}$ positions of the sequence. The aim of this proposal is to know whether it is worth to insert jobs in all possible positions, or the improvement is dependent on the stage where missing operations exists.

5 Computational experience

In this section we analyse the performance of our proposals using an extensive set of instances. We perform a computational evaluation by comparing the proposed approximate methods with a set of heuristics from the literature. By doing so, we determine the actual set of efficient heuristics for the problem under consideration. The section is organised as follows: the set of instances and the indicators to compare the algorithms are described in Subsection 5.1; the algorithms implemented in this study are described in Subsection 5.2; the results of the computational experiences are shown in Subsections 5.3 and 5.4 for dispatching rules and improvement heuristics, respectively.

5.1 Benchmark and Indicators

To compare the different approximate procedures, an extensive benchmark composed of 4,800 instances is designed, varying both the number of jobs and stages. More specifically, $n \in \{5,\ 10,\ 20,\ 30,\ 40,\ 50,\ 100,\ 200\}$. In line with the benchmark by Carlier and Neron (2000), we use $s \in \{3, 5, 10, 15, 20\}$. 30 instances are generated for each combination of the factors. The processing times are assumed to be drawn from a uniform distribution between 1 and 99. Regarding the percentage of missing operations, we use 0%, 20%, 40% and 60% as discussed in Section 3. Finally, with respect to the number of machines per stage, we assume $m_j = U[1, 5]$. A summary of the values of the parameters is shown in Table 2.

The results are evaluated by means of the average Relative Percentage Deviation (RPD) and the Average Computational Time (ACT) in seconds:

$$RPD_h = \frac{C_{\max_j} - C^{best}_{\max}}{C^{best}_{\max}} \cdot 100$$

$$ACT_h = \frac{\sum_{\forall s} T_{h,s}}{S}$$

Table 2: Test bed for performance analysis

Parameter	Values
n	5,10,20,30,40,50,100,200
s	3,5,10,15,20
p_{ij}	$U[1,99]$
% Missing Operations	0%,20%,40%,60%
m_j	$U[1,5]$
Number of Instances per combination of n and s	30

where RPD_h is the Relative Percentage Deviation of heuristic h for an specific instance, $C_{\max_h}$ is the makespan obtained by heuristic h for that instance, $C_{\max}^{best}$ is the best makespan obtained by any heuristic in that instance, $T_{h,s}$ is the CPU time in seconds of heuristic h in instance s, and S is the total number of instances.

5.2 Implemented algorithms

We test a set of 56 different heuristics classified into 2 groups: dispatching rules (24) and improvement heuristics (32). Among the dispatching rules, we use the proposals discussed in Section 4, i.e. SPTswm, LPTswm, SPTw, LPTw, SPTwswm, LPTwswm, bSPTswm, bLPTswm, bSPTw, bLPTw, bSPTwswm, and bLPTwswm. In addition, the following basic dispatching rules are tested:

- Shortest (longest) Processing Time, SPT (LPT): The jobs are sorted according to the sum of their duration in each stage, $p_i = \sum_{\forall j} p_{ij}$, in ascending (descending) order.

- Shortest (longest) Processing Time at the First Stage, SPTF (LPTF). The jobs are sorted according to their duration in the first stage, $p_i = p_{i1}$, in ascending (descending) order.

- Shortest (longest) Processing Time in the Bottleneck stage, SPTB (LPTB). The jobs are sorted according to their duration in the stage with the highest sum of processing

times for all jobs, $p_i = p_{ib}$ with $b := \arg\max_j\{\sum_{\forall i} p_{ij}\}$, in ascending (descending) order.

We also added the backward version associated to all dispatching rules as commented in Section 4.

