Contents lists available at GrowingScience

International Journal of Industrial Engineering Computations

homepage: www.GrowingScience.com/ijiec

A computational evaluation of constructive heuristics for the parallel blocking flow shop problem with sequence-dependent setup times

Imma Ribas[a*] and Ramon Companys[b]

[a]*Departament d'Organització d'Empreses, DOE – ETSEIB - Universitat Politècnica de Catalunya. BarcelonaTech, Avda. Diagonal, 647, 7th Floor, 08028 Barcelona, Spain*
[b]*Universitat Politècnica de Catalunya. BarcelonaTech, Spain*

C H R O N I C L E

Article history:
Received September 13 2020
Received in Revised Format November 28 2020
Accepted January 22 2021
Available online January, 22 2021

Keywords:
Blocking
Parallel flow shop
Distributed flow shop
Dependent setup times
Makespan

A B S T R A C T

This paper deals with the problem of scheduling jobs in a parallel flow shop environment without buffers between machines and with sequence-dependent setup times in order to minimize the maximum completion time of jobs. The blocking constraint normally leads to an increase in the maximum completion time of jobs due to the blockage of machines, which can increase even more so when setup times are considerable. Hence, the heuristic to solve this problem must take into account these specificities in order to minimize the timeout of machines. Because the procedures designed to solve the parallel flow shop scheduling problem must deal not only with the sequencing of jobs but also with their allocation to the flow shops, 36 heuristics have been tested in this paper, of which 35 combine sequencing rules with allocation methods while the last one takes a different approach that is more related to the nature of this problem. The computational evaluation of the implemented heuristics showed good performance of the heuristic designed especially for the problem (RCP0) when the setup times are considerable. Furthermore, the evaluation has also allowed us to propose a combined heuristic that leads to good solutions in a short CPU time.

1. Introduction

This paper addresses the problem of scheduling n jobs in F identical parallel flow shops without buffers and with sequence-dependent setup times to minimize the maximum completion time of jobs (makespan). The parallel flow shop is a productive configuration found in different industries, such as the glass industry (He et al., 1996) or the production process of electrical appliances and industrial supplies (Jiang and Wan, 2011). The parallel permutation flow shop scheduling problem (PPFSP) with only two machines was studied by He et al. (1996) to deal with the scheduling problem in a glass manufacturing facility, for which they proposed using mixed integer programming and an efficient heuristic. The same problem was later addressed by Cao and Chen (2003) who developed a mathematical model and a tabu search algorithm; by Al-Salem (2004) who proposed a new polynomial-time heuristic; and by Zhang and Van De Velde (2012), who presented approximation algorithms with worst-case performance guarantees for scheduling jobs in two and three lines. The general case (i.e., the m-machine PPFSP) was studied by Jiang and Wan (2011), who proposed a quantum algorithm. The Parallel Permutation Flow Shop scheduling problem is similar to the Distributed Permutation Flow Shop scheduling problem (DPFSP), which deals with scheduling jobs in several identical factories in which each has one flow shop (line) in the production system. Although both problems are conceptually different, the solution methods proposed for one can also be used to solve the other. The DPFSP for minimizing total makespan has been studied by several authors since Naderi and Ruiz (2010) first presented the problem, for which various metaheuristics have been proposed to solve it (Liu and Gao, 2010; Gao and Chen, 2011; Gao et al., 2012; Gao et al., 2013; Lin et al., 2013; Wang et al., 2013; Xu et al., 2013; Fernandez-Viagas and Framinan, 2014; Naderi and Ruiz, 2014).

* Corresponding author
E-mail: imma.ribas@upc.edu (I. Ribas)

