Journal of the Brazilian Society of Mechanical Sciences and Engineering (2018) 40:40
https://doi.org/10.1007/s40430-018-0980-4

TECHNICAL PAPER

Heuristic algorithms for scheduling hybrid flow shops with machine blocking and setup times

João Vitor Moccellin[1] · Marcelo Seido Nagano[2] · Anselmo Ramalho Pitombeira Neto[1] · Bruno de Athayde Prata[1]

Received: 30 January 2017 / Accepted: 27 November 2017 / Published online: 17 January 2018

Abstract
We investigate a new variant of the hybrid flow shop problem (HFSP) considering machine blocking and both sequence-independent and sequence-dependent setup times. Since the HFSP is NP-hard, we propose heuristic algorithms along with priority rules based on the traditional SPT and LPT rules. We carried out computational experiments on simulated problem instances to test the performance of the priority rules. The objective function adopted was makespan minimization, and we used the rate of success and the relative deviation as performance criteria. The results indicate superiority of LPT-based rules. On instances with sequence-independent setup times, the LP rule, which is based on the non-increasing sorting of total processing times, outperformed other rules in most tested instances. In instances with sequence-dependent setup times, the LPS rule, which is based on the non-increasing sum of processing time and average setup times, outperformed other rules in most tested instances.

Keywords Production scheduling · Machine blocking · Sequence-independent setup times · Sequence-dependent setup times

1 Introduction

In production planning of manufacturing systems, the efficient scheduling of jobs plays a key role for the achievement of low-cost and competitive advantage in businesses. Among the variants of scheduling problems in the literature, one that has attracted research efforts in recent years due to its practical importance is the hybrid flow shop problem (HFSP).

In the HFSP, a set of jobs with fixed production sequence are processed in a set of production stages, each one composed of multiple machines, in which at least one of these stages has more than one machine. According to Ebrahimi et al. [2], these machines can be classified in identical parallel machines, uniform parallel machines or unrelated parallel machines. The problem is to find a schedule which optimizes an objective function, usually makespan minimization, mean flowtime, or total tardiness. The HFSP is a NP-hard problem, since it is a generalization of the classical flow shop, which is known to be NP-hard for three or more production stages [5]. Therefore, obtaining optimal solutions in acceptable computational times is a difficult task when solving large real-world problems.

According to Ruiz and Vázquez-Rodríguez [16], the hybrid flow shop environment is found in several real-world applications, such as electronics, production of paper, textile and concrete, manufacturing of photographic films, civil construction projects, Internet service architectures and load transportation systems.

Traditionally, the setup time is considered as a part of the processing times. However, this assumption leads to problems in the scheduling process. Therefore, an explicit consideration for setup times has been considered [10, 12].

This paper aims at investigating a new variant of the hybrid flow shop problem considering both machine blocking and both sequence-independent and sequence-

Technical Editor: Fernando Antonio Forcellini.

✉ Bruno de Athayde Prata
baprata@ufc.br

[1] Department of Industrial Engineering, Federal University of Ceará, Campus do Pici - Bloco 714, Fortaleza CEP 60455-760, Brazil

[2] Department of Industrial Engineering, University of São Paulo, Av. Trabalhador São-Carlense, 400, São Carlos CEP 13566-590, Brazil

dependent setup times. According to our thorough literature review, this variant has not been reported yet, despite its practical importance. We propose a new constructive method for the problem, based on priority rules, which can provide good solutions with low computational effort.

The remainder of this paper is organized as follows: in Sect. 2, the literature review is presented, in Sect. 3, the scheduling problem treated in this paper is stated in Sect. 4, we propose a constructive algorithm; in Sect. 5, we discuss some results from computational experiments; finally, in Sect. 6 we draw some conclusions and suggestions for future works.

2 Literature review

The HFSP is a widely studied combinatorial optimization problem with several industrial applications. Linn and Zhang [8] carried out a survey on the HFSP in which a classification of studies in this area is proposed based on three categories: problems with two stages, problems with three stages and problems with more than three stages. Vignier et al. [17] present a state-of-the-art review on the HFSP divided in two sections: problems with two stages and general problems with k stages.

Kis and Pesch [7] performed a literature update paper reviewing the subsequent studies approaching the HFSP. Wang [18] and Quadt and Kuhn [14] also present literature reviews for the HFSP, which they call flexible flow shop scheduling problem and flexible flow line problem, respectively.

Numerous-research articles have been published on this topic. This study reviews research on the flexible flow shop scheduling problem (FFSSP) from the past and the present. The solution approaches reviewed range from the optimum to heuristics and to artificial intelligence search techniques. We not only discuss the details from the selected methods and compare them, but also provide insights and suggestions for future research.

Ruíz and Vasquéz-Rodríguez [16] present a literature review focused on solution procedures, discussing the features of the variants of the problem. In addition, they point out existing gaps and have suggested future works. Ribas et al. [15] offer a robust literature review with an innovative taxonomy for the HFSP.

Choong et al. [1] describe two metaheuristics for the HFSP: the first one mixes particle swarm optimization (PSO) with simulated annealing (SA), while the second one is a hybrid of PSO and tabu search (TS). The objective function is minimizing the completion time of all the tasks. The hybrid PSO-SA algorithm reached the best results. The authors suggest that new variants of the HFSP with precedence constraints are worth investigating.

Hidri and Haouari [6] study a variant of the HFSP with multiple centers in which each center has a set of identical machines working in parallel. These authors propose new lower and upper bounds for the mentioned variant. The computational results show an improvement of the existing bounds. These authors suggest the development of exact methods for solving the studied variant.