Regarding improvement heuristics, we include in the comparisons the four existing heuristics for related scheduling problems (HFS and PFSP) discussed in Section 2 (i.e. NEH, Raj, WT1_NEH(x), and WT2_NEH(x)), and the best performing metaheuristics for the HFSMO problem, i.e. the simulated annealing algorithm by Saravanan et al. (2014) and IHGSS. More specifically, we compare:

- The original NEH heuristic by applying the above dispatching rules as initial solutions. We denote them in the following as NEH_y, where y is the dispatching rule used as initial solution. A total of 6 heuristics are generated in this manner.
- The original Raj heuristic by applying all previously described dispatching rules as initial solutions. We denote these heuristics as Raj_y, where y is the dispatching rule used as initial solution. A total of 6 heuristics are obtained.
- The heuristics WT1_NEH(x) and WT2_NEH(x) by Kizilay et al. (2015), where the parameter x is set to be equal to the number of jobs (n), as this case provides the best results in a series of preliminary experiments. Let us denote them in the following as WT1_NEH and WT2_NEH respectively.
- The Simulated Annealing algorithm by Saravanan et al. (2014) (henceforth SA_Saravanan).
- The IHGSS by Marichelvam and Prabaharan (2014) (henceforth IHGSS_Marichelvam).

All these heuristics are re-implemented to be compared against our proposals, i.e.: DFF_N(a), DFF_R(a), DFF_N(100)_RI, and DFF_SD, with $a \in \{20, 40, 80, 100\}$. All heuristics have been coded using the same programming language (C#), and the set of

instances was executed under the same conditions, i.e. using the same computer (Intel Core i7-3770 with 3.4 GHz and 16 GB RAM) and the same libraries. In the next subsections we present the results of the computational experience, first regarding the dispatching rules, and finally regarding the improvement heuristics.

5.3 Dispatching rules

In Table 3 we show the results of the 24 dispatching rules for each level of missing operations, as well as the aggregated results. The table shows the ranking of the dispatching rules according to ARPD (we focus only on ARPD as ACT is almost negligible for all heuristics). Note that, as we are presenting the ARPD for instances with no missing operations, the shaded cells provide the same results as the cells just above.

For those instances with missing operations, there are 3 dispatching rules dominating the other 19: bSPTB, bSPT and LPT. In Table 4 we present the numerical results according to the size of the instances, classifying them into big (100 and 200 jobs), medium (30, 40 and 50 jobs), and small instances (5, 10 and 20 jobs). Although the overall best performing dispatching rules is bSPTB, depending on the size of the instances and on if only missing operations or all jobs are considered together, in some cases LPT performs better than bSPT.

5.4 Improvement Heuristics

The results of the 32 improvement heuristics are summarised in Table 5, and in Figures 9 and 10. Results for all instances are presented in Figure 9, and in the second and third columns of Table 5. Instances with missing operations are shown in Figure 10, and in the fourth and fifth columns of Table 5. The behaviour of the heuristics in both figures is similar, but the improvement obtained by the new heuristics in terms of ARPD is reduced when all instances are considered. This is foreseeable, as our heuristics have the same performance as other heuristics when there are no missing operations. Taking this into