2021 Growing Science Ltd.
doi: 10.5267/j.ijiec.2021.1.004

In this paper, we study two extensions of the classical PPFSP in order to bring the problem closer to the reality of production environments. On the one hand, we consider that no intermediate storage (buffers) exists between machines, which can provoke the blockage of machines if the job being processed finishes before the next machine is free. The blocking constraint allows modeling several production systems in which machines can be blocked due not only to the lack of buffers but also to other manufacturing or technological constraints, for instance, in systems where the parts move using robots and the treated parts must wait in the machine until the robot can pick them up and move them to the next machine (Sethi et al. (1992). Additional examples are found in the iron and steel industry (Gong et al. 2010), the processing of industrial waste, and the production of metallic parts (Martinez et al, 2006). Recently, Miyata and Nagano (2019) published a comprehensive review of the blocking flow shop problem, in which they analysed the proposed heuristic methods in order to solve the problem. The parallel blocking flow shop problem (PBFSP) has been studied by Ribas et al. (2017) who analyzed the performance of an iterated local search algorithm and an iterated greedy algorithm in order to minimize the makespan and Ribas et al.(2019) who proposed an iterated greedy algorithm to minimize total tardiness. On the other hand, we consider setup times, which are the periods of time when the machines are prepared, adjusted, or cleaned between two consecutives jobs. Setup times can be sequence-independent or sequence-dependent. In the former case, the setup time depends on the job to be processed; in the latter one, the amount of time required to configure machines also depends on the job that was previously processed in that machine. Setup activities are very important in some production systems and imply a loss of available capacity. Trovinger and Bohn (2005) reported direct savings of $1.7 million per year due to reduced setup times in a printed circuit board assembly plant. Even though considering setup times in scheduling decisions can help increase productivity and reduce both waste and non-value-added activity while improving the use of resources and meeting deadlines, it is addressed by only about 10% of the existing literature on scheduling (Allahverdi, 2015). To solve the blocking flow shop scheduling problem with setups, two aspects should be considered: when the machines can be set up and when the jobs processed by one machine can move on to the next machine. Regarding the moment in which setup can begin, three possibilities exist. For the sake of clarity, let h be the job being processed by the considered machine and i the next job to be processed in that machine:

a) When job h has finished.
b) When job h has left the machine.
c) When job i has arrived at the machine.

Concerning the moment in which a job can move from its current machine to the next one, two possibilities exist:

i. When job h has left the machine.
ii. When the setup for processing job i has finished.

In this research, we consider that setup can be performed when job h has left the machine (option b) and job i can move on to the next machine once job h has left it, even though the setup is not finished. To the best of our knowledge, the PBFSP with sequence-dependent setup times, has not been studied before, for which we have designed different constructive heuristics that combine several sequencing rules and allocation strategies in order to provide simple methods for solving the problem. Therefore, this paper tests the performance of several constructive procedures in order to propose a simple but efficient method to minimize makespan. The remainder of the paper is organized as follows. Section 2 is devoted to defining the problem. Section 3 describes the constructive methods proposed. The computational results and comparisons are shown in Section 4. Finally, Section 5 concludes and proposes future lines of research.

2. Problem definition

The parallel blocking flow shop scheduling problem (PBFSP) consists of scheduling a set of n jobs in F identical flow shops (lines). Each line has m machines and each job must be allocated to one of the f lines. The jobs allocated to a line has to be processed, in the same order, in all machines. Each job i, $i \in \{1, 2,..., n\}$ requires a fixed processing time $p_{j,i}$ and a setup time on every machine j, $j \in \{1, 2,..., m\}$. Jobs and machines are available from time zero onwards. The objective is to allocate the jobs to the lines and schedule them in order to minimize the maximum makespan (C_{max}) among lines. We denote as n_f the set of jobs assigned to line f; σ_f is the sequence of the n_f jobs assigned to line f; and $fmax$ is the line with the maximum C_{max}. Hence, a solution Π is composed of the sequence of jobs in each line ($\Pi=(\sigma_1, \sigma_2, \ldots, \sigma_F)$).

Next, we present additional notation to define the makespan calculation. Let [k] be the index of the job in the kth position in the sequence. $S_{j,i,k}$ are the sequence-dependent setup times (SDST) needed in machine j after having processed job i and before processing job [k]. This setup can be done once job i has left the machine. Let $S_{j,0,k}$ be the setup time required to process the first job in the sequence; $\sigma_{k,f}$ is the job [k] in line f; and $c_{j,k,f}$ is the departure time of job [k] from machine j of line f. Notice that if machine j+1 is available, job i can leave machine j when it is completed. In this situation, $c_{j,k,f}$ is the completion time of job [k] in machine j of line f. The outline of the makespan calculation is shown in Fig. 1.