Mousavi et al. [11] present a bi-objective variant of the HFSP with sequence-dependent setup times. The objective functions are both the minimization of makespan and total tardiness. The authors propose a bi-objective heuristic (BOH) for the determination of a Pareto front approximation. The proposed BOH outperformed a multi-objective simulated annealing (MOSA). These authors suggest the incorporation of new realistic assumptions, such as machine availability constraints and unrelated parallel machines at each stage.

Elmi and Topaloglu [3] studied hybrid flow shop robotic cells with multiple robots. They have taken of account blocking constraints, multiple part types, unrelated parallel machines and machine eligibility constraints. A mixed integer linear programming model is proposed, which could not be solved satisfactorily. A simulated annealing algorithm is developed for providing high-quality solutions in acceptable computational times. The authors suggest the consideration of dual gripper multiple robots and stochastic processing times.

Luo et al. [9] approached the HFSP with the consideration of energy consumption of machines. They adopted two objectives: minimizing the makespan and the electric power cost. Three multi-objective metaheuristics are tested: multi-objective ant colony optimization (MOACO), non-dominated sorting genetic algorithm II (NSGAII) and strength Pareto evolutionary algorithm II (SPEAII). The results indicate that this multi-objective approach is valuable in practice, since high-energy operations can be shifted to off-peak periods, during which energy cost is lower.

Zhang et al. [19] cast the vehicle scheduling problem in a maritime terminal as a variant of HFSP with a special type of blocking constraints and multiple product families, considering the makespan as the objective to be minimized. A constructive heuristic is proposed for solving the problem under study with low computational times.

Fattahi et al. [4] studied a HFSP with assembly operations, considering the minimization of makespan as an objective. A branch-and-bound (B&B) algorithm is proposed for solving the problem and a greedy randomized adaptive search procedure (GRASP) construction phase algorithm is applied to obtain upper bounds. This hybridization reduced substantially the computational times. Although the B&B has shown promising results, the instances under consideration were small (7, 10, 15 or 25

jobs products; 2 production stages; 2, 3 or 4 machines per stage).

Ebrahimi et al. [2] investigated a HFSP with sequence-dependent family setup times. They adopted two objective functions: minimization of both makespan and total tardiness. Two multi-objective metaheuristics are tested for solving the proposed variant: a NSGAII and a multi-objective genetic algorithm (MOGA), that are compared with a multiphase genetic algorithm (MPGA). The NSGA II outperformed the other algorithms. The authors suggest that taking into account stochastic due dates makes this study closer to the real-world problems.

3 Problem statement

The problem of flow shop scheduling with multiple machines, also known as either hybrid flow shop problem (HFSP) or flow shop with parallel machines, consists of a multistage flow shop in which each stage $k \in \{1,\ldots, g\}$, $g \geq 2$ is composed of m_k parallel machines. Each machine is capable of processing a single job operation per turn. A multistage flow shop is considered a hybrid flow shop if at least one production stage has more than one machine.

The problem consists in scheduling a set of jobs $J = \{1,\ldots, n\}$ in which each job has only one operation in each stage so as to optimize one or more objective-functions. In this paper, we adopted as objective-function the makespan. The operations must be performed sequentially, passing through all stages. Other usual assumption considers that an operation once started cannot be either preempted or subdivided in simultaneous sub operations. Each job $j \in J$ has a known processing time p_{jk} corresponding to stage $k \in \{1,\ldots, g\}$. Figure 1 shows a general schematic representation of the HFSP with g production stages.

The HFSP may be seen as a combination of the classical flow shop, which has one machine in each production stage ($m_k = 1,\ g > 1$), with the problem of parallel machines with a single stage ($m \geq 2,\ g = 1$), which have been intensively studied. The HFSP is NP-hard even when only one stage has more than one machine. This fact prevents the application of exact methods to problems which arise in industry.

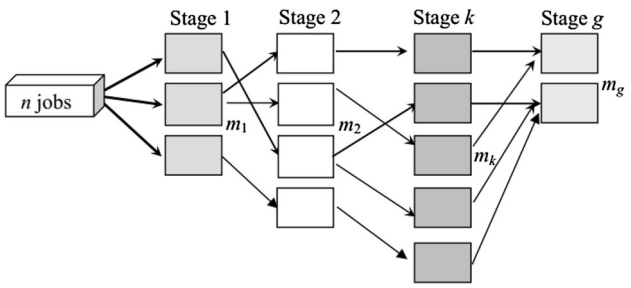

Fig. 1 Hybrid flow shop problem with g production stages

The least complex setting of the HFSP considers the machine setup times as already included in the operation processing times. This setting has received considerable attention from many researchers. However, this assumption can decrease the quality of solutions in many applications which require an explicit treatment of setup times. In general, the sum of setup and processing times is realistic when setup times are independent of the operation sequence or when variability is negligible. Even then, considering setup times as making part of the processing times, implicitly assumes a scheduling model in which the setup of a machine $M^{(k)}$ is made only after finishing the previous operation of the current job to be processed on such a machine, even if this machine is idle.

In this paper, we approached a variant of the classical HFSP, with an added degree of complexity, with the following characteristics:

- Machine setup times are explicitly separated from processing times;
- We consider both sequence-independent and sequence-dependent setup times;
- In contrast to the classical HFSP, machine setup is anticipated as soon as a machine has finished the processing of a job and is idle;
- The production system does not allow intermediate storage of jobs between production stages due to technological or operational constraints of any nature, which has as a consequence the blocking of machines.

Although the aforementioned characteristics are typical of many relevant production environments in industry, to the extent of our knowledge there has been little research on the performance of alternative scheduling policies.