Table 3: Performance of dispatching rules

	0% Missing Operations		20% Missing Operations		40% Missing Operations		60% Missing Operations		All instances	
Ranking	Disp. Rule	ARPD	Disp. Rule	ARPD	Disp. Rule	ARPD	Disp. Rule	ARPD	Disp. Rule	ARPD
1	bSPTB	8.46	bSPTB	12.10	bSPTB	12.01	bSPTB	12.33	bSPTB	11.22
2	bSPTF	10.69	LPT	12.21	bSPT	13.21	LPT	12.95	LPT	12.58
3	LPT	10.92	bSPT	12.84	LPT	14.25	bSPT	13.67	bSPT	12.76
4	LPTswm	10.92	bSPTw	13.55	bSPTw	15.07	LPTswm	14.79	bSPTw	13.94
5	bSPT	11.33	LPTw	13.63	bSPTswm	15.09	bSPTswm	15.31	bSPTswm	14.03
6	bSPTswm	11.33	bSPTwswm	14.31	LPTswm	15.58	bSPTw	15.35	LPTswm	14.05
7	bSPTwswm	11.74	bSPTswm	14.39	LPTw	15.86	LPTw	15.80	LPTw	14.42
8	bSPTw	11.79	LPTswm	14.91	bSPTwswm	15.99	bSPTwswm	16.99	bSPTwswm	14.76
9	LPTwswm	12.05	LPTwswm	15.45	LPTwswm	17.08	bSPTF	17.00	bSPTF	15.11
10	LPTw	12.38	bSPTF	15.49	bSPTF	17.28	LPTwswm	17.25	LPTwswm	15.46
11	SPTF	12.78	SPTF	17.84	SPTF	19.40	SPTF	17.26	SPTF	16.82
12	SPTB	12.89	SPTB	19.30	SPTB	20.63	SPTB	19.06	SPTB	17.97
13	bLPT	17.98	LPTB	20.93	bLPTswm	21.63	bLPTswm	20.16	bLPTswm	20.77
14	bLPTswm	17.98	SPTswm	22.31	bLPTwswm	22.02	bLPTwswm	20.71	bLPTwswm	20.95
15	bLPTw	18.01	bLPTwswm	22.75	SPTswm	22.60	SPTswm	21.75	SPTswm	21.4
16	bLPTwswm	18.31	bLPTswm	23.31	SPTwswm	22.88	SPTwswm	22.15	SPTwswm	21.86
17	SPTw	18.83	SPTwswm	23.47	LPTB	24.40	LPTB	24.95	LPTB	22.37
18	SPTwswm	18.93	bLPTF	24.82	bLPTF	25.57	bLPT	25.08	bLPT	23.95
19	SPT	18.97	SPT	25.59	bLPT	26.65	bLPTF	25.54	bLPTF	23.98
20	SPTswm	18.97	LPTF	25.67	SPT	27.07	bLPTw	26.00	bLPTw	24.46
21	LPTB	19.18	bLPTw	25.78	bLPTB	27.15	SPT	26.79	SPT	24.6
22	bLPTF	19.99	bLPT	26.10	SPTw	27.85	bLPTB	27.18	SPTw	25.07
23	LPTF	20.13	SPTw	26.21	bLPTw	28.06	SPTw	27.41	bLPTB	25.39
24	bLPTB	20.61	bLPTB	26.62	LPTF	30.58	LPTF	33.32	LPTF	27.43

Table 4: Numerical results (ARPD) for best performing dispatching rules

Size	Type	LPT	bSPTB	bSPT
BIG (100, 200 jobs)	**All Instances**	2.987	**1.923**	3.002
	Only Missing Operations	2.812	**2.044**	2.801
MEDIUM (30, 40, 50 jobs)	**All Instances**	8.541	**6.278**	8.056
	Only Missing Operations	8.704	**6.912**	8.192
SMALL (5, 10, 20 jobs)	**All Instances**	23.011	**22.372**	23.977
	Only Missing Operations	24.440	**24.117**	25.249

