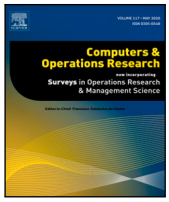
Computers & Operations Research 154 (2023) 106192

Contents lists available at ScienceDirect

Computers and Operations Research

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



Flowshop with additional resources during setups: Mathematical models and a GRASP algorithm

Juan C. Yepes-Borrero [a], Federico Perea [b,*], Fulgencia Villa [c], Eva Vallada [c]

[a] Universidad del Rosario. Escuela de Ingeniería, Ciencia y Tecnología, Cl. 12c N 6-25, 111711, Bogotá, Colombia
[b] Dpto. Matemática Aplicada II. Instituto de Matemáticas de la Universidad de Sevilla. Escuela Politécnica Superior. Edificio Celestino Mutis, Primera planta. Avda. Reina Mercedes, s/n, 41012 Sevilla, Spain
[c] Grupo de Sistemas de Optimización Aplicada, Instituto Tecnológico de Informática, Ciudad Politécnica de la Innovación, Edifico 8G, Acc. B. Universitat Politècnica de València, Camino de Vera s/n, 46021, València, Spain

ARTICLE INFO

MSC:
90B06

Keywords:
Scheduling
Flowshop
Mathematical programming
GRASP

ABSTRACT

Machine scheduling problems arise in many production processes, and are something that needs to be consider when optimizing the supply chain. Among them, flowshop scheduling problems happen when a number of jobs have to be sequentially processed by a number of machines. This paper addressees, for the first time, the Permutation Flowshop Scheduling problem with additional Resources during Setups (PFSR-S). In this problem, in addition to the standard permutation flowshop constraints, each machine requires a setup between the processing of two consecutive jobs. A number of additional and scarce resources, e.g. operators, are needed to carry out each setup. Two Mixed Integer Linear Programming formulations and an exact algorithm are proposed to solve the PFSR-S. Due to its complexity, these approaches can only solve instances of small size to optimality. Therefore, a GRASP metaheuristic is also proposed which provides solutions for much larger instances. All the methods designed for the PFSR-S in this paper are computationally tested over a benchmark of instances adapted from the literature. The results obtained show that the GRASP metaheuristic finds good quality solutions in short computational times.

1. Introduction

In a Flowshop Scheduling Problem (FSP) a set of jobs has to be processed on a set of machines, following the same route (the jobs visit the machines in the same order). The most studied variant of the problem is the one that only considers permutations, that is, the order of the jobs is the same for all the machines. This variant is known as the permutation flowshop scheduling problem (PFSP) and has been studied in the literature since the 1950's, see Johnson (1954). The objective of PFSP is to find the best schedule so that an objective function is optimized. One of the objective functions most frequently optimized in the related literature is the maximum completion time, known as *makespan* and denoted by C_{max}. Regarding the complexity of the problem, Garey et al. (1976) prove that the decision version of PFSP is $NP-Complete$ for three or more machines.

Several assumptions are considered in a PFSP: (i) each job can only be processed by one machine at the same time; (ii) one machine can only process one job at a time; (iii) the processing of a job on a machine cannot be interrupted; (iv) all jobs are independent and available at time 0; (v) the machines are continuously available; (vi) if a machine is processing a job, the next job can wait in a machine queue. In a realistic production environment, setup times on machines between two consecutive jobs have to be considered. The setup time can be defined as the time needed to set the machine up to process a given job. That is, every time a job is finished on one machine, the machine has to be readjusted in order to process the next job. Moreover, additional and limited resources (human resources, tools, molds, etc.) are needed to carry out the setups. To the best of our knowledge, there is no literature that takes into account setups with additional and scarce resources in permutation flowshop scheduling problems. This more realistic variant of the PFSP is called the *Permutation Flowshop Scheduling problem with additional Resources during Setups* (PFSR-S). The main contributions of this paper are:

- the proposal of a new and more realistic version of the permutation flowshop problem, in which machine setups are needed between the processing of any two jobs and additional resources are necessary for such setups, denoted as PFSR-S.
- two Mixed Integer Linear Programming models (MILPs) for the PFSR-S, as well as an exact algorithm based on one of them.

* Corresponding author.
E-mail addresses: juanca.yepes@urosario.edu.co (J.C. Yepes-Borrero), perea@us.es (F. Perea), evallada@eio.upv.es (E. Vallada).

https://doi.org/10.1016/j.cor.2023.106192
Received 6 April 2022; Received in revised form 11 February 2023; Accepted 11 February 2023
Available online 23 February 2023

- a GRASP metaheuristic, which finds good feasible solutions in short CPU times.

The remainder of this paper is organized as follows. In Section 2, the related literature is analysed. In Section 3, a formal definition of the problem is given. Two mixed integer mathematical models and an exact method with three steps are introduced in Section 4. In Section 5, a GRASP algorithm for the PFSR-S is presented. Section 6 explains the computational experiments carried out to analyse the performance of the designed methods. Finally, in Section 7 some conclusions and future research avenues are given.

2. Literature review

The permutation flowshop scheduling problem has been widely studied in the literature over the last 70 years, starting with the pioneering work of Johnson (1954). Both exact methods and heuristic algorithms have been proposed to tackle this problem. Some of the most recent works related to heuristic algorithms for the PFSP are Baskar and Xavior (2021), Öztop et al. (2020), Kurdi (2020), Wu et al. (2020), Fernandez-Viagas et al. (2020), Fernandez-Viagas and Framinan (2019), Dubois-Lacoste et al. (2017), Amirghasemi and Zamani (2017) and Liu et al. (2017). The interested reader can find recent reviews on the PFSP in Fernandez-Viagas et al. (2017) and Neufeld et al. (2016). As for exact methods, Sadjadi et al. (2008) reviews mathematical models for scheduling problems.

The literature that considers setup times in machine scheduling problems is extensive. The interested reader is referred to the reviews of Allahverdi et al. (2008) and Allahverdi (2015). More specifically, the following references consider setups in flowshop problems: Lu et al. (2018) and Ríos-Mercado and Bard (1998). A review of flowshop problems with setup times is found in Cheng et al. (2000).

The literature regarding flowshop problems with additional resources is less extensive. Janiak (1988) deals with a version of the general non-permutation flow-shop scheduling problem in which the duration of each operation on certain machines depends on the allocated amount of a constrained resource. Figielska (2009) develops a heuristic algorithm to solve a scheduling problem in a two-stage flowshop, with parallel unrelated machines and additional renewable resources during the first stage and a single machine during the second stage, with the objective of makespan minimization. Later, the same author proposes the same problem but with modifications to the resources in Figielska (2014). In this paper, the renewable resources are shared among the stages so some quantities of the same resource can be used at different stages at the same time. More recently, Benda et al. (2019) developed a machine learning approach to solve a real-life case that can be classified as a hybrid flow shop scheduling problem with alternative resources, sequence-dependent setup times, limited intermediate buffers and blocking.

Although the literature on flowshop problems is quite extensive, to the best of our knowledge it contains no references to the problem considered in this paper, that is, the Permutation Flowshop Scheduling Problem with additional Resources during Setups (PFSR-S).

3. Formal definition

This section formally defines the Permutation Flowshop Scheduling problem with additional Resources during Setups (PFSR-S). This problem takes the following input data:

- A set of jobs $N = \{1, \dots, n\}$. Jobs are indexed by j, k, ℓ. For modelling purposes a dummy job, indexed as 0, is considered. Let $N_0 = N \cup \{0\}$.
- Each job needs to be sequentially processed on each of the following machines $M = \{1, \dots, m\}$. We assume that all jobs follow the same route on the machines, which is (without loss of generality) $1 - 2 - \cdots - m$.

