Document downloaded from:

http://hdl.handle.net/10251/102339

This paper must be cited as:

The final publication is available at

http://doi.org/10.1016/j.ejor.2017.01.002

Copyright Elsevier

Additional Information

MIP models and matheuristics for the unrelated parallel machine scheduling problem with additional resources

Luis Fanjul-Peyro, Federico Perea, Rubén Ruiz

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.
Email: lfpeyro@hotmail.com, perea@eio.upv.es, rruiz@eio.upv.es

Abstract

In this paper we analyze a parallel machine scheduling problem in which the processing of jobs on the machines requires a number of units of a scarce resource. This number depends both on the job and on the machine. The availability of resources is limited and fixed throughout the production horizon. The objective considered is the minimization of the makespan. We model this problem by means of two integer linear programming problems. One of them is based on a model previously proposed in the literature. The other one, which is based on the resemblance to strip packing problems, is an original contribution of this paper. As the models presented are incapable of solving medium-sized instances to optimality, we propose three matheuristic strategies for each of these two models. The algorithms proposed are tested over an extensive computational experience. Results show that the matheuristic strategies significantly outperform the mathematical models.

Keywords: Parallel machine problem, Scheduling, Additional Resources, Matheuristics, Makespan.

1. Introduction and motivation

In a competitive world, in which many tasks are processed in great part (if not completely) by machines, the need for intelligent organization is a must. This need is even more imperative if one considers that the energy upon which we rely, namely food, fuel and electricity, etc., is limited. In addition to this, the commercial sector has the objective of making a profit and this

profit could be increased if optimal functionality in their production plants was achieved. The particular problem we address in this paper explores this area. We assume that a number of jobs are to be processed by a number of parallel machines or production lines, under certain conditions, and with a certain objective which will be specified later. This problem is found in many manufacturing settings, such as car factories and food processing plants, etc. In this paper we provide algorithms that help in assigning tasks to machines in an automatized way. A distinction with respect to the majority of papers dealing with scheduling of parallel machines is that we consider additional resources (for example plant personnel) that are limited and must be available to operate machinery. This extra constraint makes the problem much more complex, as will be explained throughout the paper.

The classical *unrelated parallel machine scheduling problem* (UPM) consists of processing a number of jobs on a number of parallel machines. The most common objective is the minimization of the so-called *makespan* (maximum job completion time). Each job has to be processed by exactly one machine, and processing times need not be the same for all *unrelated* machines. This problem has been mathematically termed $\mathcal{NP}$-Hard. As a matter of fact, the simpler version with two identical parallel machines was already demonstrated to be $\mathcal{NP}$-Hard by Lenstra et al. [22]. It should be noted that, in these problems, the sequence in which the jobs are processed is not relevant, as opposed to the much more complex problem we deal with in this paper. The UPM arises in many production systems. [25] can be considered one of the first papers dealing with this topic. Ever since then, the interest from the scientific community regarding the UPM has continued to increase. For general applications and reviews of the UPM, the reader is referred to books such as [28], [29], [32], or to reviews of parallel machine scheduling problems such as [1] and [21].

In the scientific literature, most of the scheduling problems treated on parallel machines do not consider that the machines need an extra resource to be able to function. This represents a gap between academic research, and the actual needs of the production sector. Such extra resources could be, for example, the operators needed to work machines. In this paper we will analyze the UPM with this additional consideration, where a fixed number of resources are needed to process the jobs on the machines. More specifically, we assume that machines need a *discrete* amount of a scarce *renewable* resource to process jobs. This amount depends both on the job and on the machine, so as to make the problem more realistic. As in [3], [5], and [30], among others,

in this paper we assume that the resources are: 1) *renewable*, because after the processing of a job, the resources used are again available for other jobs. 2) *discrete*, because the number of resources needed is a positive integer and 3) *processing*, because they are only needed while the job is being processed. We refer to the resulting problem as the *Unspecified Dynamic Unrelated Parallel Machine Scheduling problem with additional Resources* (UPMR). *Unspecified*, because there is no pre-fixed job-machine assignment, and *dynamic* in the sense that the allocation of resources to machines need not be fixed for the whole time horizon. The specified and/or static version of this problem has already been treated in the literature (see Section 2). However, and to the best of our knowledge, the unspecified and dynamic variant has been seldom studied.