$f_{max} = 0$
for $f = 1$ **to** F
  $i = \sigma_{1,f}$
  $c_{1,1,f} = s_{j,0,k} + p_{1,i}$
  for $j = 2$ **to** m
    $c_{j,1,f} = \mathbf{max}\{s_{j,0,i} ; c_{j-1,1,f}\} + p_{j,i}$
  next j
  for $k = 2$ to n_f
    $i = \sigma_{k,f}$: $h = \sigma_{k-1,f}$
    for $j = 1$ to m
      $c_{j,k,f} = \mathbf{max}\{ \mathbf{max}\{c_{j,k-1,f} + s_{j,h,i} ; c_{j-1,k,f}\} + p_{j,i} ; c_{j+1,k-1,f}\}$
    next j
  next k
  if $f_{max} < c_{m,nf,f}$ **then** $f_{max} = c_{m,nf,f}$
  next f

Being:
  $c_{0,k,f} = c_{m+1,k,f} = 0 \quad \forall\, k, f$

Fig. 1. Outline of makespan calculation

3. Constructive heuristics

The scheduling of jobs in a parallel flow shop has to determine the assignment of jobs to lines and the sequence of job allocated to each line. Hence, constructive heuristics can be built by combining ordering rules with allocation strategies. In this paper, we have tested 36 heuristics, of which 35 are the result of combining seven sequencing rules with five allocation methods plus an improvement phase. The last one, named RCP0, is totally different and is explained at the end of this section. The sequencing rules used in the first 35 heuristics are: Largest Processing Time (LPT); Shortest Processing Time (SPT); Trapeziums (TRA) (Companys, 1966); the Palmer heuristic (PAL) (Palmer, 1965); two variants of PF (McCormick et al., 1989) and HPF (Ribas and Companys, 2015).

- **LPT** sequences the jobs in non-increasing order of their total processing time (sum of the processing time in each machine).
- **SPT** sequences the jobs in non-decreasing order of their total processing time (sum of the processing time in each machine).
- **TRA** calculates two indexes for each job ($S1_i$ and $S2_i$) according to (1) and (2), respectively. Next, Johnson's algorithm (Johnson, 1954) is used to sequence the jobs by considering $S1_i$ and $S2_i$ as the processing time of job i in the first and second machine, respectively.

$$S1_i = \sum_{j=1}^{m} (m - j + 1) \cdot p_{j,i} \tag{1}$$

$$S2_i = \sum_{j=1}^{m} (j - 1) \cdot p_{j,i} \tag{2}$$

- **PAL** sequences the jobs in increasing order of $S3_i = S1_i - S2_i$.

PF sequences the jobs in an order that minimizes the idle time of machines. Next, two variants are presented (PF1 and PF2) which differ in their calculations of timeout duration.

- **PF1** considers, as do Takano and Nagano (2019), timeout to be the time when the machine is not working because it is not processing any jobs or is not being prepared for the next job. This timeout can be due to idle time, blocking time, or the sum of both. Here, the total timeout duration of machines is calculated as in (3), where i denotes the job, k the position, σ the sequence of jobs already sequenced, and $\sigma * i$ the partial sequence plus job i. Notice that timeout in this case is the difference between the departure time between job i and the previous job in the sequence, minus its processing time and the setup time required to process job i. This is because setup can be performed once the previous job has left the machine:

$$Tm1_k(i) = \sum_{j=1}^{m} (c_{j,k+1}(\sigma * i) - c_{j,k}(\sigma) - p_{j,i} - s_{j,i,k}) \tag{3}$$

The job selected is the one that leads to the minimum timeout.

- **PF2** considers a machine to be idle when it is not processing any job, but not when it is being setting up. Therefore, the total timeout of machines is now calculated as in (4):

$$Tm2_k(i) = \sum_{j=1}^{m} (c_{j,k+1}(\sigma * i) - c_{j,k}(\sigma) - p_{j,i}) \quad (4)$$

The job selected is the one that leads to the minimum timeout.

- **HPF** builds the sequence in order to minimize both the timeout of machines and the total flowtime, which is carried out with the index $ind1(i, k)$ that is calculated according to (5).

$$ind1(i,k) = \mu \cdot \sum_{j=1}^{m} (c_{j,k}(\sigma * i) - c_{j,k-1}(\sigma) - p_{j,i}) + (1-\mu) \cdot (c_{m,k}(\sigma * i) - c_{m,k-1}(\sigma)) \quad (5)$$

The five allocation methods used in the first 35 heuristics can be separated into two groups. In the first group (methods 1, 2 and 3), jobs are allocated to the lines. Then, the sequence of jobs assigned to each line is improved upon by an adaptation of the NEH-insertion procedure (Nawaz et al., 1983) . Each method is described as follows:

(1) Assign job *j* to the factory with the lowest current C_{max}, not including job *j*, at the end of its partial sequence.
(2) Assign job *j* to the factory that can complete it at the earliest time, at the end of its partial sequence.
(3) Create a sequence in each line by assigning a similar load ($\Sigma P_i/F$). This process starts in one of the lines (all of them are identical) by assigning jobs until it reaches the mean load. The process is repeated for each line.