Table 1
Processing times for each job on each machine.

p_{ij}	1	2	3	4
1	6	3	2	1
2	2	2	4	2

Table 2
Setup times for each job on each machine.

s_{1ij}	1	2	3	4
0	3	4	1	7
1	0	5	3	2
2	5	0	3	1
3	2	1	0	5
4	3	2	5	0

s_{2ij}	1	2	3	4
0	2	3	1	6
1	0	6	3	5
2	4	0	3	1
3	3	4	0	1
4	7	8	4	0

- The processing of job j on machine i takes $p_{ij} \in \mathbb{Z}^+$ units of time.
- Machine i needs a setup time of $s_{ijk} \in \mathbb{Z}^+$ time units between the processing of jobs j and k (if no other job is processed between them).
- There are $R_{\max} \in \mathbb{Z}^+$ units of an additional resource.
- The setup of machine i between jobs j, k needs $r_{ijk} \in \mathbb{Z}^+$ units of the additional resource, during the s_{ijk} units of time that the setup lasts.

The PFSR-S looks for a sequence of jobs, processed on all machines in a given order, so that no machine processes more than one job at the same time, no more than $R_{\max}$ units of the additional resource are used at any time and the makespan (defined as the completion time of the last job to be processed) is minimized. To illustrate, we now show an example of the PFSR-S.

Example 3.1. Consider an example with $n = 4$ jobs and $m = 2$ machines (modified from that in Rios-Mercado and Bard, 1999). The processing times and setups times are given in Table 1 and Table 2 respectively. Note that the first row (labelled 0) in Table 2 shows the initial setup time for each machine if the first job to be processed is the column job.

An optimal solution to the PFSP without additional resources would be the sequence $(3, 1, 2, 4)$, with a makespan of 24, as shown in Fig. 1 (top graph). We represent the processing of jobs in red, and the setups in grey. Blank areas represent idle times.

If the number of resources needed is constant for each setup, $r_{ijk} = 2 \ \forall \ i, j, k$, and the available number of resources is $R_{\max} = 3$, we note that the previous solution would not be feasible, because four resources are needed in time unit 1 and in time units $14 - 16$ as well (the last row of the figure shows the resource consumption in each time unit). The infeasibility relating to time unit 1 can easily be solved without increasing the makespan by postponing the setup of machine 2 before processing job 3 by one or two units of time (before the processing of job 3 actually starts). However, the excess of resources caused by the overlap of the machine setups between jobs $j = 1$ and $j = 2$ (time units $\{14, 15, 16\}$) may need a new sequence, which would possibly imply an increase in makespan. An optimal solution to this new problem (PFSR-S) is the sequence shown in Fig. 1, the bottom graph, with sequence $(3, 2, 4, 1)$, and a makespan of 26. Note that, in general, the optimal sequence for the problem without resources is not the same as the optimal sequence for the problem with resources.

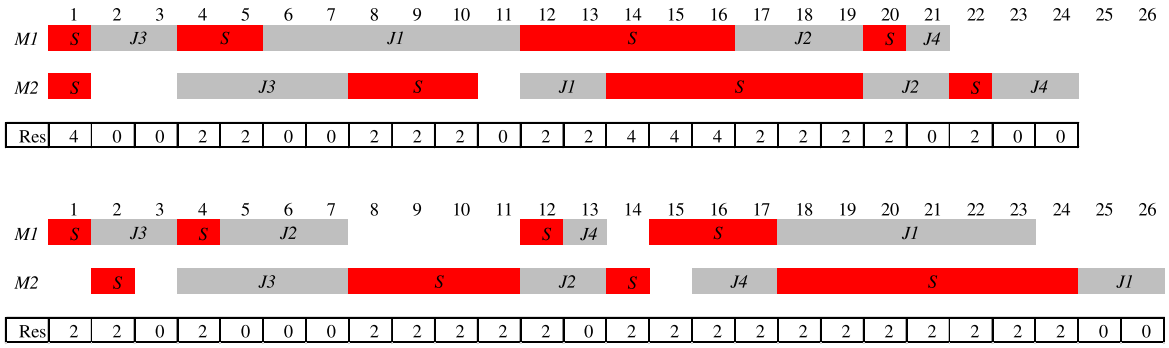


Fig. 1. Solution to Example 3.1.

4. Exact algorithms

In this section, three exact methods to solve the PFSR-S are presented: Two mixed integer linear programming models (MILP1 and MILP2) and one exact algorithm with three different steps (TPE) are considered. TPE combines the solution to the problem without taking into account the resource constraint for other solutions where resource constraints are satisfied.

4.1. MILP1: time index

In this section an initial mathematical programming model for the PFSR-S is shown, adapted from that found in Yepes-Borrero et al. (2020) for the parallel machine scheduling problem with resources during setups. Besides the sets introduced in Section 3, we need the set $T = \{1, \dots, t_{\max}\}$ to represent the time units, indexed by t. It can be noted that $t_{\max}$ is an upper bound on the makespan for the problem which needs to be computed. The concepts of consecutive jobs, predecessor and successor are now defined.

Definition 4.1. Given a solution $(j_1, j_2, \dots, j_n)$ to a PFSR-S, we say that two jobs $j, k \in N$ are consecutive if j is before k in the permutation, and no other job is between them. If the ordered job pair (j, k) is consecutive, we say that j is the predecessor of k, and k is the successor of j.

The model proposed uses the following variables:

- $X_{jk} = 1$ if the ordered job pair (j, k) is consecutive, zero otherwise. Note that, as opposed to the model in Yepes-Borrero et al. (2020), subindex i is not needed as the sequence is the same for all machines.
- $W_{ijkt} = 1$ if the setup of machine i between jobs j and k finishes at time t, zero otherwise.
- $C_{\max}$ denotes the makespan.

Once we have defined these variables, model MILP1 can be described as follows:

$$\min C_{\max} \tag{1}$$

$$\text{s.t.:} \sum_{j \neq k} X_{jk} = 1, \ k \tag{2}$$

$$\sum_{k \neq j} X_{jk} = 1, \ j \tag{3}$$

$$\sum_{t} W_{ijkt} = X_{jk}, \ i, j, k: \ k \neq j \tag{4}$$

$$\sum_{t} tW_{i,0,k,t} \geq s_{i,0,k} X_{0,k}, \ i, k: \ k \in N \tag{5}$$

$$\sum_{t} tW_{ijkt} \geq \sum_{\ell} \sum_{t} W_{i\ell jt}(t + p_{ij} + s_{ijk}) - M(1 - X_{jk}),$$
$$i, j, k: \ k \neq j \tag{6}$$

$$\sum_{t} tW_{i+1,j,k,t} \geq \sum_{t} tW_{ijkt} - s_{ijk} + p_{i+1,j} + s_{i+1,j,k} - M(1 - X_{jk}),$$
$$i, j, k: \ i < m, j \neq k, \tag{7}$$

$$\sum_{i,j,k \in N, k \neq j, t' \in \{t, \dots, t + s_{ijk} - 1\}} r_{ijk} W_{ijkt'} \leq R_{\max}, \ t \tag{8}$$

$$\sum_{t} tW_{mjkt} \leq C_{\max}, \ j, k: \ j \neq k \tag{9}$$

When no domain is specified for an index, all the values in the corresponding set are considered (N_0 for j, k, ℓ, M for i, T for t, t').