The UPMR problem takes the following input data: a list of m available machines (indexed by i and i'); a list of n jobs to be processed (indexed by j and j'); $R_{\max}$ units of a certain resource; $p_{ij} \in \mathbb{Z}^+$ units of time and $r_{ij} \in \mathbb{Z}^+$ units of the resource, which needed to process job j at machine i, $\forall\ i = 1, ..., m, j = 1, ..., n$.

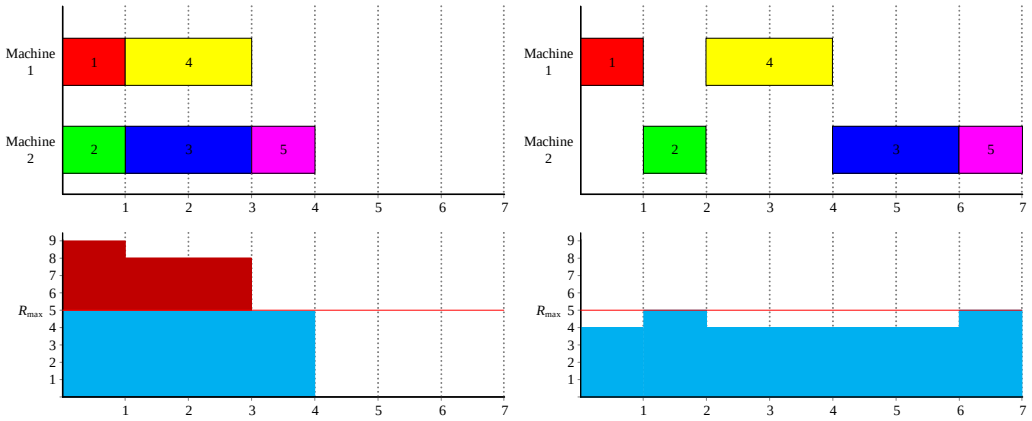
The objective is the minimization of the makespan (denoted by $C_{\max}$), and the following constraints have to be satisfied: the same machine cannot process more than one job at the same time; each job must be processed by exactly one machine; the processing of a job cannot be interrupted before completion; no more than $R_{\max}$ units of the resource can be used at any time.

The following example illustrates problem UPMR and the differences between this problem and the regular and much simpler unrelated parallel machine scheduling problem (UPM).

Example 1.1. *Consider the following instance of an UPMR with two machines ($m = 2$), five jobs ($n = 5$), five units of a scarce resource ($R_{\max} = 5$) and the following processing times and resource needs:*

$$P = \begin{pmatrix} 1 & 2 & 2 & 2 & 1 \\ 2 & 1 & 2 & 3 & 1 \end{pmatrix}; \quad R = \begin{pmatrix} 4 & 3 & 3 & 4 & 2 \\ 2 & 5 & 4 & 2 & 5 \end{pmatrix},$$

where P and R are the $n \times m$ matrices whose entries are the processing times p_{ij} and resource needs r_{ij} of the assignment machine i and job j, respectively. The unrelated parallel machine scheduling problem would have as optimal makespan $C_{\max} = 4$. The optimal solution is shown in Figure 1a, top graph. As we can see in the bottom graph, the maximum availability of resources is



(a) Optimal solution without resources.

(b) Solution with resources.

Figure 1: Optimal solution for the example without resources and after considering resources. The top graphs are Gantt diagrams representing the solutions, and the corresponding bottom graphs show the use of resources (vertical axis) per time unit (horizontal axis).

violated by four units, between time 0 and time 1, and by three units between time 1 and time 3. Constructing a resource-feasible solution from the optimal solution without resources and keeping the jobs assigned to the same machines, results in the solution of Figure 1b. Resources are not overused now but the makespan has increased to seven units and both machines incur in idle-times.

If we calculate the optimal solution considering resources for the same problem we obtain the solution given in Figure 2. As can be observed, resources are better used, machines are now always busy, and the makespan has only increased one unit with respect to the resource-unconstrained solution, $C_{\max} = 5$. *This new solution is, however, completely different from the solution without resources as no job is assigned to the same machine. The sequence of jobs has also changed in both machines. This example shows that the sequence for the problem without resources might yield a suboptimal solution when the resource constraint is added.*

As has been shown in Example 1.1, the unrelated parallel machines problem with resources is different from the problem without resources. The regular parallel machines problem is just an assignment problem, and the only decision to be made is which machine each job must be assigned to. Assigned jobs are processed in any order until completion, and the machines are never idle in between jobs. The version with resources is much more complex as the