Consider an illustrative example with five jobs, three machines, two machines per stage and three stages. The processing times and sequence-independent setup times are presented in the Tables 1 and 2, respectively. Figure 2 shows an illustrative solution for the variant under proposition, the sequence {1, 2, 3, 4, 5}, whose makespan is 280 time units. In Fig. 2 we also have two types of non-productive times: In gray we have setup times, while we have

Table 1 Processing times matrix for the presented example

Job	Machine type		
	1	2	3
1	44	46	45
2	37	46	35
3	31	45	25
4	45	49	39
5	48	26	42

Table 2 Sequence-independent setup times for the presented example

Job	Machine type		
	1	2	3
1	29	27	10
2	18	20	30
3	13	13	18
4	21	29	12
5	12	29	26

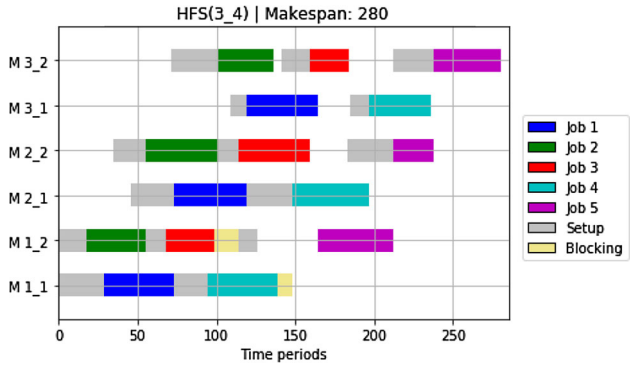

Fig. 2 A feasible solution for the proposed variant

machine blocking times in yellow, which represent the buffer zero constraints. A given machine cannot receive other jobs until the subsequent machine begins the effective processing of the current job.

4 Description of the proposed constructive heuristics

In this work, by virtue of the originality of studying the HFSP with sequence-dependent anticipated setup and machine blocking, we propose a constructive heuristic method based on priority rules. These rules define a job ordering to be followed during the scheduling, which is performed one job at a time. It is worth noting that the possibility of machine blocking prevents the scheduling by production stage, since the scheduling of jobs in a particular stage depends on the jobs already scheduled in the subsequent stage. Moreover, priority rules have great practical importance, since they are simple and provide solutions with little computational effort.

In practice, the most used priority rules are the following: shortest processing time (SPT), longest processing time (LPT), earliest due date (EDD), minimum slack time (MSL), first-in-first-out (FIFO) and last-in-last-out (LIFO). There are also weighted variants of these rules. It is well known (see e.g., [13]) that SPT rule minimizes the mean flow time for the classical single machine scheduling problem, and LPT rule is a heuristic for minimizing the makespan of jobs scheduled in the traditional parallel machines scheduling model. In Sects 4.1 and 4.2 below, we treat the cases for sequence-independent and sequence-dependent setup times separately.

4.1 Sequence-independent setup times

In this case, we propose seven priority rules, from which three of them are based on SPT rule, three ones subsequent are based on LPT and the remaining rule is a random one just for comparison purposes. At first, we describe some notations. Let:

- p_{jk}: processing time of job $j \in \{1, 2,\ldots, n\}$ in production stage $k \in \{1, 2,\ldots, g\}$; (All machines in stage k are identical ones);
- s_{jk}: setup time of a machine in stage $k \in \{1, 2,\ldots, g\}$ for processing job $j \in \{1, 2,\ldots, n\}$;
- $P_j = \sum_{k=1}^{g} p_{jk}$: sum of processing times of job j in all production stages;
- $S_j = \sum_{k=1}^{g} s_{jk}$: sum of setup times for job j in all production stages;
- $PS_j = \sum_{k=1}^{g} \left(p_{jk} + s_{jk}\right)$: sum of processing time and setup time for job j over all stages.

The proposed rules are described in Table 3, while Fig. 3 describes the proposed constructive heuristic for job scheduling.

As aforementioned, we use as objective function, the makespan.

4.2 Sequence-dependent setup times

In this case, we propose 11 priority rules, from which 5 are based on SPT rule and 5 are based on LPT one and the remaining rule is a random one just for comparison purposes. We add some notation to the one already described in Sect. 4.1. Let:

Table 3 Proposed priority rules for ordering jobs with sequence-independent setup times

#	Rule	Order jobs according to
1	LP	Non-increasing P_j
2	LS	Non-increasing S_j
3	LPS	Non-increasing PS_j
4	SP	Non-decreasing P_j
5	SS	Non-decreasing S_j
6	SPS	Non-decreasing PS_j
7	RAND	Random order

Fig. 3 Proposed constructive heuristic for job scheduling with sequence-independent setup times

Inputs

n: number of jobs;
g: number of production stages;
m_k: number of parallel identical machines in stage $k \in \{1, 2, ..., g\}$;
p_{jk}: processing time of job $j \in \{1, 2, ..., n\}$ in stage $k \in \{1, 2, ..., g\}$;
s_{jk}: setup time for job $j \in \{1, 2, ..., n\}$ in a machine of stage $k \in \{1, 2, ..., g\}$;
$ORP_v = \{J_1, J_2, ..., J_u, ..., J_n\}$: job ordering according to one of the proposed priority rules $v \in \{1, 2, ..., 7\}$. (Refer to Table 3)

Variables

C_{qk} : load of machine q in stage k;
R_{jqk}: time at which machine q of stage k is ready for processing job j;
b_{jk}: starting time of processing job j in stage k;
c_{jk}: finishing time of processing job j in stage k;
T_{qk}: cumulative blocking time of machine q of stage k.