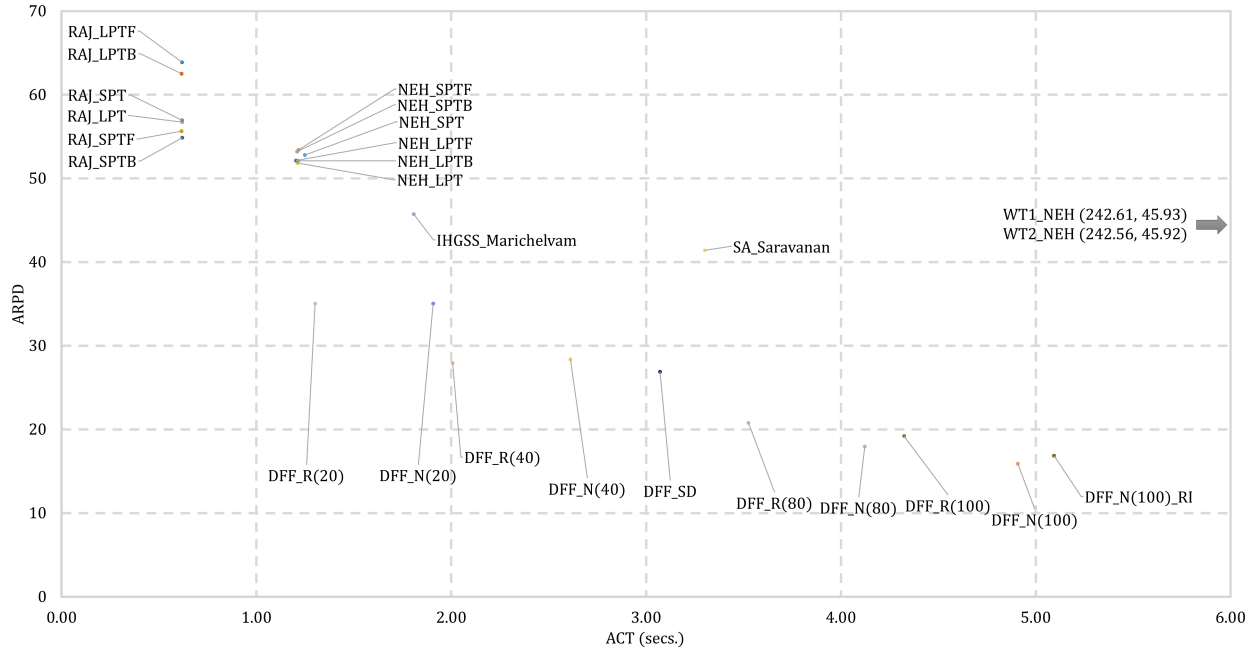

Figure 9: Results of the heuristics for all instances

account, we focus on the results of the instances with missing operations (see Figure 10). We define a number of groups according to their ACT to simplify the comparison between heuristics.

A number of conclusions can be drawn from our comparison, which are statistically justified by means of Holm's procedure (Holm, 1979). In this procedure, each hypothesis is analysed using a non-parametric Mann-Whitney test (see e.g. Fernandez-Viagas and Framinan, 2015). We first sort the hypotheses in non descending order of the p-values found in the test. Next, we check all hypotheses and reject hypothesis i if its p-value is

Table 5: ACT_h and $ARPD_h$ for each heuristic for all instances (second and third columns) and for the set of instances with missing operations (fourth and fifth columns).

Heuristic	All instances		Instances with missing operations	
	ACT_h	$ARPD_h$	ACT_h	$ARPD_h$
RAJ_LPT	0.62	56.72	0.62	65.58
RAJ_LPTF	0.62	63.89	0.62	74.02
RAJ_LPTB	0.62	62.52	0.62	72.34
RAJ_SPT	0.62	56.95	0.62	65.90
RAJ_SPTF	0.62	55.65	0.62	64.51
RAJ_SPTB	0.62	54.86	0.62	63.48
NEH_SPT	1.25	52.79	1.25	61.38
NEH_SPTF	1.22	53.41	1.22	62.17
NEH_SPTB	1.21	53.22	1.21	61.84
NEH_LPT	1.21	51.83	1.22	60.34
NEH_LPTF	1.20	52.12	1.20	60.56
NEH_LPTB	1.21	52.09	1.22	60.62
IHGSS_Marichelvam	1.81	45.74	1.85	51.79
SA_Saravanan	3.30	41.41	3.38	47.71
WT1_NEH	242.61	45.93	243.55	53.94
WT2_NEH	242.56	45.92	243.67	53.93
DFF_R(20)	1.30	35.07	1.54	37.31
DFF_R(40)	2.01	27.94	2.48	27.87
DFF_R(80)	3.53	20.77	4.51	18.44
DFF_R(100)	4.32	19.22	5.57	16.40
DFF_N(20)	1.91	35.02	2.16	38.51
DFF_N(40)	2.61	28.36	3.10	29.70
DFF_N(80)	4.12	17.97	5.11	15.98
DFF_N(100)	4.91	15.90	6.16	13.28
DFF_SD	3.07	26.88	3.70	27.72
DFF_N(100)_RI	5.09	16.86	6.40	14.55