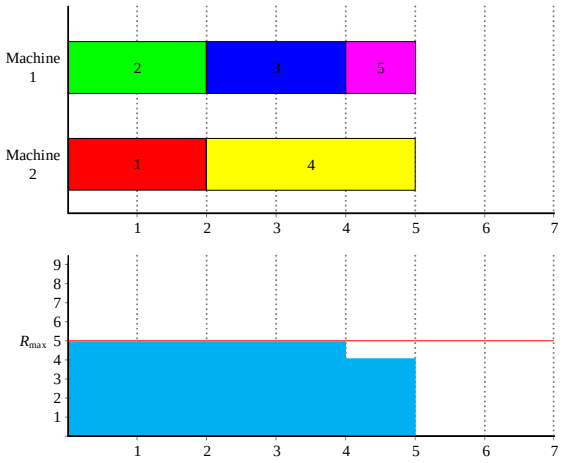


Figure 2: Optimal solution considering resources.

assignment, and also the starting and completion times, must be determined. Moreover, at times machines might be unable to process the next job due to resource shortages and idle times might appear.

The rest of the paper is structured as follows: Section 2 presents an overview of the literature related to the problem studied. Section 3 introduces two different mixed integer linear programming (MILP) models, which are used later on as a basis for several matheuristic strategies in Section 4. All these algorithms are computationally tested in Section 5. The paper closes with some conclusions and pointers to future research.

2. Literature review

Although not as old as the version without additional resources, the UPMR (or close variations of it) have been studied in the literature for the last three decades. Back in the eighties, [4] studied a simplified UPMR problem in which all machines are identical, and therefore requirements for additional resources and processing times only depend on the job in order to minimize the flow time. The authors showed that unless the resource needs are unitary, the problem is $\mathcal{NP}$-Hard in the strong sense. Also in relation to identical parallel machines, [34] proved that the problem is polynomially solvable, when there is only one type of resource, the jobs need either zero or one unit of this resource to be processed and the objective is the minimization of the deviations of the completion times with respect to a given due date.

The static version of the problem, which assumes that the allocation of resources to machines is given and fixed during the whole time horizon, has already been proposed and studied in the literature. For instance [7] studied the static identical parallel-machine flexible-resource scheduling problem with unspecified job assignment, for which MILP programs and heuristics were proposed. Some dynamic versions have also been studied. For example, [16] and [17] studied the problem in which the processing times depend on the number of resources allocated. [18] proposed a $(3.5 + \varepsilon)$-approximation algorithm for a simplified version of the same problem, in which all machines are identical.

Similarly to the static version of the UPMR, but with less strict constraints, is that of the *dedicated* machines. This problem assumes that the sets of jobs are divided into m groups, and the members of group i can only be processed on machine i. [19] and [20] studied the complexity of this problem and proposed several algorithms, for the case of one type of resource and more than one respectively.

More recently, [8] studied another simplified version of the UPMR, in which the number of resources needed to process a job does not depend on the machine on which the job is to be processed. The aim is to minimize the total completion time of the jobs (different from makespan), and the authors applied a Lagrangian-Based constraint programming approach. [11] proposed an integer programming model, a constraint programming model and a combined IP/CP model for a UPMR problem with machine eligibility constraints (not all jobs can be processed on all machines). These models are later applied to a real instance by the same authors in [12], in which two different resources are present: machine operators and a certain tool to process jobs.

Our research is novel as it consists of an in-depth study of two mathematical models for the more complex unspecified and dynamic variant of the UPMR. One of them is an adaptation of that presented in [9] and [10], whereas the other one is an original contribution of this paper, based on packing problem models. Several matheuristics are proposed which are able to solve moderately sized instances.

3. MILP modelling

In this section, two different MILP formulations are proposed. The first one is based on a model previously introduced in the literature. The second

one models the UPMR problem as a special packing problem. To the best of our knowledge, this second model is original and the main theoretical contribution of this paper. Some other models based on adaptations of UPM models were tested. However, they yielded worse results and for the sake of brevity, they are not detailed in this paper.