In the second group are methods 4 and 5, for which the allocation and improvement phases are mixed. This means that a job assigned to a line is allocated to the best position, i.e., the one that minimizes the partial makespan. These methods are described as follows:

(4) Assign job *j* to the line with the lowest current C_{max}, not including job *j*. Next, sequence the job at the position that minimizes its C_{max}.
(5) Assign job *j* to the line that would finish it at the earliest time if placed at the end of the sequence. Next, place the job at the position that minimizes its C_{max}.

The heuristics that result from combining the seven sequencing rules with the five allocation methods are labeled with the name of the sequencing rule plus the number of the allocation method:

Method 1: LPT1, SPT1, TRA1, PAL1, PF11, PF21, HPF1
Method 2: LPT2, SPT2, TRA2, PAL2, PF12, PF22, HPF2
Method 3: LPT3, SPT3, TRA3, PAL3, PF13, PF23, HPF3
Method 4: LPT4, SPT4, TRA4, PAL4, PF14, PF24, HPF4
Method 5: LPT5, SPT5, TRA5, PAL5, PF15, PF25, HPF5

The last proposed heuristic method uses a totally different approach, given that the job and line are selected at the same time.

- **RCP0:** At each iteration, select the line that has the last machine available sooner. Next, choose the job that leads to the minimum timeout, which is calculated by equation (4).
  This process is repeated until all jobs have been assigned.
  Finally, the sequence of jobs in each line is improved by an adaptation of the NEH-insertion procedure (Nawaz et al., 1983).

4. Computational evaluation

This section shows the computational evaluation of the heuristics, which was conducted on a set of instances that was generated *ad-hoc*. The instances were combinations of n={25, 50, 75, 100, 150, 200} and m={5, 10, 15, 20}. There were 24 groups of instances and 10 instances per group, for a total of 240 instances that are solved for F={2, 3, 4, 5} and three levels of setup times. Hence, the test is conducted on over 2880 instances. The processing time was generated according to a uniform distribution in the range [1,99], as is done in the scheduling literature. Finally, we generated the three levels of setup according to a uniform distribution. For the low level, the range interval is [1, 20]; for the medium level, it is [1, 50]; and for the high level, it is [1,120]. The evaluation of the heuristics is done by comparing the quality of their solutions and the CPU time required to reach these solutions. The quality of solutions is measured by the relative percentage deviation (RPD) that is calculated in (6):

$$RPD = \frac{Sol_{i,j} - Best_i}{Best_i} \cdot 100 \quad (6)$$

With $Sol_{i,j}$ being the makespan obtained by heuristic j in instance i, and $Best_i$ is the minimum makespan obtained by any of the heuristics tested over the same instance. The computational effort is measured by the average CPU time (ACPU), in seconds, required by the heuristic to solve the instances. The heuristics were codified in QB64 and tested on the same computer: an Intel i3 with 2.3 GHz and 8 GB RAM memory. Both instances and solutions are available from the authors on request. The first analysis allows comparing the heuristics in terms of their relationship between solution quality and computational effort. Fig. 1 shows the ARPD versus ACPU values for each heuristic. It can be observed that heuristics using allocation methods 4 and 5 require more computational effort than the others do. Notice that these heuristics are grouped at around 0.45s to the right side of the figure, far from the other heuristics, which are grouped at around 0.1s. Regarding the ARPD, we find the best performing procedures at the bottom of the figure. It is worth noting that, although TRA5 has the lowest overall ARPD, the time required to find the solution is about four times that of RCP0, whose overall ARPD is not too far from the best one. To facilitate this analysis, Table 1 details the ARPD and ACPU per heuristic, as well as the breakdown of the overall ARPD by each level of setup times. This table has been ordered according to the overall ARPD. The position in the ranking is shown to the left of each heuristic's name.