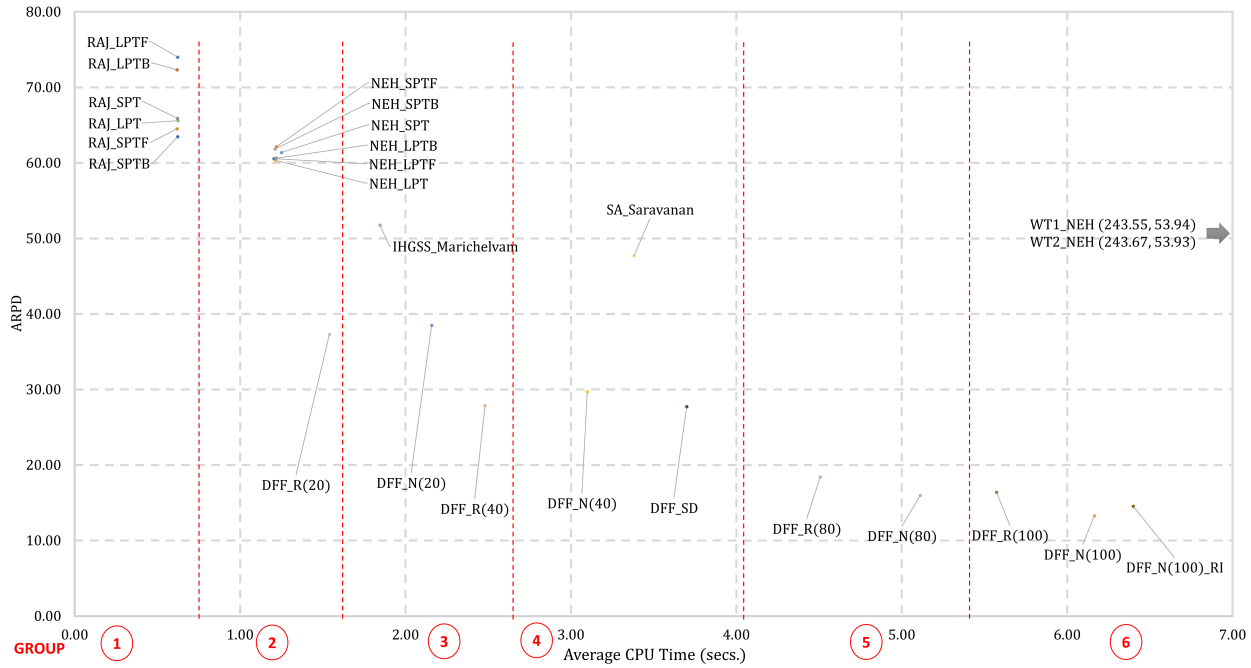

Figure 10: Results of the heuristics for instances with missing operations

lower than $\frac{\alpha}{(k-i+1)}$, where k is the total number of hypotheses. The results of this procedure are presented in Table 6. We also present the mean and least significance difference (LSD) 95% intervals in Figure 11.

The following conclusions for the groups (and the hypotheses for the subsequent statistical analysis) can be obtained:

- Group 1. All Raj heuristics are included, as they achieve the shortest computation times. However, their performance is not good in terms of ARPD. To analyse the statistical significance of these heuristics, we check if the following two hypotheses hold: in H_1 we compare the best performing Raj heuristic (Raj_SPTB) with the second one (Raj_SPTF) and in H_2 we compare Raj_SPTB with the worst performing one (Raj_LPTF) to analyse the influence of the initial solution in the heuristic. According to the results in Table 6, the first hypothesis cannot be rejected, meaning that we cannot assure that there is a significant difference between the two heuristics. Nevertheless the second hypothesis can be rejected, i.e. the best and worst initial sequence are significantly different.

Table 6: Holm's procedure

H_i	Hypothesis	p-value	**Mann-Whitney**	$\frac{\alpha}{k-i+1}$	**Holm's rocedure**
H_2	Raj_SPTB = Raj_LPTF	0.000	R	0.0031	R
H_4	NEH_LPT = NEH_bSPTF	0.000	R	0.0033	R
H_5	NEH_LPT = DFF_R(20)	0.000	R	0.0035	R
H_6	DFF_R(40) = DFF_N(20)	0.000	R	0.0038	R
H_7	DFF_R(40)= IHGSS_Marichelvam	0.000	R	0.0042	R
H_9	DFF_SD = SA_Saravanan	0.000	R	0.0045	R
H_{10}	DFF_N(80) = DFF_R(80)	0.000	R	0.0050	R
H_{11}	DFF_N(100) = DFF_R(100)	0.000	R	0.0056	R
H_{12}	DFF_N(100) = DFF_N(100)_RI	0.000	R	0.0062	R
H_{13}	DFF_N(100) = WT1_NEH	0.000	R	0.0071	R
H_{14}	DFF_N(100) = WT2_NEH	0.000	R	0.0083	R
H_{15}	DFF_R(20) = IHGSS_Marichelvam	0.000	R	0.0100	R
H_{16}	DFF_R(40) = SA_Saravanan	0.000	R	0.0125	R
H_1	Raj_SPTB = Raj_SPTF	0.194		0.0167	
H_8	DFF_SD = DFF_N(40)	0.260		0.0250	
H_3	NEH_LPT = NEH_LPTF	0.769		0.0500	