3.1. A MILP based on previous research

In this section we adapt the MILP program introduced by [9] (page 441, constraints (43) to (48)) to the UPMR studied in this paper. Another and (perhaps) more similar generalization of the resource constrained parallel machine scheduling problem is given by [10] (page 452, constraints (12) to (16)). The model presented in [9] assumes that the number of resources assigned affects the processing times. This is different from the UPMR presented here, which assumes that a fixed amount of resources are needed for processing jobs on machines, and this number may not be changed. As well as index i for machines and index j for jobs, we need index k to denote the time. As opposed to i and j, which are clearly bounded by m and n, respectively, the maximum time in which a job can be processed, denoted by $K_{\max}$, is not trivial to obtain. Further discussions about $K_{\max}$ are given in the experimental section.

The first MILP program for the UPMR (denoted by UPMR-S) uses the following variables:

- $x_{ijk} = 1$ if job j is assigned to machine i and completes its processing at time k, and zero otherwise. Note that this variable only exists for $k \geq p_{ij}$.
- $C_{\max}$ is the makespan.

Model UPMR-S consists of minimizing $C_{\max}$, subject to the following constraints:

$$\sum_i \sum_{k \geq p_{ij}} k x_{ijk} \leq C_{\max}, \ \forall \ j, \tag{1}$$

$$\sum_i \sum_{k \geq p_{ij}} x_{ijk} = 1, \ \forall \ j, \tag{2}$$

$$\sum_j \sum_{s \in \{\max\{k, p_{ij}\}, \ldots, k+p_{ij}-1\}} x_{ijs} \leq 1, \forall \ i, k, \tag{3}$$

$$\sum_i \sum_j \sum_{s \in \{\max\{k, p_{ij}\}, \ldots, k+p_{ij}-1\}} r_{ij} x_{ijs} \leq R_{\max}, \forall \ k. \tag{4}$$

Constraints (1) determine the makespan. Constraints (2) dictate that each job is assigned to exactly one machine, and finishes at exactly one time. Constraints (3) ensure that the same machine does not process more than one job at any time. Constraints (4) require that no more than $R_{\max}$ units of resource are used at any time.

3.2. A packing problem based model

In this section, we use the ideas obtained from bin-packing formulations to model UPMR as a mixed integer linear programming program. In 2D bin-packing problems, the objective is to place a set of rectangular items into a rectangular case. For a survey of two-dimensional packing problems see [23]. When one of the dimensions of the case is fixed, the problem is known as a strip packing problem. In this problem, the objective is to place a set of rectangles into the case so that the length of the other dimension is minimized. Given its resemblance to the Gantt diagrams of the scheduling problems (see figures 1 and 2), we will consider that the height of the case (denoted by H, which represents the units of resource) is fixed, and the width (denoted by W, which represents the makespan) is to be minimized so that all items fit in the case.

In the strip-packing problem, the n rectangles to be placed in the strip have width w_k and height h_k, for $k = 1, ..., n$. For the sake of notation, and without loss of generality, we will denote the items as ij (since they correspond to machine-job assignments) instead of k. If we consider each rectangle to be a job-machine assignment (previously fixed), we have that the width of ij is $w_{ij} = p_{ij}$, and its height is $h_{ij} = r_{ij}$. In strip-packing problems, the objective is to find the location of each rectangle, whose top-right corner coordinates can be denoted by variables x_{ij} and y_{ij} (the x-axis will represent time and

the y-axis will represent the resources). Therefore, the height of the case H represents the maximum allowed units of resource $R_{\max}$, and its width W represents the makespan $C_{\max}$, to be minimized. Figure 3 shows a graphical representation of these ideas.

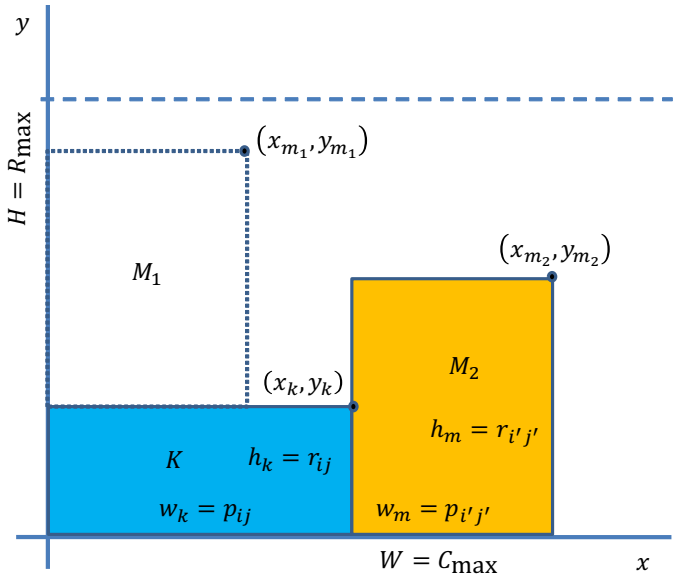


Figure 3: Strip Packing: new rectangle M can be set in positions M_1 or M_2, among other possibilities, with no overlap in x-axis or y-axis with the previously set rectangle K.