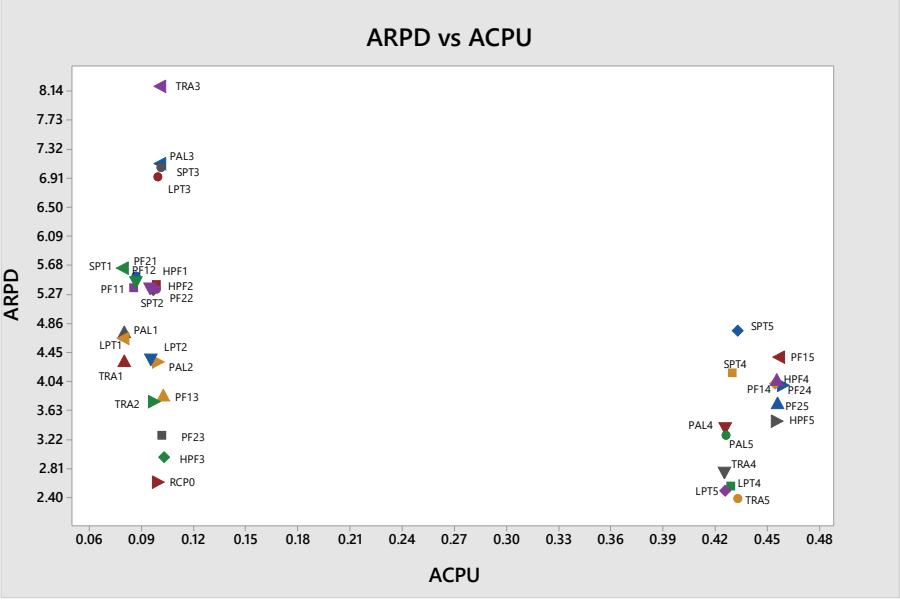

Fig.1. ARPD versus ACPU per heuristic

Regarding the overall ARPD values, the top ten heuristics (see Table 1) notably use allocation methods 3, 4 and 5; but, interestingly, the heuristic RCP0 also appears in this group. Recall that this is the heuristic using a different approach that is more related to the nature of the problem, meaning that the line and job are selected simultaneously at each iteration. It is worth mentioning that the top five rankings include—apart from heuristic RCP0—the heuristics formed by the TRA and LPT sequencing rules. This means that neither the allocation method nor the ordering rule is in itself relevant, only the combination of both.

Table 1
ARPD and ACPU of heuristics ordered by ARPD

Rnk	Heur	L	M	H	ARPD	ACPU	Rnk	Heur	L	M	H	ARPD	ACPU
1	TRA5	1.84	1.85	3.47	2.39	0.43	19	PAL2	3.80	3.66	5.48	4.31	0.10
2	LPT5	2.04	1.92	3.53	2.50	0.43	20	LPT2	3.94	3.67	5.50	4.37	0.10
3	LPT4	2.01	1.94	3.74	2.56	0.43	21	PF15	4.14	3.94	5.06	4.38	0.46
4	RCP0	3.46	2.92	1.48	2.62	0.10	22	LPT1	4.11	3.93	5.88	4.64	0.08
5	TRA4	2.19	2.21	3.90	2.77	0.43	23	PAL1	4.10	4.06	5.96	4.71	0.08
6	HPF3	2.57	2.74	3.59	2.97	0.10	24	SPT5	4.42	4.26	5.58	4.76	0.43
7	PF23	2.93	2.90	3.99	3.27	0.10	25	PF12	4.86	4.91	6.22	5.33	0.10
8	PAL5	2.98	2.81	4.05	3.28	0.43	26	PF22	4.85	4.83	6.33	5.34	0.10
9	PAL4	3.01	2.91	4.27	3.40	0.43	27	PF11	4.82	4.84	6.44	5.37	0.09
10	HPF5	2.20	3.06	5.17	3.48	0.45	28	SPT2	4.81	4.84	6.45	5.37	0.10
11	PF25	2.24	3.78	5.09	3.70	0.46	29	HPF 2	4.91	4.96	6.36	5.41	0.10
12	TRA2	2.89	3.16	5.21	3.75	0.10	30	PF21	4.96	4.88	6.55	5.46	0.09
13	PF13	2.84	3.23	5.38	3.82	0.10	31	HPF 1	4.84	5.04	6.70	5.53	0.09
14	PF14	3.57	3.55	4.83	3.98	0.46	32	SPT1	5.02	5.18	6.72	5.64	0.08
15	PF24	3.62	3.47	4.93	4.01	0.45	33	LPT3	4.96	5.67	10.14	6.92	0.10
16	HPF 4	3.57	3.65	4.90	4.04	0.46	34	SPT3	5.06	5.94	10.17	7.06	0.10
17	SPT4	3.74	3.75	4.98	4.16	0.43	35	PAL3	7.81	6.66	6.86	7.11	0.09
18	TRA1	3.51	3.69	5.70	4.30	0.08	36	TRA3	8.22	7.46	8.94	8.20	0.10