Algorithm

1. (Initialization) $C_{qk} \leftarrow 0$, $T_{qk} \leftarrow 0$ for $k \in \{1, 2, ..., g\}$, $q \in \{1, 2, ..., m_k\}$
2. For $j \in ORP_v$ do:
   2.1 For $k \in \{1, 2, ..., g\}$, $q \in \{1, 2, ..., m_k\}$ do:
   $R_{jqk} \leftarrow C_{qk} + s_{jk}$
   $q^* \leftarrow \text{argmin}_q \{R_{jqk}\}$
   $R^*_{jk} \leftarrow \min_q \{R_{jqk}\}$
   2.2. (Schedule job j in stage 1):
   $b_{j1} \leftarrow R^*_{j1}$
   $c_{j1} \leftarrow b_{j1} + p_{j1}$
   $C_{q^*1} \leftarrow c_{j1}$
   2.3. For $k \in \{1, 2, ..., g\text{-}1\}$ do:
   If $c_{jk} > R^*_{jk}$ then:
   $b_{j(k+1)} \leftarrow c_{jk}$
   $c_{j(k+1)} \leftarrow b_{j(k+1)} + p_{j(k+1)}$
   $C_{q^*k} \leftarrow c_{jk}$
   $C_{q^*(k+1)} \leftarrow c_{j(k+1)}$
   else: (machine is blocked)
   $b_{j(k+1)} \leftarrow R^*_{jk}$
   $c_{j(k+1)} \leftarrow b_{j(k+1)} + p_{j(k+1)}$
   $C_{q^*k} \leftarrow b_{j(k+1)}$
   $C_{q^*(k+1)} \leftarrow c_{j(k+1)}$
   $T_{q^*k} \leftarrow T_{q^*k} + R^*_{j(k+1)} - c_{jk}$

- s_{ijk}: setup time of a machine in stage $k \in \{1, 2,\ldots, g\}$ for processing job $j \in \{1, 2,\ldots, n\}$ when preceded by job $i \in \{1, 2,\ldots, n\}$ with $i \neq j$. When $i = j$, $s_{ijk} = s_{jjk}$ is the setup time when job j is the initial job scheduled to be processed in stage k; (There is no preceding job yet).
- $S'_{jk} = \frac{1}{n}\sum_{i=1}^{n} s_{ijk}$: average setup time in stage k taking into account all preceding jobs of job j and the setup time when job j is the initial one in stage k;
- $\underline{S}_j = \sum_{k=1}^{g} s'_{jk}$: sum of average setup times in all stages for job j;
- $P\underline{S}_j = \sum_{k=1}^{g}\left(p_{jk} + S'_{jk}\right)$: sum of processing time and average setup time for job j over all stages;
- $s_{jk}^{\max} = \max_{i\in\{1,2,\ldots,n\}} s_{ijk}$: maximum setup time for job j in stage k;
- $s_{jk}^{\min} = \min_{i\in\{1,2,\ldots,n\}} s_{ijk}$: minimum setup time for job j in stage k;
- $PS_j^{\max} = \sum_{k=1}^{g}\left(p_{jk} + s_{jk}^{\max}\right)$: sum of the processing times and maximum setup times for job j over all stages;

- $PS_j^{\min} = \sum_{k=1}^{g} \left(p_{jk} + s_{jk}^{\min}\right)$: sum of the processing times and minimum setup times for job j over all stages.

The proposed rules are described in Table 4, while Fig. 4 describes the proposed constructive heuristic for job scheduling.

5 Computational experiments

5.1 Experimental design

We used as parameters for the computational experiments the number of jobs (n), the number of production stages (g), the number of parallel identical machines per stage (m_k) and the setup intervals (s). The values of these parameters were defined based on the literature review, as presented in Table 5. We considered a fixed interval for the processing times, with random values uniformly distributed between 1 and 99. The total number of instance classes is given by 10 (n) × 2 (g) × 2 (m_k) × 3 (s) = 120. For each class, we randomly generated 100 test instances aiming at reducing the sampling error, which gives a total of 12,000 test instances.

In Table 1 all problem classes were considered according to the combination of the 10 sets of n jobs (10, 20, 30,…,100), with the 2 sets of stages g (3, 7), 2 sets of parallel machines m_k (3, 7) and 3 sets of setup times s ([1, 25], [26, 75], [76, 125]). This scheme results in 120 test problem classes. In the computational experimentation, we solved 100 problems for each class, totalizing 12,000 tested problems.

On the other hand, as presented in Table 6, the 12 subclasses (A, B, C,…, K, L) are related to the 12 parameter combinations g, mk, and s. Thus, the subclass A has 100 test problems with $n = 10$, 100 test problems with $n = 20$, and so on, until 100 test problems with $n = 100$. Each subclass presents 1000 test problems, composed of 100 test problems for each value of n. Therefore, the total number of evaluated test problems is 12 × 1000 = 12,000.

Table 4 Proposed priority rules for ordering jobs with sequence-dependent setup times

#	Rule	Order jobs according to
1	LP	Non-increasing P_j
2	LS	Non-increasing S_j
3	LPS	Non-increasing PS_j
4	$\text{LPS}^{\max}$	Non-increasing $PS_j^{\max}$
5	$\text{LPS}^{\min}$	Non-increasing $PS_j^{\min}$
6	SP	Non-decreasing P_j
7	SS	Non-decreasing S_j
8	SPS	Non-decreasing PS_j
9	$\text{SPS}^{\max}$	Non-decreasing $PS_j^{\max}$
10	$\text{SPS}^{\min}$	Non-decreasing $PS_j^{\min}$
11	RAND	Random order

5.2 Statistics used in the analysis of the computational experiments

The results obtained in the computational experiments were analyzed by the success rate and the mean relative deviation for all the priority rules in Tables 3 and 4. The success rate (SR) is calculated as the number of times that a given rule results in the best solution (with or without a draw) divided by the number of test instances in a given instance class. The relative deviation (RD) is the variation percentage corresponding to the best solution founded by the methods. If the relative deviation is equal to zero for a given method, it returned the best solution for that instance. One can observe that more than one method may provide the best solution.