In the UPMR problem, jobs are not pre-assigned to any machine. Therefore, we need to define a set of m cases for each job: one for each machine. Because jobs are processed by only one machine, for each job we must select only one case out of this set. Therefore, for the UPMR problem model based on strip-packing (denoted by UPMR-P), we need the next set of binary variables:

- $s_{ij} = 1$ if job j is assigned to machine i, and zero otherwise.

The reader should note that, for any pair of rectangles ij and $i'j'$, we need to ensure that they do not overlap. We therefore define two kinds of no-overlap constraints:

- Horizontal-axis no-overlap constraint

$$x_{ij} - x_{i'j'} \geq p_{ij}. \tag{5}$$

Note that this constraint imposes that j finishes its processing p_{ij} units of time after j' is processed. In order to prevent overlap on the horizontal axis we impose either this constraint, or the one exchanging ij with $i'j'$.

- Vertical-axis no-overlap constraint

$$y_{ij} - y_{i'j'} \geq r_{ij}. \tag{6}$$

Note that this constraint imposes that the rectangle corresponding to job j is located r_{ij} units above the rectangle of job j'. In order to prevent overlap on the vertical axis we impose either this constraint, or the one exchanging ij with $i'j'$.

We should note that, if $i = i'$ we only need to impose the horizontal-axis no-overlap constraints (i.e. jobs are assigned to the same machine, and they will finish one after the other, so they do not share resources), whereas if $i \neq i'$, we need to impose both constraints. In order to decide which of these constraints are activated, for each pair of rectangles ij and $i'j'$ we define the following two sets of binary variables

- $u_{iji'j'} = 0$ if the horizontal-axis no-overlap constraint (5) is activated for pairs of jobs ij and $i'j'$, and takes value one otherwise (i.e. job j finishes processing later than j').

- $v_{iji'j'} = 0$ if the vertical-axis no-overlap constraint (6) is activated for pairs of jobs ij and $i'j'$, and takes value one otherwise (i.e. job j is located above job j').

Therefore, the MIP model (denoted by UPMR-P) for our problem is:

$$\min C_{\max} + \frac{1}{M} \sum_j \sum_i x_{ij} \tag{7}$$

$$\text{s.t.:} \sum_i s_{ij} = 1 \ \forall \ j, \tag{8}$$

$$s_{ij} p_{ij} \leq x_{ij} \leq C_{\max}, \quad \forall \ i, j, \tag{9}$$

$$s_{ij} r_{ij} \leq y_{ij} \leq R_{\max} s_{ij}, \quad \forall \ i, j, \tag{10}$$

$$M u_{iji'j'} + x_{ij} - x_{i'j'} \geq p_{ij}, \ \forall \ i, j, i', j' \geq j + 1, \tag{11}$$

$$M u_{i'j'ij} + x_{i'j'} - x_{ij} \geq p_{i'j'}, \ \forall \ i, j, i', j' \geq j + 1, \tag{12}$$

$$u_{iji'j'} + u_{i'j'ij} \leq 1 + (1 - s_{ij}) + (1 - s_{i'j'}), \ \forall \ i, j, i', j' \geq j + 1, i = i', \tag{13}$$

$$M v_{iji'j'} + y_{ij} - y_{i'j'} \geq r_{ij}, \ \forall \ i, j, i', j' \geq j + 1, \ i' \neq i, \tag{14}$$

$$M v_{i'j'ij} + y_{i'j'} - y_{ij} \geq r_{i'j'}, \ \forall \ i, j, i', j' \geq j + 1, \ i' \neq i, \tag{15}$$

$$u_{iji'j'} + u_{i'j'ij} + v_{iji'j'} + v_{i'j'ij} \leq 3 + 2((1 - s_{ij}) + (1 - s_{i'j'})),$$
$$\forall \ i, j, i', j' \geq j + 1, \ i' \neq i. \tag{16}$$