Next, we conduct a deeper analysis on the performance of these top ten heuristics by examining their behaviors in terms of number of lines and level of setup time (Fig. 2) and number of jobs per each level of setup time (Fig. 3). Fig. 2 shows that the behaviors of most heuristics are similar in terms of the number of lines and the setup level. Their ARPD is quite similar when setup time is in the low or moderate levels, and their performance is worse (higher ARPD) when the setup level is high. This is due to the procedure RCP0, which has an excellent performance when the setup time is high (especially for two and three lines) and is thus able to find most of the best solutions for these instances. On the other hand, the behavior of heuristic HPF5 (at the bottom of our top ten ranking) is also different from the majority of the other methods, since it becomes worse when setup time increases. Regarding the number of lines when considering all heuristics that rank less than RCP0, HPF3 is the best-performing heuristic when there are two lines; but its performance worsens when the number of lines increases, a situation in which TRP5 is better, followed by LPT5. However, allocation methods 4 and 5 require four times more CPU time than RCP0 and HPF3.

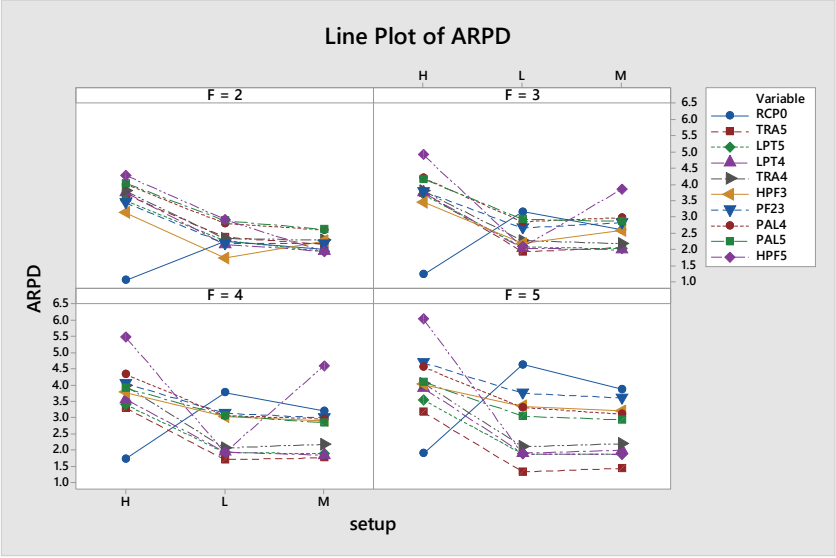

Fig. 2. ARPD per heuristic and level of setup time for each number of lines.

Looking at the effect of the number of lines (Fig. 3), we can see that the performance of RCP0 increases when the number of jobs increases. The first panel of Fig. 3 shows that ARPD is around zero when the number of jobs is 200 and the setup level is high, which means that ARPD obtains the best solutions for these instances. This affects the ARPD of the other heuristics for this level of setup times, where the performance is worse when the number of jobs increases. Conversely, for low and moderate levels of setup times: the higher the number of jobs, the better the ARPD value.

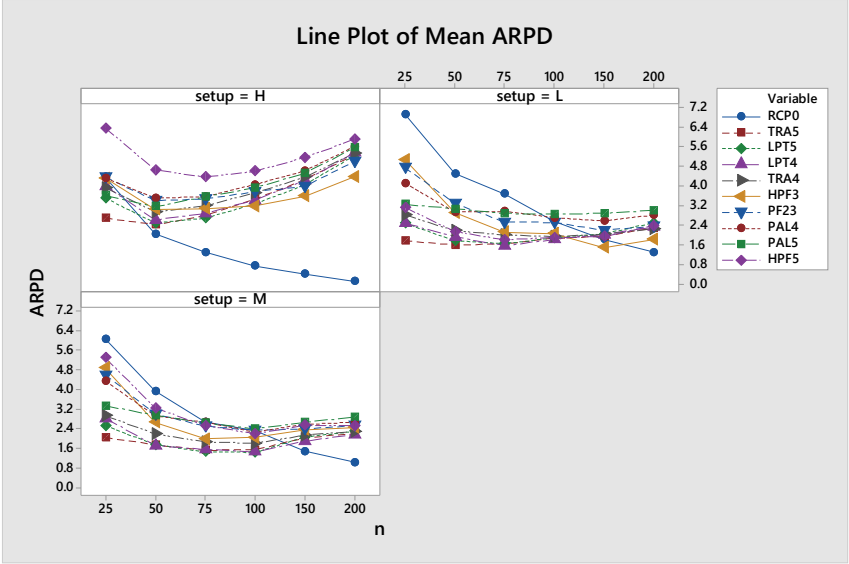

Fig. 3. ARPD per heuristic and number of jobs for each level of setup time.