Constraints (2) ensure that any job k has a predecessor. Analogously, (3) ensures that any job j has a successor. Constraints (4) dictate that, if j precedes k, then W_{ijkt} takes a value of one for exactly one time instant t, for each machine i. (5) sets a lower bound on the time when the initial setup of machine i should finish. (6) ensures that, for any machine i, the end of the setup between any consecutive ordered job pair (j, k) has to finish, at least, when the setup on the previous machine finishes, plus the processing time of job j, plus the setup time between j and k. Note that, by abuse of notation, we use M both to denote this constant and to denote the set of machines. Constraints (7) impose that, if j precedes k, the time the setup on machine $i+1$ between these two jobs finishes must be, at least, the time the setup on the machine before (i) finishes, minus the setup of i, plus the processing time of j on $i+1$, plus the setup of machine $i+1$. Constraints (8) ensure that no more than $R_{\max}$ resources are used at any point of time t. Finally, (9) defines the makespan (note that the makespan of the schedule will always be given by the last machine). M is a big-M constant.

The number of variables and constraints in this model is enormous, partly due to the time index. Therefore, in the next section we propose another formulation which does not rely on a time index, but rather on formulations based on two-dimensional bin-packing problems.

4.2. MILP2: bin packing

In this section another MILP model is proposed. This second model, denoted as MILP2, is based on bin-packing models. Bin-packing approaches have proven to be successful in other scheduling problems with additional resources, see Fanjul-Peyro et al. (2017) and Fanjul-Peyro (2020). In MILP2, we assume that for each consecutive ordered job pair (j, k), there are m boxes (corresponding to the setups needed between them in each of the m machines), which are denoted as the 3-tuple (i, j, k). Such boxes will be placed on a two-dimensional chart, where the horizontal axis corresponds to time and the vertical axis corresponds to resources. Fig. 2 shows how the boxes in the two solutions given in Example 3.1 would be. It can be noted that in the first solution the maximum height of the boxes exceeds the horizontal line $R_{\max} = 3$, which is the availability of resources. This implies that the corresponding solution is infeasible. However, in the second solution the maximum height of the boxes is always below $R_{\max}$, which implies that the resource constraint is satisfied.

The following sets of variables are needed for this model:

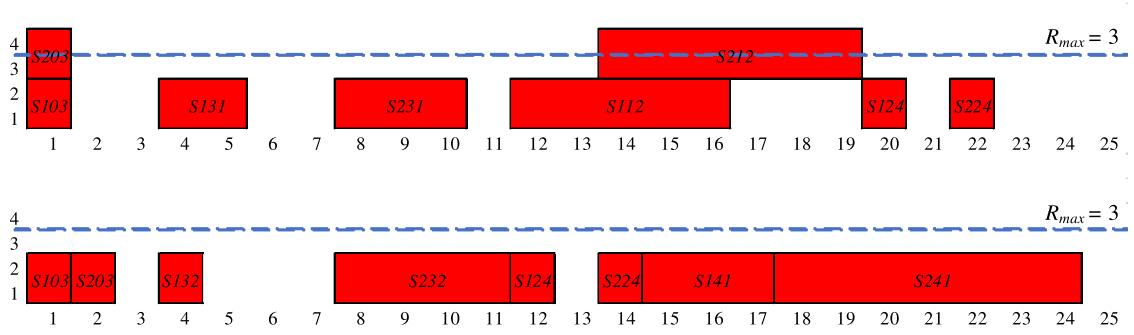


Fig. 2. Boxes in the two solutions given in Example 3.1.

- $X_{jk} = 1$ if the ordered job pair (j, k) is consecutive, zero otherwise.
- $H_{ijk} \geq 0$ is the time when the setup of machine i between jobs j, k finishes, that is, the horizontal coordinate of the top right corner of the box (i, j, k).
- $Y_{ijk} \geq 0$ is the vertical coordinate of the top right corner of the box (i, j, k), which is associated with the use of resources.
- $U_{ijki'j'k'}$ is a binary variable that takes a value of zero if a no-overlap constraint on the horizontal axis needs to be activated between the boxes (i, j, k) and (i', j', k'), being (i, j, k) right of (i', j', k'), and 1 otherwise.
- $V_{ijki'j'k'}$ is a binary variable that takes a value of zero if a no-overlap constraint on the vertical axis needs to be activated between the boxes (i, j, k) and (i', j', k'), being (i, j, k) top of (i', j', k'), and 1 otherwise.

In addition, the following set of indexes will be applied:

$$C = \{(i, j, k, i', j', k') : i \neq i', j \neq k, j' \neq k', j' \geq j, k \neq 0, k' \neq 0\}. \quad (10)$$

From these variables, model MILP2 consists of:

$$\min C_{\max} \quad (11)$$

$$\text{s.t.:} \sum_{j \neq k} X_{jk} = 1, \ k \quad (12)$$

$$\sum_{k \neq j} X_{jk} = 1, \ j \quad (13)$$

$$H_{ijk} \leq C_{\max}, \ i, j, k \quad (14)$$

$$Y_{ijk} \leq R_{\max}, \ i, j, k \quad (15)$$

$$s_{ijk} X_{jk} \leq H_{ijk}, \ i, j, k \quad (16)$$

$$MU_{ijki'j'k'} + H_{ijk} \geq H_{i'j'k'} + s_{ijk}, \ (i, j, k, i', j', k') \in C \quad (17)$$

$$MU_{i'j'k'ijk} + H_{i'j'k'} \geq H_{ijk} + s_{i'j'k'}, \ (i, j, k, i', j', k') \in C \quad (18)$$

$$M(1 - X_{kk'}) + H_{ikk'} \geq \sum_{j} H_{ijk} + p_{ik} + s_{ikk'}, \ i, k, k' : k \neq k', k \neq 0 \quad (19)$$

$$M(1 - X_{jk}) + H_{i+1,j,k} \geq H_{ijk} - s_{ijk} + p_{i+1,j} + s_{i+1,j,k}, \ k \neq j, i < m \quad (20)$$

$$r_{ijk} X_{jk} \leq Y_{ijk}, \ i, j, k \quad (21)$$

$$MV_{ijki'j'k'} + Y_{ijk} \geq Y_{i'j'k'} + r_{ijk}, \ (i, j, k, i', j', k') \in C \quad (22)$$

$$MV_{i'j'k'ijk} + Y_{i'j'k'} \geq Y_{ijk} + r_{i'j'k'}, \ (i, j, k, i', j', k') \in C \quad (23)$$

$$U_{ijki'j'k'} + U_{i'j'k'ijk} + V_{ijki'j'k'} + V_{i'j'k'ijk} \leq 3 + 2((1 - X_{jk}) + (1 - X_{j'k'})), \ (i, j, k, i', j', k') \in C \quad (24)$$

(12) ensures that each job k has exactly one job which precedes it. Analogously, (13) ensures that each job j precedes exactly one job. (14) properly defines the makespan. (15) ensures that no more resources than $R_{\max}$ are used at any time.

The next sets of constraints model the time axis (horizontal). (16) sets a lower bound on the end of the setup of (i, j, k). This is needed if $j = 0$. For other values of j this represents a feasible cut. (17) states that, if variable $U_{ijki'j'k'}$ takes a value of zero, then (i, j, k) is on the right of (i', j', k') and does not overlap on the horizontal axis. Conversely, (18) states that, if variable $U_{i'j'k'ijk}$ takes a value of zero, then (i', j', k') is on the right of (i, j, k) and does not overlap on the horizontal axis. (19) ensures that for each machine i, if k precedes k', their setup on i cannot finish before the end of the setup before k, plus the processing time of k, plus the setup time between k and k'. (20) states that, if j precedes k, their setup on machine $i + 1$ cannot finish before the beginning of their setup on machine i, plus the processing time of j on $i + 1$ plus their setup time on machine $i + 1$.