Thus, the better algorithm is the one providing a lower value of mean relative deviation (the simple average of the relative deviations) for a given class of problems. The relative deviation (RD_h) for the method h for a given problem is calculated as in Eq. (1):

$$\text{RD}_h = \frac{D_h - D^*}{D^*} \tag{1}$$

in which D_h is the makespan returned by the method h and D^* is the best makespan returned by the tested methods.

In this work there were not considered the average computational times of a given method for the comparative evaluation because the characteristics of the proposed algorithms should not imply in differences with statistical significance. Furthermore, such computational times are relatively smalls due to the fact that usually constructive heuristic methods have high computational efficiency.

5.3 Results and discussion

To present the results in a summarized way, we grouped the instance classes according to the parameter "number of jobs" (n) in 12 groups. Table 6 gives the labels for each group and the corresponding values for the remaining experimental parameters.

In the Sects. 5.3.1 and 5.3.2, we show the results for sequence-independent and sequence-dependent setup times, respectively.

Inputs

n: number of jobs;
g: number of production stages;
m_k: number of parallel identical machines in stage $k \in \{1, 2, ..., g\}$;
p_{jk}: processing time of job $j \in \{1, 2, ..., n\}$ in stage $k \in \{1, 2, ..., g\}$;
s_{ijk}: setup time for job $j \in \{1, 2, ..., n\}$ in a machine of stage $k \in \{1, 2, ..., g\}$ after processing job $i \in \{1, 2, ..., n\}$;
$ORP_v = \{J_1, J_2, ..., J_u, ..., J_n\}$: job ordering according to one of the proposed priority rules $v \in \{1, 2, ..., 11\}$. (Refer to Table 4)

Variables

C_{qk} : load of machine q in stage k;
R_{jqk}: time at which machine q of stage k is ready for processing job j;
b_{jk}: starting time of processing job j in stage k;
c_{jk}: finishing time of processing job j in stage k;
T_{qk}: cumulative blocking time of machine q of stage k.

Algorithm

1. (Initialization) $C_{qk} \leftarrow 0$, $T_{qk} \leftarrow 0$ for $k \in \{1, 2, ..., g\}$, $q \in \{1, 2, ..., m_k\}$
2. For $j \in ORP_v$ do:
   2.1 For $k \in \{1, 2, ..., g\}$, $q \in \{1, 2, ..., m_k\}$ do:
   $R_{jqk} \leftarrow C_{qk} + s_{ijk}$
   $q^* \leftarrow \text{argmin}_q \{R_{jqk}\}$
   $R^*_{jk} \leftarrow \min_q \{R_{jqk}\}$
   2.2. (Schedule job j in stage 1):
   $b_{j1} \leftarrow R^*_{j1}$
   $c_{j1} \leftarrow b_{j1} + p_{j1}$
   $C_{q^*1} \leftarrow c_{j1}$
   2.3. For $k \in \{1, 2, ..., g\text{-}1\}$ do:
   If $c_{j\,k} > R^*_{jk}$ then:
   $b_{j(k+1)} \leftarrow c_{j\,k}$
   $c_{j(k+1)} \leftarrow b_{j(k+1)} + p_{j(k+1)}$
   $C_{q^*k} \leftarrow c_{jk}$
   $C_{q^*(k+1)} \leftarrow c_{j(k+1)}$
   else: (machine is blocked)
   $b_{j(k+1)} \leftarrow R^*_{jk}$
   $c_{j(k+1)} \leftarrow b_{j(k+1)} + p_{j(k+1)}$
   $C_{q^*k} \leftarrow b_{j(k+1)}$
   $C_{q^*(k+1)} \leftarrow c_{j(k+1)}$
   $T_{q^*k} \leftarrow T_{q^*k} + R^*_{j(k+1)} - c_{jk}$

Fig. 4 Proposed constructive heuristic for job scheduling with sequence-dependent setup times

Table 5 Parameters used in the computational experiments

Parameter	Number of levels	Values
Number of jobs (n)	10	10, 20, 30, 40, 50, 60, 70, 80, 90, 100
Number of stages (g)	2	3, 7
Number of parallel identical machines per stage (m_k)	2	2, 5
Distribution of setup times (s)	3	U[1, 25], U[26, 75], U[76, 125]

Table 6 Tested instance classes

Class	Parameters g/m_k/s
A	3/2/U[1, 25]
B	3/2/U[26, 75]
C	3/2/U[76, 125]
D	3/5/U[1, 25]
E	3/5/U[26, 75]
F	3/5/U[76, 125]
G	7/2/U[1, 25]
H	7/2/U[26, 75]
I	7/2/U[76, 125]
J	7/5/U[1, 25]
K	7/5/U[26, 75]
L	7/5/U[76,125]

5.3.1 Sequence-independent setup times

Several priority rules alternate as the best rule for a given set of instances with regard to the SR. Therefore, the SR results are analyzed concomitantly with results obtained for RD. Table 7 and Fig. 5 present results obtained considering SR. Table 8 and Fig. 6 present results obtained considering RD. Table 9 summarizes the comparison of best priority rules for each instance class.

The obtained results clearly point to a better performance of the LP priority rule. The worst priority rules were LS, SS and RAND. Regarding the SR indicator, one can notice a superiority earlier stated for the LP rule, however, without a tendency for the increase of the RS for small and medium size test instances. The positive tendency occurs for test instances with a number of jobs greater than 80.

As expected, the random dispatch rule RAND presented in general the worst performance, what indicates that structured rules lead to better results. It is worth noting that most of the structured rules which did not have good performance are based on the traditional priority rule SPT. It is well known that in production planning environments with parallel machines, this rule does not present good results when the objective function is related to the makespan minimization, as it was the case with the SPT-based priority rules SP, SS and SPS.