The objective function (7) minimizes $C_{\max}$ and forces x_{ij} to zero whenever this is possible (if job j is not assigned to machine i). Out of several possibilities tested to achieve this (among others, the natural constraints $x_{ij} \leq M s_{ij}$), the one that demonstrated the best performance is the one aforementioned. Constraints (8) ensure that each job j is only assigned to one machine. Constraints (9) and (10), respectively, set the bounds for the horizontal and vertical axis, setting this bound for the vertical axis to zero if s_{ij} is zero. Constraints (11) and (12) are the horizontal-axis no-overlap constraint. Constraints (13) ensure that, if j and j' are processed by the same machine i, then either j finishes p_{ij} units of time after j' has finished ($u_{ijij'} = 0$) or j' finishes $p_{ij'}$ units of time after j has finished ($u_{ij'ij} = 0$). Note that (13) are relaxed if $s_{ij} = 0$ or if $s_{ij'} = 0$. Constraints (14) and (15) are the vertical-axis no-overlap constraints. Finally, constraints (16) impose that if j and j' are processed by two different machines i and i', then at least one of the four binary variables $u_{iji'j'}, u_{i'j'ij}, v_{iji'j'}, v_{i'j'ij}$ should be equal to zero. It should be noted that this implies that there is no overlap in, at least, one dimension. Note that (16) are relaxed when s_{ij} or $s_{i'j'}$ are zero. The main strength of this model is that it does not rely on a time index, as opposed to UPMR-S, with the consequent reduction of the number of variables.

4. Matheuristic strategies

Matheuristics can be broadly defined as algorithms in which metaheuristic techniques are combined with mathematical programming models. Although a relatively new concept, it has been gaining more and more attention over recent years and it has recently been applied to many combinatorial optimization problems. To cite two examples, a project scheduling problem is solved using matheuristics in [33], and some location problems in [31]. Within scheduling, [2] and [6] use matheuristic approaches for single machine scheduling problems. However, for as far as we know, there are no matheuristic approaches for many parallel machine scheduling problems, and none with the consideration of additional resources.

In this section, three matheuristic strategies based on the MILP models introduced in Section 3 are shown.

- The first matheuristic, called *machine-assignmet fixing*, assigns jobs to machines by solving the UPM problem, to later on solve the UPMR problem in which the job-machine assignment is the one given by the UPM solution. We then solve the UPMR having such an assignment fixed. We used this strategy because both the UPM and specified UPMR problem are much faster to solve than the unspecified UPMR.

- The second one, called *job-machine reduction*, reduces the set of potential job-machine assignments by discarding, for each job, those machines that yield the largest processing times. This strategy was chosen because of the good results obtained in the UPM (see [14]).

- The third one, called *greedy-based fixing*, is a greedy strategy, in which the problem is sequentially solved for small subsets of jobs, keeping the assignments found in the previous iterations fixed. This strategy was chosen both for its simplicity and for the extensive, and mainly successful, use that greedy algorithms and their variants have had in the scheduling literature.

These methods are further detailed in the following sections.

4.1. Machine-assignment fixing

This algorithm firstly solves the regular unrelated parallel machine problem minimizing the makespan (UPM problem without the resource constraints), which can be done very rapidly in a solver, as shown in [13]. We will see

in the experimental section that all instances tested can easily be solved to optimality if the resources are ignored. The solution to this problem, referred to as x^*, gives an assignment of jobs to machines without specifying when jobs are processed (note that this is irrelevant in the UPM with the makespan criterion). In a second phase, the MILP program of the UPMR is relaxed in such a way that the variables that assume any job-assignment other than the one obtained by UPM are fixed to zero.

For example, assuming that job j is assigned to machine i, $x^*_{ij} = 1$ in the UPM model, only those variables in the UPMR model are considered for every job j. In the UPMR-S model, those variables are x_{ijk}, and in the UPMR-P model are x_{ij} and y_{ij}. This means that, for variables x_{ijk}, the i and j indexes are fixed as per the optimal values in x^* and only the k subindex needs to be found. The procedure is similar for the model UPMR-P. We have named this matheuristic machine-assignment fixing (MAF), of which a pseudocode in shown in Algorithm 1.

Name: MAF
Define $x_{ij} = 1$ if machine i processes job j, zero otherwise;
Let x^* be a solution to the UPM problem;
Solve UPMR-S considering only x_{ijk} such that $x^*_{ij} = 1$;
or
Solve UPMR-P considering only x_{ij} and y_{ij} such that $x^*_{ij} = 1$;
Algorithm 1: Machine-assignment fixing algorithm (MAF).