The next constraints model the resources axis (vertical axis). (21) sets a lower bound on Y variables for consecutive jobs. (22) states that, if $V_{ijki'j'k'}$ takes a value of zero, then (i, j, k) is on top of (i', j', k') and does not overlap on the vertical axis. Conversely, (23) states that, if $V_{i'j'k'ijk}$ takes a value of zero, then (i', j', k') is on top of (i, j, k) and does not overlap on the vertical axis.

Combining both axis, (24) ensures that if j precedes k and j' precedes k', then at least one of the four no-overlap constraints should be activated between setups (i, j, k) and $(i', j'.k')$, and therefore, they should not overlap on at least one dimension.

4.3. Three-phase exact algorithm (TPE)

The two previous models do not solve instances with more than 10 jobs, as demonstrated in Section 6.1. In order to improve the makespan of the best solution found by the MILP, and to reduce the GAPs, we design a three-phase exact (TPE) algorithm consisting of the following steps:

1. Solving the problem without resources by means of a MILP (denoted as MILP-NR). Store $X_{NR}, C^{NR}_{\max}$, the optimal sequence found and makespan respectively. Note that this solution may be infeasible for PFSR-S.
2. Solving the problem with resources, imposing that the sequence is X_{NR}, and that $C^{NR}_{\max}$ is a lower bound for $C_{\max}$, by means of MILP2 (the new model is denoted as MILP-Restricted). The solution to this problem $(X_R, C^R_{\max})$ is a feasible solution for PFSR-S. Note that if $C^R_{\max} = C^{NR}_{\max}$, then we are certain that $(X_R, C^R_{\max})$ is an optimal solution for PFSR-S, since $C^R_{\max}$ is a lower bound.
3. If $(X_R, C^R_{\max})$ is not optimal for PFSR-S, solving the problem with resources by means of MILP2, without imposing any value on any variable, but providing the solver with the feasible solution found in step 2 (this model is denoted as MILP-Aided).

These steps are summarized in Algorithm 1.

Despite the improvement provided by the TPE with respect to the MILPs proposed, the PFSR-S seems to be too complex to be addressed by exact algorithms. Therefore, in the next section a metaheuristic algorithm is proposed, which aims to find (good) feasible solutions in short CPU times.

Algorithm 1: Three-Phase Exact algorithm.

$TimeAvailable$;
Solve MILP-NR $\Rightarrow (X_{NR}, C^{NR}_{\max})$;
Update $TimeAvailable$;
Solve MILP-Restricted $\Rightarrow (X_R, C^{R}_{\max})$;
if $C^{NR}_{\max} = C^{R}_{\max}$ **then**
  $(X^*, C^*_{\max}) = (X_R, C^{R}_{\max})$;
else
  Update $TimeAvailable$;
  Solve MILP-Aided with allowed CPU time
   $TimeAvailable \Rightarrow (X^*, C^*_{\max})$;
end
Return the solution $(X^*, C^*_{\max})$.

5. GRASP algorithm

To find solutions to the PFSR-S in reasonable times, we propose a GRASP (Greedy Randomized Adaptive Search Procedure) algorithm. GRASP algorithms were introduced by Feo and Resende (1995) and are one of the most commonly used multi-start methods. A complete GRASP iteration has two phases:

- a constructive phase that consists of building a partial solution (see Section 5.1).
- a local search phase in order to improve on the solution found in the constructive phase (see Section 5.2)

However, due to the nature of the PFSR-S, the solution obtained after these two phases might be a non-feasible solution, because it uses more resources those available. Therefore, a repairing phase is proposed, in which the previous solution is evaluated and (if necessary) repaired so that it satisfies the resource constraints.

In short, the aim of this algorithm is to build good solutions in the constructive phase (not necessarily feasible), try to improve on them in the local search, and then, if the solution obtained is not feasible, to repair it by making it feasible (satisfying the resource constraints) in the repairing phase. Algorithm 2 shows the general procedure of our GRASP algorithm.

Algorithm 2: General GRASP algorithm procedure

$BestSol \leftarrow LargeNumber$
while $ExecutionTime < timeLimit$ **do**
  $CurrentSol \leftarrow$ Constructive phase;
  $CurrentSol \leftarrow$ Local search($CurrentSol$);
  $CurrentSol \leftarrow$ Repairing phase($CurrentSol$);
  if $BestSol > CurrentSol$ **then**
    $BestSol \leftarrow CurrentSol$
  end
end
Return $BestSol$;

In the rest of this section we detail each of the phases of our GRASP algorithm. Two constructive phases are proposed. In the first one, information about the resources is used and the resulting GRASP algorithm is denoted as GRASP_R. At the end of this section we propose another constructive phase in which information about the resources is not considered. The resulting GRASP algorithm is denoted as GRASP_NR.

5.1. Constructive phase

To build the initial solution, a list with all the jobs not assigned yet is created (not-assigned jobs are called *pending jobs*). Initially, the set of pending jobs is N, and the partial solution is empty. At each iteration, one candidate is randomly chosen from a restricted candidate list (RCL), which is assigned to the partial solution. The RCL is created at each iteration by testing the insertion of each pending job into all possible positions in the partial solution built so far (the solution with all assigned jobs until the current iteration). At each insertion, we evaluate a value based on the λ value proposed in Yepes-Borrero et al. (2020). The insertions with lower λ are the best candidates (see Section 5.1.1). Note that each candidate is a job-position pair with its respective λ value.

The size of the RCL depends on a parameter $\alpha \in [0, 1]$ as follows:

$$RCL(\alpha) = \{(j,k) : \lambda_{jk} \leq BestValue + \alpha(WorstValue - BestValue)\}, \quad (25)$$

where $BestValue$ is the minimum value of λ and $WorstValue$ is the maximum value of λ, that is

$$BestValue = \min_{j,k} \lambda_{jk}, \; WorstValue = \max_{j,k} \lambda_{jk}. \quad (26)$$

If $\alpha = 1$, $RCL(\alpha)$ contains all possible insertions, and therefore the algorithm is completely random. If $\alpha = 0$, $RCL(\alpha)$ contains only the best candidate (or all candidates which attain the best λ value), therefore, the algorithm is completely greedy. The value of α is calibrated in Section 6. Algorithm 3 summarizes this constructive phase.

Algorithm 3: Summary of the constructive phase

$PendingJobs \leftarrow N$;
$PartialSolution \leftarrow \emptyset$;
while $PendingJobs \neq \emptyset$ **do**
  foreach $j \in PendingJobs$ **do**
    foreach $k \in \{1, ..., n - |PendingJobs| + 1\}$ **do**
      Calculate λ_{jk};
    end
  end
  $BestValue = \min_{jk} \lambda_{jk}$, $WorstValue = \max_{j,k} \lambda_{jk}$;
  $RCL(\alpha) = \{(j,k) : \lambda_{jk} \leq$
   $BestValue + \alpha(WorstValue - BestValue)\}$;
  Randomly choose $(j^*, k^*) \in RCL(\alpha)$;
  Update $PartialSolution$: Assign job j^* in position k^*;
  Remove j^* from $PendingJobs$;
end

5.1.1. Computation of λ

As in Yepes-Borrero et al. (2020), the idea of the λ value is to evaluate both the makespan and the resource consumption that would result when inserting one pending job j into one specific position k, over a partial solution already created by the algorithm during that iteration. Note that, if the current partial solution already has n^* jobs assigned, then there are $n^* + 1$ possible positions where we can insert a pending job j. Besides, note that when doing such insertions generally, new setups will be done, and some setups will no longer be done. Fig. 3 shows an example of inserting a new job j into position k, which is between jobs l and l'. In Fig. 3 (a), in the partial solution jobs $l - l'$ are consecutive, which implies that setup $s_{ill'}$ is needed. In Fig. 3 (b) job j is inserted between jobs l and l'. Note that setup $s_{ill'}$ should no longer be done, but two new setups should be done (s_{ilj} and $s_{ijl'}$).