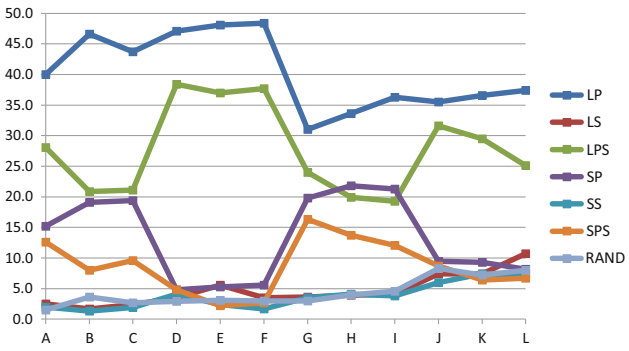

Fig. 5 Comparison of average SR for each proposed priority rule over class instances with sequence-independent setup times

On the other hand, the priority rules that performed best in the tested instances were LP and LPS, which are based on the traditional LPT. In production environments with parallel machines, the LPT rule seeks to balance the machines workload so as to minimize the makespan. The LS rule does not present good results for the variant under study.

5.3.2 Sequence-dependent setup times

Table 10 and Fig. 7 present results obtained considering SR. Table 11 and Fig. 8 present results obtained considering RD. Table 12 summarizes the comparison of best priority rules for each instance class.

In general, for problem instances with number of stages $g = 3$, the priority rule LP$\underline{S}$ has achieved the best performance, with the exception for problem instances with setup times in the interval [1, 25], in which the priority rule LPS$^{\min}$ performed better than the LP$\underline{S}$ one. In the case of the HFSP with a greater number of production stages ($g = 7$), the priority rule LP$\underline{S}$ also showed the best results, with the exception of the instance class L, for which LP$\underline{S}$ performed best only for instances with number of machines $n \geq 90$. As expected, the random rule RAND showed the worst results.

Table 7 SR for each priority rule and instance class with sequence-independent setup times (average over the number of jobs n)

Priority rule	Instance class											
	A	B	C	D	E	F	G	H	I	J	K	L
LP	40.0	46.6	43.7	47.1	48.1	48.4	31.0	33.6	36.3	35.5	36.6	37.4
LS	2.5	1.7	2.4	3.5	5.6	3.5	3.6	3.9	4.1	7.4	7.4	10.7
LPS	28.1	20.9	21.1	38.4	37.0	37.7	24.0	19.9	19.3	31.6	29.5	25.1
SP	15.2	19.1	19.4	4.8	5.3	5.6	19.8	21.8	21.3	9.5	9.3	8.2
SS	2.0	1.3	1.9	4.1	2.4	1.7	3.5	4.1	3.8	6.0	7.4	7.2
SPS	12.6	8.0	9.6	4.8	2.2	2.7	16.3	13.7	12.1	8.7	6.4	6.7
RAND	1.5	3.6	2.7	2.9	3.1	3.0	3.0	4.0	4.6	8.3	7.2	8.0

Table 8 RD for each priority rule and instance class with sequence-independent setup times (average over the number of jobs *n*)

Priority rule	Instance class											
	A	B	C	D	E	F	G	H	I	J	K	L
LP	0.8	0.5	0.4	0.6	0.6	0.5	1.3	1.0	0.8	0.8	0.8	0.6
LS	7.2	4.6	3.5	7.3	5.1	4.0	5.2	3.8	2.9	4.8	3.7	2.9
LPS	1.3	1.4	1.0	0.8	0.7	0.4	1.5	1.5	1.3	1.0	1.1	1.2
SP	2.7	1.9	1.4	6.1	5.0	3.9	2.4	1.9	1.4	4.7	4.5	3.3
SS	7.4	5.0	3.8	7.6	6.1	4.9	5.2	3.7	3.1	4.8	4.3	3.4
SPS	3.0	2.7	2.1	6.3	5.3	4.0	2.5	2.5	2.0	4.6	4.4	3.6
RAND	7.4	4.8	3.6	7.7	5.7	4.5	5.1	3.6	2.8	4.7	4.1	3.2

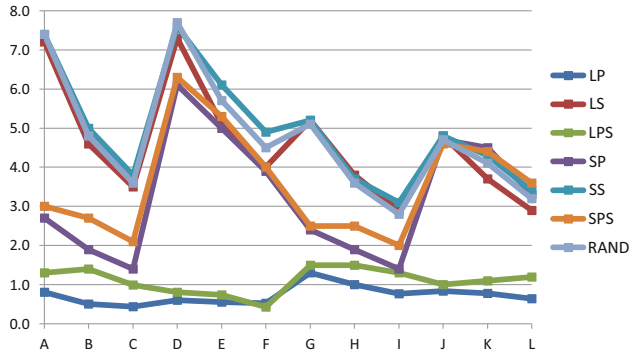

Fig. 6 Comparison of average percent relative deviation for each proposed priority rule by instance category with sequence-independent setup times

Table 9 Best priority rules for each instance class with sequence-independent setup times

Instance class	Best priority rules
A	LP, LPS, SP and SPS
B	LP, LPS, SP and SPS
C	LP, LPS, SP and SPS
D	LP, LPS, SP/SPS and SS
E	LP, LPS, SP and SS
F	LP, LPS, SP and SS
G	LP, LPS, SP and SPS
H	LP, SP, LPS and SPS
I	LP, SP, LPS and SPS
J	LP, LPS, SP and SPS
K	LP, LPS, SP and SS
L	LP, LPS, SP and SS

It is worth noting that most of the structured rules which did not have good performance are based on the traditional priority rule SPT. It is well known that in production-planning environments with parallel machines, this rule does not present good results when the objective function is related to the makespan minimization, as it was the case with the SPT-based priority rules SP, SS, SPS, SPS^{max} and SPS^{min}.