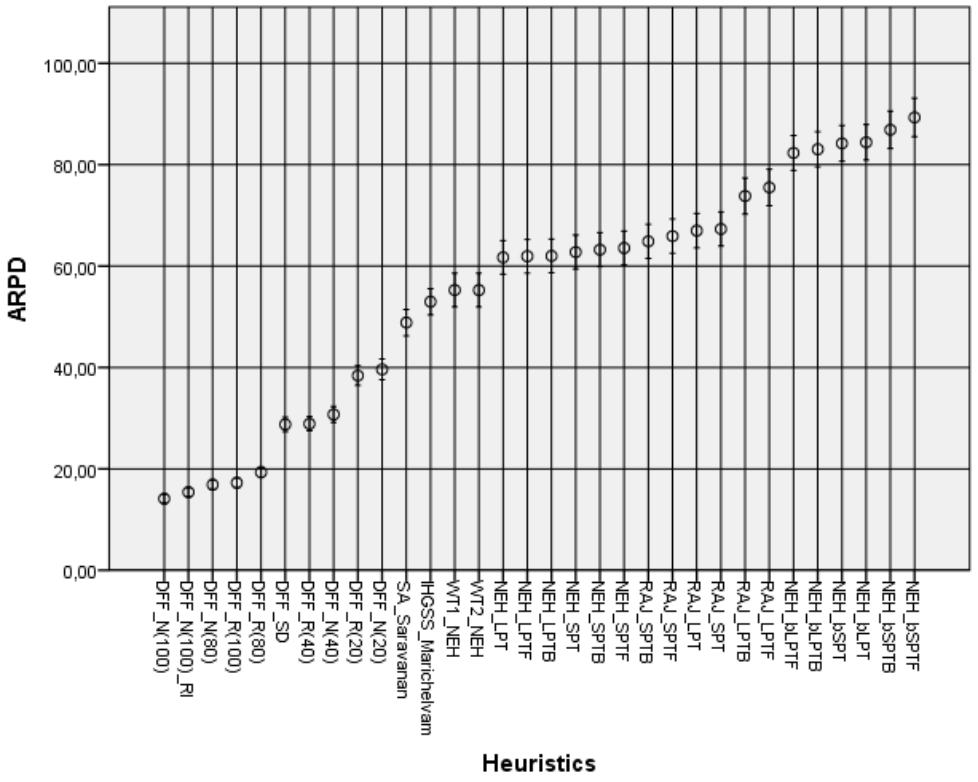

Figure 11: Mean and LSD intervals 95% for RPD

- Group 2. We include the versions of the NEH heuristic and DFF_R(20). The first conclusion is that the NEH heuristics initialised with backward dispatching rules do

not perform well for the problem under consideration, obtaining the worst ARPD and a non-competitive ACT. Among NEH heuristics, the best performing is NEH_LPT, closely followed by NEH_LPTB and NEH_LPTF. DFF_R(20) performs better with respect to ARPD, although it has a larger ACT.

To check these conclusions we test the following three hypotheses, i.e.

- in H_3 we compare the best performing NEH heuristic (NEH_LPT) with the second one (NEH_LPTF);
- in H_4 we compare the best performing heuristic (NEH_LPT) with the worst performing one (NEH_bSPTF);
- and in H_5 we compare the best NEH heuristic (NEH_LPT) with one of our proposals (DFF_R(20)).

According to Table 6, the same results are found as for the Raj_y heuristics in the first two hypotheses, i.e. the first hypothesis cannot be rejected and the second one can be rejected. The H_5 hypothesis can be rejected according to the results. Therefore, there are significant differences between the best NEH and our proposal DFF_R(20), being our proposal the best performing heuristic of this group.