This information, when inserting job j in position k, is summarized by λ_{jk} as follows:

$$\lambda_{jk} = \sum_{i \in M} (p_{ij} + (\theta_{s_{ijk}} \theta_{r_{ijk}}) - (\gamma_{s_{ijk}} \gamma_{r_{ijk}})) \quad (27)$$

where:

- $\theta_{s_{ijk}}$ is the sum of all new setup times when inserting job j in position k on machine i. Note that in general, if a job is inserted into a partial solution, more than one new setup must be done. In the example in Fig. 3, we have $\theta_{s_{ijk}} = s_{ilj} + s_{ijl'}$.
- $\theta_{r_{ijk}}$ is the sum of all new resources needed to do the new setups required when inserting job j into position k on machine i. In the example, we have $\theta_{r_{ijk}} = r_{ilj} + r_{ijl'}$.

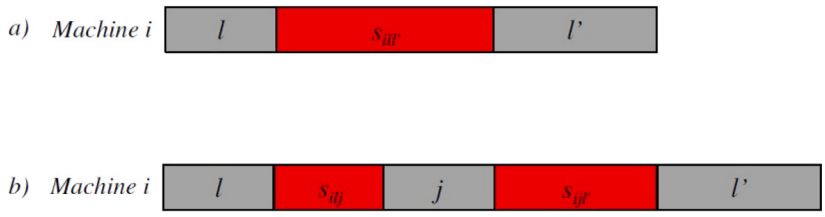


Fig. 3. Example of job insertion.

- $\gamma_{s_{ijk}}$ is the setup time that should no longer be done when inserting job j into position k on machine i. Note that if a job is inserted into a partial solution, in general, the setup that was previously in that position should no longer be done. In the example, we have $\gamma_{s_{ijk}} = s_{ill'}$.
- $\gamma_{r_{ijk}}$ is the number of resources needed to do the setup that should no longer be done when inserting job j in position k on machine i. In the example, we have $\gamma_{r_{ijk}} = r_{ill'}$.

It is important to note that there is a summation in the calculation of λ, this is because the insertion of a job on all the machines must be tested. In short, with this value we are seeking sequences with a good makespan which do not require too many resources.

5.2. Local search

After the constructive phase, a local search phase on the sequence generated is applied. Given a sequence of jobs $\{1, 2, \dots, j-1, j, j+1, \dots, j'-1, j', j'+1, \dots, n\}$, each pair of jobs j, j' is swapped, in this way creating the new sequence $\{1, 2, \dots, j-1, j', j+1, \dots, j'-1, j, j'+1, \dots, n\}$. $\lambda_{jj'}$ is computed as explained before. The swap with lower λ value (more negative) is kept and the process is repeated. If no swap has $\lambda < 0$, the local search ends. Algorithm 4 summarizes the local search procedure.

Algorithm 4: Summary of the local search.

```
Improvement ← 1;
while Improvement ← 1 do
    Improvement ← 0;
    BestImprovement ← 0;
    foreach j ∈ N do
        foreach j' ∈ N \ {j} do
            Calculate λ_jj';
            if λ_jj' < BestImprovement then
                BestImprovement = λ_jj';
                Improvement ← 1;
                BestSwap ← (j, j');
            end
        end
    end
    Do BestSwap;
end
```

5.3. Repairing phase

It is important to note that the solutions obtained in the constructive phase are not necessarily feasible solutions. In fact, the aim in the construction phase is to find solutions which are not necessarily feasible, but rather solutions with shorter setup times and lower consumption of resources in order to make it easy to repair them in the repairing phase. The reader should note that, after the Local Search phase, solutions are not necessarily feasible either.

In this repairing phase, the consumption of resources at each instant t, denoted as R_t, is evaluated. If $R_t > R_{\max}$ for some $t \in \{0, \dots, C_{\max}\}$, the solution is not feasible and must be repaired. When an instant t is found in which the resource constraint is not satisfied, we postpone the beginning of the setup by one time unit on the last machine that is doing setups at instant t. Afterwards, the start time of the processing of the jobs on the following machines must be evaluated because it is possible that by postponing the beginning of a setup, the processing start time of some jobs may also change. Remember that in the flowshop problem, the start time of job j on machine i must be greater than the completion time of that job on the previous machine $i-1$. After evaluating the start time of the jobs on the following machines, the evaluation of resources is repeated from the instant where the feasibility problem appeared. This process is repeated until the resource constraints are met for all $t \in \{0, \dots, C_{\max}\}$. Note that $C_{\max}$ may change during the process and, therefore, $C_{\max}$ is updated at each iteration. Algorithm 5 summarizes the repairing phase procedure.

Algorithm 5: Summary of the repairing phase.

```
t = 0;
while t < C_max do
    Calculate R_t;
    if R_t > R_max then
        Postpone the beginning of the setup that is being done at
          instant t and starts the setup the latest;
        Update C_max;
    else
        t++;
    end
end
```

5.4. GRASP without resources in the constructive phase

The reader may note that, when computing λ (see Eq. (27)), the information about the resources is considered by means of $\theta_{r_{ijk}}$ and $\gamma_{r_{ijk}}$. The resulting algorithm is denoted as GRASP_R. In order to evaluate if such a use of information about resources is effective in the final GRASP algorithm, a different computation of λ is proposed in which information about resources is not considered. In other words, in order to find the best candidates for insertions (job j and position k), only the makespan of such an insertion is evaluated. To do this, the computation of λ is modified in such a way that $\theta_{r_{ijk}}$ and $\gamma_{r_{ijk}}$ are fixed to 1. In this way, resource needs do not affect the assessment of each potential insertion. The resulting algorithm will be denoted as GRASP_NR, and will be tested together with the other algorithms in the next section.

6. Computational experiments

To assess the performance of the models and algorithms proposed in this paper, an experimental study was performed accross a benchmark of instances. First, the proposed benchmark is explained in detail and then the results of each study are shown and discussed.

All experiments are run on virtual machines with 2 virtual processors and 8 GB of RAM memory each under Windows 10 Enterprise 64 Bits. The GRASP algorithm has been coded in Microsoft Visual Studio 2019 using C#. The MILP models have been coded in GAMS and solved with CPLEX 12.10. Instances and complete results are available from the authors upon request.

6.1. Benchmark of instances

We test our algorithms over two sets of instances, named Set_1 and Set_2 respectively. Both are based on the one used in Vallada and Ruiz (2011), originally proposed for the unrelated parallel machine scheduling problem with setups without additional resources. Those instances have been adapted to the PFSR-S and completed by adding the resource needs (r_{ijk}) and the number of available resources ($R_{\max}$).

Set_1 has instances with number of jobs $n \in \{5, 10\}$ and number of machines $m = 2$. We set $R_{\max} = m = 2$. Each combination of (n, m) is replicated 10 times as follows, where $U\{a, b\}$ denotes an integer uniform distribution between $a \in \mathbb{Z}$ and $b \in \mathbb{Z}$.