On the other hand, the priority rules that performed best in the tested instances were LP, LPS, LPS^{max} and LPS^{min}, which are based on the traditional LPT. In production environments with parallel machines, the LPT rule seeks to balance the machines workload so as to minimize the makespan.

Table 10 SR for each priority rule and instance class with sequence-dependent setup times (average over the number of jobs *n*)

Priority rule	Instance class											
	A	B	C	D	E	F	G	H	I	J	K	L
LP	33.9	25.6	26.7	44.4	35.5	36.2	27.3	26.5	28.1	31.1	33.4	33.7
LS	1.8	2.6	1.5	3.3	3.6	3.9	4.2	5.6	6.2	8.8	6.6	8.2
LPS	35.3	28.8	28.4	45.5	40.7	42.6	27.0	26.1	26.7	32.9	34.8	32.8
LPS^{max}	33.5	25.8	26.7	44.9	37.0	35.5	27.3	26.8	28.0	31.3	33.8	33.6
LPS^{min}	33.9	25.7	26.2	45.1	34.6	36.6	27.3	26.1	28.1	31.2	33.5	34.1
SP	13.1	11.3	12.1	3.8	3.2	3.1	19.6	17.3	15.7	10.4	7.3	6.1
SS	2.0	2.2	2.9	3.5	3.2	3.3	4.3	5.1	4.5	7.7	7.0	9.3
SPS	13.3	12.3	12.7	3.9	3.9	3.1	17.3	16.2	14.5	8.4	7.7	7.4
SPS^{max}	12.9	11.5	11.1	3.9	3.6	3.1	19.6	17.2	16.0	10.4	7.3	6.1
SPS^{min}	12.6	11.5	12.1	3.7	3.4	3.0	19.6	17.3	16.0	10.4	7.3	6.1
RAND	2.0	2.1	2.5	3.4	2.9	3.2	3.9	5.0	5.0	8.9	7.3	7.9

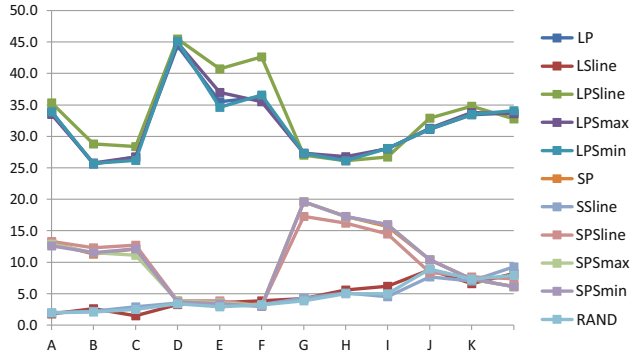

Fig. 7 Comparison of average SR for each proposed priority rule over class instances with sequence-dependent setup times

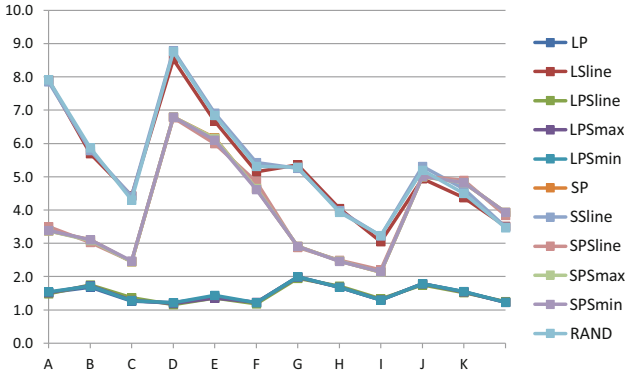

Fig. 8 Comparison of average percent relative deviation for each proposed priority rule by instance category with sequence-dependent setup times

6 Concluding remarks

In this paper, we have investigated a new variant for the hybrid flow shop scheduling problem (HFSP) considering both machine blocking and both sequence-independent and sequence-dependent setup times. The HFSP objective is minimizing the total time to complete the schedule (makespan). We developed constructive heuristic algorithms based on priority rules. We also proposed seven priority rules in the case of sequence-independent and eleven priority rules in the case of sequence-dependent setup times, respectively, based on the SPT and LPT rules.

Computational experiments were carried out to evaluate the performance of the alternative proposed rules. We used the rate of success and the relative deviation statistics as performance measures. Computational times were not used for comparison purposes since their differences among priority rules were negligible. In most tested problem instances, the priority rules based on the SPT one generated the worst solutions. On the other hand, the priority rules based on the LPT showed the best performance. In particular, the LP rule outperformed all other tested priority rules in most cases with sequence-independent setup times, while LPS rule

Table 12 Best priority rules for each instance class sequence-dependent setup times

Instance class	Best priority rules
A	LPS^{min}, LPS, LP and LPS^{max}
B	LPS, LP, LPS^{max} and LPS^{min}
C	LPS, LPS^{min}, LPS^{max} and LP
D	LPS^{min}, LPS^{max}, LP and LPS
E	LPS, LPS^{max}, LP and LPS^{min}
F	LPS, LPS^{min}, LP and LPS^{max}
G	LPS, LPS^{max}, LPS^{min} and LP
H	LPS, LPS^{max}, LP and LPS^{min}
I	LPS, LPS^{max}, LP and LPS^{min}
J	LPS, LPS^{max}, LP and LPS^{min}
K	LPS, LPS^{max}, LP and LPS^{min}
L	LPS^{min}, LP, LPS^{max} and LPS

Table 11 RD for each priority rule and instance class with sequence-dependent setup times (average over the number of jobs *n*)