- Group 3. This group includes the metaheuristic IHGSS_Marichelvam and the heuristics DFF_N(20) and DFF_R(40), and. We can conclude from Figure 10 that the best performing one is DFF_R(40) although it takes more ACT than DFF_(20). IHGSS_Marichelvam takes less time than the other two heuristics but its ARPD is also higher. To assert that the DFF_R(40) heuristic is the best perfoming one, we analyse the following hypotheses: in H_6 we compare DFF_R(40) and DFF_N(20); in H_7 our proposal DFF_R(40) is compared with IHGSS_Marichelvam. For both hypotheses, statistical significative differences have been found and, in light of these results, we can state that DFF_R(40) is the best performing heuristic of this group.

- Group 4. We consider here DFF_SD and DFF_N(40), and SA_Saravanan. From Figure 10 it can be seen that DFF_SD is the best performing heuristic of the group in terms of quality of solutions, i.e. ARPD, although DFF_N(40) has a lower ACT. It can be also seen that SA_Saravanan is not efficient. To check these conclusions we include two new hypotheses: In H_8 we compare DFF_SD and DFF_N(40) and we obtain that there are not significative differences between them. However, in H_9 we compare DFF_SD and SA_Saravanan, and we find that their ARPD values are significatively different, indicating that our proposal is efficient and improves the previous state-of-the art algorithm of this group.

- Group 5. In this group we include DFF_N(80) and DFF_R(80). It turns out that DFF initialised with NEH heuristic performs better than when initialised with Raj (see Figure 10), although it needs more CPU time. From Table 6, it can be seen that hypothesis H_{10} can be rejected, i.e. the heuristics are significatively different, being DFF_N(80) the best performing heuristic of the group.

- Group 6. This group includes DFF_R(100), DFF_N(100), DFF_N(100)_RI, WT1_NEH and WT2_NEH. Among them, DFF_N(100) obtains the best results in terms of ARPD, and is second in terms of ACT after DFF_R(100). We compare DFF_N(100) against the other 4 heuristics in this group, as can be seen in hypotheses H_{11}, H_{12}, H_{13}, and H_{14}. From Table 6, it can be seen that the hypotheses can be rejected, meaning that there are significant differences between these heuristics. Therefore we conclude that our proposal DFF_N(100) is the best performing one.

We also compare the already existing heuristics for the problem under study, i.e. IHGSS_Marichelvam and SA_Saravanan, with the new proposals with better ARPDs for lower ACTs, i.e. DFF_R(20) and DFF_N(40) respectively. It can be seen that the new proposals performs better in both cases. To confirm this conclusion we include hypotheses H_{15} and H_{16} and, according to Table 6, it can be seen thats there are significative

differences among them.

6 Conclusions and Further Research

In this paper, an analysis of the effect of missing operations on the hybrid flowshop problem with makespan minimisation criterion has been carried out. First, we study the empirical hardness of this problem as compared to the classical HFS. The analysis shows that, depending on the prevalence of missing operations in the jobs, the problem is different than the classical HFS without missing operations. We also analyse a number of different factors (number of jobs, number of stages, etc.) influencing the structure of solutions of the problem.

To take advantage of the special behaviour of the missing operations, we develop a number of dispatching rules and improvement heuristics that are compared with already existing ones both for the special case with missing operations, and for the general HFS case. From this comparison, we found that our proposals perform more efficiently than existing heuristics, therefore representing a new state-of-the art for the problem.

A number of open research issues can be addressed from this work. First, regarding the empirical hardness analysis, an in-depth study of different processing times could be interesting. Furthermore, we assume that missing operations can appear in any stage of the shop. It could therefore be interesting to study different patterns in the appearance of missing operations, as it is more realistic to assume that only some stages would have missing operations.

Acknowledgements

This research has been funded by the Spanish Ministry of Science and Innovation, under the project "PROMISE" with reference DPI2016-80750-P.

References

Armentano, V. A. and Ronconi, D. P. (1999). Tabu search for total tardiness minimization in flowshop scheduling problems. *Computers and Operations Research*, 26(3):219–235.

Behnamian, J. and Fatemi Ghomi, S. (2011). Hybrid flowshop scheduling with machine and resource-dependent processing times. *Applied Mathematical Modelling*, 35(3):1107–1123.