- Setup times: $s_{ijk} \in U\{1, 9\}$.
- Resource needs: $r_{ijk} \in U\{1, R_{\max}\} = U\{1, m\}$.
- Job processing times: $p_{ij} \in U\{1, 99\}$.

Therefore, Set_1 consists of $2 \times 10 = 20$ instances, which are used to analyse the performance of the exact algorithms proposed in Section 4.

Set_2 has instances with $n \in \{10, 20, 30, 40, 50, 60\}$, $m \in \{5, 10, 15, 20\}$ and four different distributions for the generation of setup times, namely $U\{1, 9\}$, $U\{1, 49\}$, $U\{1, 99\}$ and $U\{1, 124\}$. By combining the six different values of n, the four different values of m and the four different distributions for the setup times we have $6{\times}4{\times}4 = 96$ different configurations. In all of them $R_{\max} = m$. Each configuration has been randomly replicated 10 times, generating p_{ij} and r_{ijk} as in Set_1. Then, Set_2 has in total $96 \times 10 = 960$ instances.

In order to compare the proposed algorithms, the Relative Percentage Deviation (RPD) is computed for each algorithm and instance, according to (28).

$$RPD = \frac{C_{\max}(alg) - C_{\max}(best)}{C_{\max}(best)} \cdot 100, \tag{28}$$

where $C_{\max}(alg)$ is the makespan for the solution obtained with the tested algorithm and $C_{\max}(best)$ is the best known makespan for the instance.

6.2. Results of exact approaches

The set of instances Set_1 is used to analyse the performance of the three exact methods: MILP1, MILP2 and TPE. These three approaches can optimally solve instances with $n = 5$ in less than 10 s. However 10-job instances are much harder to solve optimally. After two hours of computational time, the worst model is the time-index model (MILP1). This model is not able to even find a feasible solution for instances with 10 jobs. Therefore, only the performance of MILP2 and TPE will be further analysed.

Table 3 shows for each 10-job instance the value of the objective function (Z) and the lower bound (LB) for MILP2 and each step of the TPE algorithm (pointing out that the solution after Step 1 is, in general, infeasible). Both MILP2 and TPE algorithm were run for two hours (7200 s) without proving any optimal solutions. It can be noted that the average value of the solutions obtained by MILP2 is 351.9 (columns MILP2), whereas the average for the TPE algorithm is 328.2 (columns TPE step3), which implies an improvement of 7% on average. However, it should also be highlighted that the third step of the TPE algorithm did not improve on any of the solutions found in the second step (columns TPE step2). This second step is a matheuristic (optimal solution assuming the sequence is that given by the permutation in the flowshop scheduling problem without resources). The first two steps of TPE solve the corresponding MILPs to optimality (the lower bounds are equal to the objective function values achieved). In addition, it was shown that MILP2 achieved almost the same results in one hour of CPU time as in two hours of CPU time, meaning that the model is not expected to improve on results even if more CPU time was allowed. MILP1 was not tested, as it did not even yield a feasible solution in the time available.

Table 3
Objective function value (Z) and lower bound (LB) for MIPL2 and each step of TPE method for 10-job instances. Both methods are run for two hours.

	MILP2		TPE step1		TPE step2		TPE step3	
	Z	LB	Z	LB	Z	LB	Z	LB
ID1	328	65	273	273	325	325	325	273
ID2	342	47	295	295	376	376	372	295
ID3	359	70	283	283	357	357	357	283
ID4	318	58	263	263	298	298	298	263
ID5	356	65	251	251	278	278	278	251
ID6	375	51	292	292	308	308	308	292
ID7	373	65	299	299	350	350	350	299
ID8	343	73	275	275	331	331	331	275
ID9	379	62	288	288	322	322	322	288
ID10	346	75	269	269	341	341	341	269
Average	351.9	63.1	278.8	278.8	328.6	328.6	328.2	278.8

Table 4
CPU time in seconds for each step of TPE method for 10-job instances.

	TPE step1	TPE step2	TPE step3
ID1	605	5	6590
ID2	877	5	6318
ID3	693	4	6502
ID4	679	3	6518
ID5	588	3	6609
ID6	793	4	6403
ID7	937	4	6259
ID8	724	4	6472
ID9	642	3	6555
ID10	740	4	6456
Average	727.7	4.1	6468.2

Table 4 shows the computational time, in seconds, spent by each step of the TPE algorithm. It can be observed that most of the CPU time is devoted to the third phase (10% for the first phase, 0.05% for the second and 89.95% for the third). While step 1 (finding the optimal solution to the problem without resources) is relatively hard to solve, and step 3 (improving the solution given by step 2) is very hard, we note that step 2 (repairing the sequence given by step 1) is very fast. As seen above, in the third phase the algorithm did not manage to improve on the lower bound found in the first phase, nor the upper bound found in the second. The solutions after step 2 are 7% better than the solutions given by MILP2, using around 10.05% of the computational time.

6.3. Results of GRASP

The instances in Set_1 and Set_2 are solved with the two versions of our GRASP algorithm, named GRASP_NR (which does not consider information about resources during the constructive phase, see Section 5.4) and GRASP_R (which does consider information about resources during the constructive phase, see Section 5) respectively. For both GRASPs the stopping criterion is CPU time, which depends on the size of the instance and is equal to $n \times m$ seconds.

6.3.1. Calibration of the RCL size

It is necessary to calibrate the value of the α parameter to run both versions of the GRASP because it directly affects the size of the restricted candidate list (RCL). With this aim, 96 instances from Set_2 (one for each combination of n, m and the different distributions for the setup times) have been selected as a calibration set.

Table 5 shows the average RPD for the different α levels for both algorithms (GRASP_R and GRASP_NR). We observe that there are differences among the different α levels, with $\alpha = 0.25$ being the best value in both algorithms.

These values suggest that the value of α does affect the average RPD. However, in order to check if these differences are statistically significant, a statistical analysis is needed. An analysis of variance

Table 5
Average RPDs for different values of α in both GRASP algorithms.

Algorithm	$\alpha = 0$	$\alpha = 0.25$	$\alpha = 0.5$	$\alpha = 0.75$	$\alpha = 1$
GRASP_R	0.088	0.005	0.075	0.140	0.167
GRASP_NR	0.088	0.007	0.084	0.141	0.166

Table 6
Performance of GRASP_R and GRASP_NR in 5-job instances, with $\alpha = 0.25$.

	Z^*	GRASP_R		GRASP_NR	
		Z	RPD (%)	Z	RPD (%)
ID1	198	205	3.5	255	28.8
ID2	209	211	1.0	368	76.1
ID3	204	217	6.4	358	75.5
ID4	123	125	1.6	196	59.3
ID5	138	140	1.4	269	94.9
ID6	178	204	14.6	351	97.2
ID7	227	231	1.8	370	63.0
ID8	161	180	11.8	276	71.4
ID9	224	238	6.3	389	73.7
ID10	150	166	10.7	217	44.7
Average	181.2	191.7	5.9	304.9	68.5

(ANOVA, see Montgomery 2019), cannot be applied, since for our data, the assumptions of normality and homoscedasticity are not fulfilled. Therefore, a non-parametric Kruskal–Wallis is performed (see Kruskal and Wallis 1952). In this analysis, RPD is the response variable and the only factor considered is $\alpha \in \{0, 0.25, 0.5, 0.75, 1\}$, while a different test is done for each algorithm. For both algorithms, the p-value obtained in the test is less than 2×10^{-16}. Therefore, the equality of means is rejected, and we conclude that different levels of α yield significantly different average RPDs.