4.2. Job-machine reduction

This algorithm aims at reducing the large amount of variables present in the MILP models defined before. This is done in such a way that for every job j, only the "best" machines can be used. By best, we mean here those machines with the shortest processing times for each job. The number of potential machines selected for each job, here denoted as $\ell \in \mathbb{Z}^+$, is a parameter of the algorithm, and will be tested in the experimental section.

For example, for each job j we set variables to zero for all machines i so that the processing time p_{ij} is not among the ℓ smallest in the list $\{p_{ij}, i = 1, ..., m\}$. Actually, these variables are not even defined in the final model. This method is referred to as job-machine reduction (JMR), of which a pseudocode is given in Algorithm 2.

Name: $JMR(\ell)$
for $j = 1, ..., n$ **do**
  Let $\{i_1, ..., i_m\} : p_{i_1 j} \leq p_{i_2 j} \leq ... \leq p_{i_m j}$;
  (in UPMR-S) Remove $x_{ijk}\ \forall\ i \notin \{i_1, ..., i_\ell\}$;
  or
  (in UPMR-P) Remove x_{ij} and $y_{ij}\ \forall\ i \notin \{i_1, ..., i_\ell\}$;
end
Solve UPMR-S or UPMR-P;

Algorithm 2: Job-machine reduction algorithm (JMR) for a fixed value of ℓ.

4.3. Greedy-based fixing

This sequential algorithm, referred to as GBF, works as follows. At each iteration, a group of $g \in \mathbb{Z}^+$ jobs are selected, solving the UPMR problem for these jobs only. Then, the UPMR problem for other g jobs is solved taking into account the solution obtained before. The process continues until all jobs have been assigned to machines and has a scheduled starting time. This strategy closely resembles the K-greedy algorithms, used to solve some combinatorial problems, like data association in [27].

Initial experiments quickly showed that it was beneficial to consider previously scheduled jobs and to include them in the next iteration. Therefore, GBF first solves the UPMR with g unscheduled jobs. Then, at each subsequent iteration, the job scheduled last on the machine generating the makespan and $g-1$ new unscheduled jobs are added to the problem. Then the UPMR model is solved considering that the previously scheduled $g-1$ jobs are fixed for the rest of the algorithm. Therefore, there is one first iteration with g unscheduled jobs. Then we have $\lfloor \frac{n-g}{g-1} \rfloor$ iterations, each one with one previously scheduled job and $g-1$ new unscheduled jobs, where $\lfloor \cdot \rfloor$ denotes the integer part operator. There might be a final iteration with the $\Big((n-g) \bmod (g-1)\Big) + 1$ final unscheduled jobs, where "mod" denotes the module operator.

The GBF matheuristic firstly requires jobs to be sorted according to some criterion. Jobs are sorted in a decreasing order of the sum of their fastest machine processing times. More specifically, for each job we first sort the processing times on the m machines and obtain a list $\{i_1, i_2, \ldots, i_m\}$, so that $p_{i_1 j} \leq p_{i_2 j} \leq \ldots \leq p_{i_m j}$. Then we add up the first $\lfloor \frac{m}{2} \rfloor$ of these processing times, $P_{j_{50\%}} = \sum_{l=1}^{\lfloor \frac{m}{2} \rfloor} p_{i_l j}$. Jobs are then sorted in decreasing order of $P_{j_{50\%}}$.

When selecting, at each iteration, the job scheduled last on the machine generating the makespan, there might be ties, i.e., two machines with the same maximum completion time equaling the makespan value. In this case, the job with the largest resource consumption is selected. In the rare event in which there is also a tie in this second criterion, the last job among the tied ones in the list of jobs sorted by $P_{j_{50\%}}$ is selected. The GBF matheuristic is expressed as a pseudocode in Algorithm 3.