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

Carlier, J. and Neron, E. (2000). An exact method for solving the multi-processor flowshop. *RAIRO - Operations Research*, 34:1–25.

Fernandez-Viagas, V. and Framinan, J. M. (2015). 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.

Framinan, J. M., Ruiz Usano, R., and Leisten, R. (2001). Sequencing CONWIP flow-shops: analysis and heuristics. *International Journal of Production Research*, 39(12):2735–2749.

Gupta, J. N. (1988). Two-stage, hybrid flowshop scheduling problem. *Journal of the Operational Research Society*, 39(4):359–364.

Holm, S. (1979). A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6(2):65–70.

Kizilay, D., Tasgetiren, M., Pan, Q.-K., and Wang, L. (2015). An iterated greedy algorithm for the hybrid flowshop problem with makespan criterion. *IEEE Symposium Series on Computational Intelligence - CIPLS 2014, Proceedings*, 16-23.

Kurz, M. and Askin, R. (2003). Comparing scheduling rules for flexible flow lines. *International Journal of Production Economics*, 85(3):371–388.

Leisten, R. and Kolbe, M. (1998). A note on scheduling jobs with missing operations in

permutation flow shops. *International Journal of Production Research*, 36(9):2627–2630.

Leon, V. and Ramamoorthy, B. (1997). An adaptable problem-space-based search method for flexible flow line scheduling. *IIE Transactions*, 29(2):115–125.

Linn, R. and Zhang, W. (1999). Hybrid flow shop scheduling: a survey. *Computers and Industrial Engineering*, 37(1):57–61.

Marichelvam, M. and Prabaharan, T. (2014). Performance evaluation of an improved hybrid genetic scatter search (IHGSS) algorithm for multistage hybrid flow shop scheduling problems with missing operations. *International Journal of Industrial and Systems Engineering*, 16:120–141.

McCormick, S., Pinedo, M. L., Shenker, S., and Wolf, B. (1989). Sequencing in an assembly line with blocking to minimize cycle time. *Operations Research*, 37(6):925–935.

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

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

Nawaz, M., Enscore Jr., E., and Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, 11(1):91–95.

Pan, Q.-K., Wang, L., Li, J.-Q., and Duan, J.-H. (2014). A novel discrete artificial bee colony algorithm for the hybrid flowshop scheduling problem with makespan minimisation. *Omega*, 45(0):42 – 56.

Perez-Gonzalez, P. and Framinan, J. (2015). Single machine interfering jobs problem with flowtime objective. *Journal of Intelligent Manufacturing*, doi:10.1007/s10845-015-1141-6.

Rajendran, C. (1993). Heuristic algorithm for scheduling in a flowshop to minimize total flowtime. *International Journal of Production Economics*, 29(1):65–73.

Rajendran, C. and Ziegler, H. (2001). A performance analysis of dispatching rules and a heuristic in static flowshops with missing operations of jobs. *European Journal of Operational Research*, 131(3):622 – 634.

Ribas, I., Leisten, R., and Framinan, J. M. (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.

Ruiz, R., Serifoglu, F., and Urlings, T. (2008). Modeling realistic hybrid flexible flowshop scheduling problems. *Computers and Operations Research*, 35(4):1151–1175.

Ruiz, R. and Vazquez-Rodriguez, J. A. (2010). The hybrid flow shop scheduling problem. *European Journal of Operational Research*, 205(1):1 – 18.

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

Saravanan, M., Sridhar, S., and Harikannan, N. (2014). Optimization of realistic multi-stage hybrid flow shop scheduling problems with missing operations using meta-heuristics. *International Journal of Engineering and Technology*, 6(1):484–496.

Sridhar, J. and Rajendran, C. (1993). Scheduling in a cellular manufacturing system: a simulated annealing approach. *International Journal of Production Research*, 31(12):2927–2945.

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

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

Tseng, C.-T., Liao, C.-J., and Liao, T.-X. (2008). A note on two-stage hybrid flowshop scheduling with missing operations. *Computers and Industrial Engineering*, 54(3):695–704.

Vignier, A., Billaut, J.-C., and Proust, C. (1999). Les problèmes d'ordonnancement de type flow-shop hybride : état de l'art. *RAIRO - Operations Research*, 33(2):117–183.