The previous analysis confirms that not all values of α yield the same average RPD. In order to find the best level for this factor, a non-parametric Wilcoxon test (see Wilcoxon 1992) is performed between the two values of α that returned the lowest average RPD in the sample analysed, namely $\alpha = 0.25$ and $\alpha = 0.5$. For both algorithms, the p-value obtained in this test is less than 2×10^{-16}. Therefore, we can state that there are significant differences between $\alpha = 0.25$ and $\alpha = 0.5$, with $\alpha = 0.25$ being the best value in both cases.

6.3.2. Performance of GRASP in Set_1 instances

Tables 6 and 7 show the performance of both GRASP_NR and GRASP_R for each 5-job and 10-job instance respectively. In Table 6, column Z^* is the optimal solution for each 5-job instance. In Table 7, column Z_{best} is the best solution found by any of the following methods: MILP2, TPE, GRASP_R and GRASP_NR. In both tables, the third and fourth columns show the value of the objective function provided by the GRASP_R and its RPD, and the fifth and sixth columns show the same for the GRASP_NR. Both tables show that including resource consumption information provides better solutions. In addition, GRASP_R provides 50% of the best solutions for 10-job instances in 20 s, as opposed to the 7200 s employed by the exact algorithms introduced in this paper.

6.3.3. Performance of GRASP in Set_2 instances

Tables 8 and 9 show the average RPD (%) over the 960 instances in Set_2 for GRASP_R and GRASP_NR, according to the number of jobs and machines respectively. As before, GRASP_R performs better than GRASP_NR. Figs. 4 and 5 show that as the problem size increases, the difference between the two methods becomes more evident.

7. Conclusions and future research

In this paper, the aim is to reduce the gap between academic research and real-life scheduling in flow shop problems. The permutation flowshop scheduling problem with additional resources during setups (PFSR-S) is introduced for the first time. Efficient smart tools have been designed to solve the PFSR-S: mixed integer programming models, an exact algorithm and a metaheuristic. Simulation using a benchmark adapted from the scheduling literature allows us to state that exact methods do not perform well in this problem due to its enormous complexity. On the other hand, a metaheuristic such as the GRASP algorithm designed in this paper seems like a reasonable option, since it yields good quality solutions in reduced computational times. For the GRASP algorithm, it has been confirmed that information about resources should be considered during the constructive phase, because the solutions returned yield lower relative percentage deviations with respect to best solutions known. Additionally, the size of the RCL (controlled by means of a parameter $\alpha \in [0, 1]$), should be reduced, but not by too much. More specifically, it has been empirically verified that $\alpha = 0.25$ yields the best results in terms of relative percentage deviations. Future research on this topic will focus on bi-objective approaches to simultaneously minimize the makespan and the use of resources.

Table 7
Performance of GRASP_R and GRASP_NR in 10-job instances, with $\alpha = 0.25$.

	Z_{best}	GRASP_R		GRASP_NR	
		Z	RPD (%)	Z	RPD (%)
ID1	325	349	7.4	482	48.3
ID2	342	349	2.0	518	51.5
ID3	332	332	0.0	555	67.2
ID4	298	298	0.0	524	75.8
ID5	278	302	8.6	524	88.5
ID6	304	304	0.0	606	99.3
ID7	345	345	0.0	576	67.0
ID8	331	352	6.3	553	67.1
ID9	318	318	0.0	585	84.0
ID10	341	355	4.1	464	36.1
Average	321.4	330.4	2.9	538.7	68.5

Table 8
Average RPDs of GRASP_R and GRASP_NR depending on the number of jobs for $\alpha = 0.25$.

	10	20	30	40	50	60	Total
GRASP_R	0.51	0.04	0.00	0.00	0.00	0.00	0.09
GRASP_NR	3.26	8.91	10.99	12.38	13.51	14.43	10.58

Table 9
Average RPDs of GRASP_R and GRASP_NR depending on the number of machines for $\alpha = 0.25$.

	5	10	15	20
GRASP_R	0.10	0.07	0.07	0.12
GRASP_NR	7.99	11.17	11.42	11.74

CRediT authorship contribution statement

Juan C. Yepes-Borrero: Methodology, Software, Writing – review & editing. **Federico Perea:** Conceptualization, Methodology, Software, Writing – review & editing. **Fulgencia Villa:** Methodology, Data curation, Writing – review & editing. **Eva Vallada:** Methodology, Investigation, Writing – review & editing.

Data availability

Data will be made available on request.

Acknowledgements

Juan C. Yepes-Borrero acknowledges financial support by Colfuturo under program Crédito-Beca grant number 201503877 and from El Instituto Colombiano de Crédito Educativo y Estudios Técnicos en el

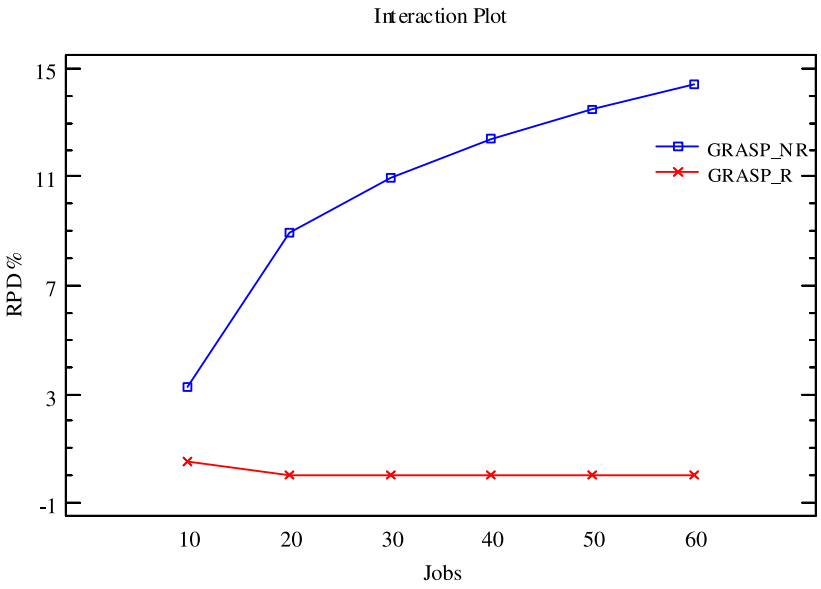


Fig. 4. Average RPDs of GRASP_R and GRASP_NR depending on the number of jobs for $\alpha = 0.25$.

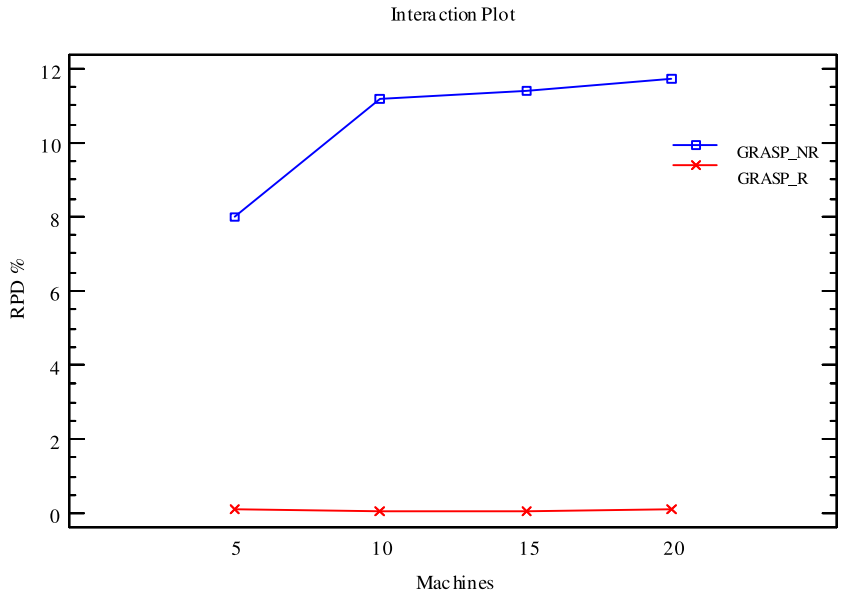


Fig. 5. Average RPDs of GRASP_R and GRASP_NR depending on the number of machines for $\alpha = 0.25$.