Thus, RCP0 is recommended when the setup times can be about the same magnitude as the processing time of jobs, since it provides a good solution in little CPU time. However, TRA5 and LPT5 are the ones with similar behavior in all situations and additionally provide good solutions, but both require greater computational effort. Finally, even though HPF3 is in the fifth position of the ARPD ranking, it also provides a competitive solution in less time than TRA5 and LPT5. Therefore, these four procedures are recommended not only for solving the problem but also as the initial solution procedure of metaheuristic methods, since this difference between solution quality and required CPU time makes it impossible to know beforehand which one can help metaheuristics to find better solution. Regarding which of these procedures would be more suitable as a scheduling method in a real environment, we recommend creating a process that runs both RCP0 and HPF3 and keeps the best of both solutions. This would lead to reaching very competitive results (as seen in Table 2) in half the time needed by TRA5 and LPT5 to reach a solution.

Table 2
ARPD per heuristic and level of setup time

Heuristics	L	M	H	ARPD	ACPU
RCP0/HPF3	1.67	1.54	0.90	1.37	0.2
TRA5	1.84	1.85	3.47	2.39	0.43
LPT5	2.04	1.92	3.53	2.50	0.43
HPF3	2.57	2.74	3.59	2.97	0.1
RCP0	3.46	2.92	1.48	2.62	0.1

5. Conclusions

This paper has approached the parallel flow shop scheduling problem with blocking and sequence-dependent setup times. To the best of our knowledge, this is the first attempt to study and propose heuristics for solving this problem. The procedures for addressing the scheduling problem in this environment must consider the allocation of jobs to lines and the scheduling of jobs in each line. The way in which these two related decisions are taken can affect the performance of heuristics, since the timeout of machines can be affected by both blocking and setup times. Hence, this paper has tested 36 heuristics, of which 35 combine seven ordering rules with five allocation methods. The heuristics with allocation methods 1, 2, 4, and 5 first order the jobs according to a sequencing rule, then the jobs are assigned to the lines according to a given criterion. Allocation method 3 is slightly different, since a line is first selected and the jobs are then sequenced in this line until reaching the average workload, which is the moment when the process starts again for another line. Conversely, the 36th heuristic, named RCP0, selects a line at each iteration and assigns the job that leads to the minimum timeout of machines. Our computational evaluation indicates that the heuristic with the best average performance is TRA5, but it requires four times more CPU time than the very competitive performances of RCP0 and HPF3. This is especially true for RCP0 when the setup times are high, a situation in which it performs the best. A deeper analysis on the combined effect of setup times and numbers of lines and of jobs showed that the behaviors of RCP0 and HPF3 are complementary, i.e., when one is best the other is not as competitive and vice versa. For this reason, we propose running these two procedures in one integrated heuristic that keeps the best of both results. Such an integrated heuristic could reach a solution in half the time of TRA5 and thus lead to very competitive results.

Reference

Al-Salem, A. (2004). A Heuristic to Minimize Makespan in Proportional Parallel Flow Shops. *International Journal of Computing & Information Sciences*, *2*(2), 98.

Allahverdi, A. (2015). The third comprehensive survey on scheduling problems with setup times/costs. *European Journal of Operational Research*, *246*(2), 345–378.

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

Companys, R. (1966). Métodos heurísticos en la resolución del problema del taller mecánico. *Estudios Empresariales*, *5*(2), 7–18.

Fernandez-Viagas, V., & Framinan, J. M. (2014). A bounded-search iterated greedy algorithm for the distributed permutation flowshop scheduling problem. *International Journal of Production Research*, *53*(4), 1111–1123.

Gao, J., & Chen, R. (2011). An NEH-based heuristic algorithm for distributed permutation flowshop scheduling problems, *6*(14), 3094–3100.

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

Gao, J., Chen, R., Deng, W., & Liu, Y. (2012). Solving multi-factory flowshop problems with a novel variable neighbourhood descent algorithm. *Journal of Computational Information Systems*, *8*(5), 2025–2032.