Priority rule	Instance class											
	A	B	C	D	E	F	G	H	I	J	K	L
LP	1.5	1.7	1.3	1.2	1.4	1.2	2.0	1.7	1.3	1.8	1.5	1.2
LS	7.9	5.7	4.4	8.5	6.7	5.1	5.4	4.0	3.0	4.9	4.4	3.5
LPS	1.5	1.8	1.4	1.2	1.4	1.2	2.0	1.7	1.3	1.7	1.5	1.3
LPS^{max}	1.5	1.7	1.3	1.2	1.4	1.2	2.0	1.7	1.3	1.8	1.5	1.2
LPS^{min}	1.5	1.7	1.3	1.2	1.4	1.2	2.0	1.7	1.3	1.8	1.6	1.2
SP	3.4	3.1	2.4	6.8	6.2	4.6	2.9	2.5	2.1	5.0	4.8	3.9
SS	7.9	5.8	4.4	8.8	6.9	5.4	5.3	4.0	3.2	5.3	4.6	3.5
SPS	3.5	3.0	2.5	6.8	6.0	4.8	2.9	2.5	2.2	5.0	4.9	3.8
SPS^{max}	3.4	3.1	2.4	6.8	6.2	4.6	2.9	2.5	2.1	5.0	4.8	3.9
SPS^{min}	3.4	3.1	2.5	6.8	6.1	4.6	2.9	2.5	2.1	5.0	4.8	3.9
RAND	7.9	5.9	4.3	8.8	6.8	5.3	5.3	3.9	3.2	5.2	4.5	3.5

outperformed other tested priority rules in most cases with sequence-dependent setup times.

Our purpose in this paper was to compare alternative priority rules, not to compare the priority rule scheduling method with scheduling based on mathematical programming. This is why we do not provide quality measures based on lower bounds or optimal solutions as they require developing and solving mathematical programming models. Priority rules are often used in practice and comparing alternative rules is relevant from both a practical as well as a theoretical standpoint.

As extensions of this work, we recommend the use of metaheuristics to improve the solutions generated by the priority rules. In addition, the development of integer linear programming models for the variants under study is suggested, since the models can be used to determine optimal solutions for small-size problem instances. Future studies could also investigate the behavior of the proposed priority rules considering other objective functions, such as total tardiness minimization. There is no guarantee that LPT-based rules will also have good performance when applied to different objective functions. Finally, it is worth studying multi-objective variants of the HSFP due to the fact that in many real industrial settings, the scheduling problems have a multi-objective nature.

Acknowledgements The support of the National Council for Scientific and Technological Development (CNPq) is acknowledged and appreciated.

References

1. Choong F, Phon-Amnuaisuk S, Alias MY (2011) Metaheuristic methods in hybrid flow shop scheduling problem. Expert Syst Appl 38:10787–10793
2. Ebrahimi M, Fatemi Ghomi SMT, Karimi B (2014) Hybrid flow shop scheduling with sequence dependent family setup time and uncertain due dates. Appl Math Model 38:2490–2504
3. Elmi A, Topaloglu S (2013) A scheduling problem in blocking hybrid flow shop robotic cells with multiple robots. Comput Oper Res 40:2543–2555
4. Fattahi P, Hosseini SMH, Jolai F, Tavakkoli-Moghaddam R (2014) A branch and bound algorithm for hybrid flow shop scheduling problem with setup time and assembly operations. Appl Math Model 38:119–134
5. Garey MR, Johnson DS, Sethi R (1976) The complexity of flowshop and jobshop scheduling. Math Oper Res 1(2):117–129
6. Hidri L, Haouari M (2011) Bounding strategies for the hybrid flow shop scheduling problem. Appl Math Comput 217(21):8248–8263
7. Kis T, Pesch E (2005) A review of exact solution methods for the non-preemptive multiprocessor flowshop problem. Eur J Oper Res 164:592–608
8. Linn R, Zhang W (1999) Hybrid flow shop scheduling: a survey. Comput Ind Eng 37(1):57–61
9. Luo H, Du B, Huang GQ, Chen H, Li X (2013) Hybrid flow shop scheduling considering machine electricity consumption cost. Int J Prod Econ 146:423–439
10. Moccellin JV, Nagano MS (2011) Heuristic for flow shop sequencing with separated and sequence-independent setup times. J Braz Soc Mech Sci Eng 33:74–78
11. Mousavi SM, Zandieh M, Amiri M (2011) An efficient bi-objective heuristic for scheduling of hybrid flow shops. Int J Adv Manuf Technol 54:287–307
12. Nagano MS, Araujo DC (2014) New heuristics for the no-wait flowshop with sequence-dependent setup times problem. J Braz Soc Mech Sci Eng 36:139–151
13. Pinedo ML (2008) Scheduling: theory, algorithms and systems, 3rd edn. Springer, New York
14. Quadt D, Kuhn H (2007) A taxonomy of flexible flow line scheduling procedures. Eur J Oper Res 178(3):686–698
15. Ribas I, Leisten R, Framinan JM (2010) Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective. Comput Oper Res 37:1439–1454
16. Ruiz R, Vázquez-Rodríguez JA (2010) The hybrid flow shop scheduling problem. Eur J Oper Res 205:1–18
17. Vignier A, Billaut JC, Proust C (1999) Les problèmes d'ordonnancement de type flow-shop hybride: état de l'art. RAIRO 33(2):117–183
18. Wang H (2005) Flexible flow shop scheduling: optimum, heuristics and artificial intelligence solutions. Expert Syst 22:78–85
19. Zhang Y, Liang X, Li W, Zhang Y (2013) Hybrid flow shop problem with blocking and multi- product families in a maritime terminal. In: 2013 IEEE 10th International Conference on Networking, Sensing and Control (ICNSC)