Exterior - ICETEX under program Pasaporte a la ciencia - Doctorado, Foco-reto país 4.2.3, grant number 3568118. This research has been partially supported by the Agencia Estatal de Investigación (AEI) and the European Regional Development's fund (ERDF): PID2020-114594GB-C21; Regional Government of Andalusia: projects FEDER-US-1256951, AT 21_00032, and P18-FR-1422; Fundación BBVA: project NetmeetData (Ayudas Fundación BBVA a equipos de investigación científica 2019). The authors are partially supported by Agencia Valenciana de la Innovación (AVI) under the project "ireves (innovación en vehículos de emergencia sanitaria): una herramienta inteligente de decisión" (No. INNACC/2021/26) partially financed with FEDER funds (interested readers can visit http://ireves.upv.es), and by the Spanish Ministry of Science and Innovation under the project "OPRES-Realistic Optimization in Problems in Public Health" (No. PID2021-124975OB-I00), partially financed with FEDER funds. Part of the authors are supported by the Faculty of Business Administration and Management at Universitat Politècnica de València.

References

Allahverdi, A., 2015. The third comprehensive survey on scheduling problems with setup times/costs. European J. Oper. Res. 246 (2), 345–378.

Allahverdi, A., Ng, C.T., Cheng, T.C.E., Kovalyov, M.Y., 2008. A survey of scheduling problems with setup times or costs. European J. Oper. Res. 187 (3), 985–1032.

Amirghasemi, M., Zamani, R., 2017. An effective evolutionary hybrid for solving the permutation flowshop scheduling problem. Evol. Comput. 25 (1), 87–111.

Baskar, A., Xavior, M.A., 2021. New idle time-based tie-breaking rules in heuristics for the permutation flowshop scheduling problems. Comput. Oper. Res. 133, 105348.

Benda, F., Braune, R., Doerner, K.F., Hartl, R.F., 2019. A machine learning approach for flow shop scheduling problems with alternative resources, sequence-dependent setup times, and blocking. OR Spectrum 41 (4), 871–893.

Cheng, T.E., Gupta, J.N., Wang, G., 2000. A review of flowshop scheduling research with setup times. Prod. Oper. Manage. 9 (3), 262–282.

Dubois-Lacoste, J., Pagnozzi, F., Stützle, T., 2017. An iterated greedy algorithm with optimization of partial solutions for the makespan permutation flowshop problem. Comput. Oper. Res. 81, 160–166.

Fanjul-Peyro, L., 2020. Models and an exact method for the unrelated parallel machine scheduling problem with setups and resources. Expert Syst. Appl.: X 5.

Fanjul-Peyro, L., Perea, F., Ruiz, R., 2017. Models and matheuristics for the unrelated parallel machine scheduling problem with additional resources. European J. Oper. Res. 260, 482–493.

Feo, T.A., Resende, M.G., 1995. Greedy randomized adaptive search procedures. J. Global Optim. 6 (2), 109–133.

Fernandez-Viagas, V., Framinan, J.M., 2019. A best-of-breed iterated greedy for the permutation flowshop scheduling problem with makespan objective. Comput. Oper. Res. 112, 104767.

Fernandez-Viagas, V., Molina-Pariente, J.M., Framinan, J.M., 2020. Generalised accelerations for insertion-based heuristics in permutation flowshop scheduling. European J. Oper. Res. 282 (3), 858–872.

Fernandez-Viagas, V., Ruiz, R., Framinan, J.M., 2017. A new vision of approximate methods for the permutation flowshop to minimise makespan: State-of-the-art and computational evaluation. European J. Oper. Res. 257 (3), 707–721.

Figielska, E., 2009. A genetic algorithm and a simulated annealing algorithm combined with column generation technique for solving the problem of scheduling in the hybrid flowshop with additional resources. Comput. Ind. Eng. 56 (1), 142–151.

Figielska, E., 2014. A heuristic for scheduling in a two-stage hybrid flowshop with renewable resources shared among the stages. European J. Oper. Res. 236 (2), 433–444.

Garey, M., Johnson, D., Sethi, R., 1976. The complexity of flowshop and jobshop scheduling. Math. Oper. Res. 1, 117–129.

Janiak, A., 1988. General flow-shop scheduling with resource constraints. Int. J. Prod. Res. 26 (6), 1089–1103.

Johnson, S.M., 1954. Optimal two- and three-stage production schedules with setup times included. Nav. Res. Logist. Q. 1, 61–68.

Kruskal, W.H., Wallis, W.A., 1952. Use of ranks in one-criterion variance analysis. J. Amer. Statist. Assoc. 47 (260), 583–621.

Kurdi, M., 2020. A memetic algorithm with novel semi-constructive evolution operators for permutation flowshop scheduling problem. Appl. Soft Comput. 94, 106458.

Liu, W., Jin, Y., Price, M., 2017. A new improved NEH heuristic for permutation flowshop scheduling problems. Int. J. Prod. Econ. 193, 21–30.

Lu, S., Liu, X., Pei, J., Pardalos, P.M., 2018. Permutation flowshop manufacturing cell scheduling problems with deteriorating jobs and sequence dependent setup times under dominant machines. Optim. Lett. 1–15.

Montgomery, D.C., 2019. Design and Analysis of Experiments, tenth ed. John Wiley & Sons, New York.

Neufeld, J.S., Gupta, J.N., Buscher, U., 2016. A comprehensive review of flowshop group scheduling literature. Comput. Oper. Res. 70, 56–74.

Öztop, H., Tasgetiren, M.F., Eliiyi, D.T., Pan, Q.-K., Kandiller, L., 2020. An energy-efficient permutation flowshop scheduling problem. Expert Syst. Appl. 150, 113279.

Ríos-Mercado, R.Z., Bard, J.F., 1998. Computational experience with a branch-and-cut algorithm for flowshop scheduling with setups. Comput. Oper. Res. 25 (5), 351–366.

Rios-Mercado, R.Z., Bard, J.F., 1999. A branch-and-bound algorithm for permutation flow shops with sequence-dependent setup times. IIE Trans. 31 (8), 721–731.

Sadjadi, S., Aryanezhad, M., Mohsen, Z., 2008. The general flowshop scheduling problem: Mathematical models. J. Appl. Sci. 8.

Vallada, E., Ruiz, R., 2011. A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. European J. Oper. Res. 211 (3), 612–622.

Wilcoxon, F., 1992. Individual comparisons by ranking methods. In: Kotz, S., Johnson, N.L. (Eds.), Breakthroughs in Statistics: Methodology and Distribution. Springer New York, New York, NY, pp. 196–202.

Wu, P., Yang, Q., Chen, W., Mao, B., Yu, H., 2020. An improved genetic-shuffled frog-leaping algorithm for permutation flowshop scheduling. Complexity 2020.

Yepes-Borrero, J.C., Villa, F., Perea, F., Caballero, J.P., 2020. GRASP algorithm for the unrelated parallel machine scheduling problem with setup times and additional resources. Expert Syst. Appl. 141, 1–12.