Gong, H., Tang, L., & Duin, C. W. (2010). A two-stage flow shop scheduling problem on a batching machine and a discrete machine with blocking and shared setup times. *Computers & Operations Research*, *37*(5), 960–969.

He, D. W., Kusiak, A., & Artiba, A. (1996). A scheduling problem in glass manufacturing. *IIE Transactions* , *28*(2), 129–139.

Jiang, Y., & Wan, S. (2011). Parallel flow shop scheduling problem using quantum algorithm. In *International Conference*

on Applied Informatics and Communication (pp. 269-274). Springer, Berlin, Heidelberg.
Johnson, S. M. (1954). Optimal two-and three-stage production schedules with set up times included. *Naval Research Logistics Quarterly*, *1*(1), 61–68.
Lin, S.-W., Ying, K.-C., & Huang, C.-Y. (2013). Multiprocessor task scheduling in multistage hybrid flowshops: A hybrid artificial bee colony algorithm with bi-directional planning. *Computers & Operations Research*, *40*(5), 1186–1195.
Liu, H., & Gao, L. (2010). A discrete electromagnetism-like mechanism algorithm for solving distributed permutation flowshop scheduling problem. In *2010 International Conference on Manufacturing Automation* (pp. 156–163). IEEE
Martinez, S., Dauzère-Pérès, S., Guéret, C., Mati, Y., & Sauer, N. (2006). Complexity of flowshop scheduling problems with a new blocking constraint. *European Journal of Operational Research*, *169*(3), 855–864.
McCormick, S. T., Pinedo, M. L., Shenker, S., & Wolf, B. (1989). Sequencing in an assembly line with blocking to minimize Cycle Time. *Operations Research*, *37*(6), 925–936.
Miyata, H. H., & Nagano, M. S. (2019). The blocking flow shop scheduling problem: A comprehensive and conceptual review. *Expert Systems with Applications*, *137*, 130–156.
Naderi, B., & Ruiz, R. (2010). The distributed permutation flowshop scheduling problem. *Computers & Operations Research*, *37*(4), 754–768.
Naderi, B., & Ruiz, R. (2014). A scatter search algorithm for the distributed permutation flowshop scheduling problem. *European Journal of Operational Research*, *239*(2), 323–334.
Nawaz, M., Enscore Jr, E. E., & Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, *11*(1), 91–95.
Palmer, D. S. (1965). Sequencing jobs through a multi-stage process in the minimum total time- a quick method of obtaining a near optimum. *Operational Research Quarterly*, *16*(1), 101–107.
Ribas, I., & Companys, R. (2015). Efficient heuristic algorithms for the blocking flow shop scheduling problem with total flow time minimization. *Computers & Industrial Engineering*, *87*, 30–39.
Ribas, I., Companys, R., & Tort-Martorell, X. (2017). Efficient heuristics for the parallel blocking flow shop scheduling problem. *Expert Systems with Applications*, *74*, 41–54.
Ribas, I., Companys, R., & Tort-Martorell, X. (2019). An iterated greedy algorithm for solving the total tardiness parallel blocking flow shop scheduling problem. *Expert Systems with Applications*, *121*, 347–361.
Sethi, S. P., Sriskandarajah, C., Sorger, G., Blazewicz, J., & Kubiak, W. (1992). Sequencing of parts and robot moves in a robotic cell. *International Journal of Flexible Manufacturing Systems*, *4*(3-4), 331–358.
Takano, M. I., & Nagano, M. S. (2019). Evaluating the performance of constructive heuristics for the blocking flow shop scheduling problem with setup times. *International Journal of Industrial Engineering Computations*, *10*(1), 37–50.
Trovinger, S. C., & Bohn, R. E. (2005). Setup time reduction for electronics assembly: Combining simple (SMED) and IT-based methods. *Production and Operations Management*, *14*(2), 205–217.
Wang, S., Wang, L., Liu, M., & Xu, Y. (2013). An effective estimation of distribution algorithm for solving the distributed permutation flow-shop scheduling problem. *International Journal of Production Economics*, *145*(1), 387–396.
Xu, Y., Wang, L., Wang, S., & Liu, M. (2013). An effective hybrid immune algorithm for solving the distributed permutation flow-shop scheduling problem. *Engineering Optimization*, *46*(9), 1269–1283.
Zhang, X., & Van De Velde, S. (2012). Approximation algorithms for the parallel flow shop problem. *European Journal of Operational Research*, *216*(3), 544–552.