Name: $GBF(g)$
Solve UPMR-S or UPMR-P for jobs $1, ..., g$;
Let d^* be a solution;
Let $\bar{j}$ be the job yielding the makespan;
(in UPMR-S) Fix x_{ijk} according to d^* $\forall$ $j \in \{1, ..., g\} \setminus \{\bar{j}\}$;
or
(in UPMR-P) Fix x_{ij} and y_{ij} according to d^* $\forall$ $j \in \{1, ..., g\} \setminus \{\bar{j}\}$;
$h = 1$;
while $hg \leq n$ **do**
  Solve UPMR-S or UPMR-P for jobs $\{hg, ..., (h+1)g - 1\} \cup \{\bar{j}\}$;
  Let j^* be the job yielding the makespan ;
  Let d^* be a solution;
  (in UPMR-S) Fix x_{ijk} according to
    d^* $\forall$ $j \in (\{hg, ..., (h+1)g - 1\} \cup \{\bar{j}\}) \setminus \{j^*\}$;
  or
  (in UPMR-P) Fix x_{ij} and y_{ij} according to
    d^* $\forall$ $j \in (\{hg, ..., (h+1)g - 1\} \cup \{\bar{j}\}) \setminus \{j^*\}$;
  Update $\bar{j} = j^*$ and $h = h + 1$;
end

Algorithm 3: Greedy-based fixing algorithm (GBF).

5. Computational experiments and statistical evaluation

In this section we show the results obtained in an exhaustive computational experiment with instances of small and medium sizes. All these results are comprehensively analyzed through statistical techniques.

The solver used has been IBM ILOG-CPLEX 12.6. (Another alternative state-of-the-art solver, Gurobi 6.0, did not yield, in some limited experiments, better results and was disregarded). All experiments have been carried out

in a computational cluster formed by 30 blade servers. Each server contains two Intel XEON E5420 processors running at 2.5 GHz and 16 GBytes of RAM memory. However, the specific tests are performed on virtual machines running on this cluster. Each virtual machine runs Microsoft Windows 7 64 bit operating system and has one single virtual processor and 2 GBytes of RAM. Therefore, there is no parallel computing, just a random distribution of model-instances to virtual machines in order to speed-up the experiments. The platform used for the codes employed is Microsoft Visual Studio 2010 and the programming language Visual Basic .NET.

5.1. Instance generation

In order to generate the instances, several factors are considered. Apart from the number of jobs n and the number of machines m, the magnitude and dispersion of the processing times is of paramount importance, as shown in [13]. The last aspect to consider is the units available of the resource and the consumption of these by the jobs. Following [13], the processing times are generated in five different ways: three are purely at random, the fourth assumes that the processing times heavily depend on the job, and the fifth assumes that the processing times heavily depend on the machine:

1. $p_{ij} = U(1, 100)$, which is the most commonly used distribution in the scheduling literature about parallel machines, where $U(a, b)$ is a random integer uniformly distributed between a and b (both extremes included).
2. $p_{ij} = U(10, 100)$, very common in recent years in unrelated parallel machine scheduling problems, see [15, 24, 26] .
3. $p_{ij} = U(100, 200)$, introduced by [13] so that the relative difference between the minimum and maximum processing times is leveled out and the result is a more realistic instance.
4. Correlated jobs. Each job is assigned a $p_j = U(1, 100)$, plus an additional time as $p'_{ij} = U(1, 20)$, having $p_{ij} = p_j + p'_{ij}$.
5. Correlated machines. Each machine is assigned a $p_i = U(1, 100)$, and the processing of each job in this machine is increased by $p'_{ij} = U(1, 20)$, having $p_{ij} = p_i + p'_{ij}$.

With these five different intervals that control the processing times of the jobs, we consider the following factors for the instances:

1. The number of jobs n considered is 8, 12, 16, 20, 25 and 30. 8, 12 and 16 are considered small-size, whereas 20, 25 and 30 are deemed to be medium-size.

2. The number of machines m considered is 2, 4, and 6.
3. The number of resources needed by each job at a given machine has been randomly generated in two different ways:
   (a) $r_{ij} = U(1, 9)$.
   (b) By intervals. In this case, the number of resources r_{ij} needed by each machine-job pair increases with the processing times, following the next formula:

$$r_{ij} = \left\lfloor \frac{p_{ij} - U_{\min}}{d} \right\rfloor + R_{lo} + U(-3, 3),$$

   where $d = \frac{U_{\max} - U_{\min}}{R_{up} - R_{lo} + 1}$, $U_{\max}$ and $U_{\min}$ are the theoretical maximum and minimum processing times, and R_{up} and R_{lo} are the theoretical maximum and minimum number of resources needed by a job-machine assignment. Note that this expression is truncated to lie in the interval [1,9] if necessary.

The $R_{\max}$ is computed so that all instances are feasible. Since the number of resources required varies between $r_{\min} = 1$ and $r_{\max} = 9$, we compute the number of maximum available resources as:

$$R_{\max} = m\frac{r_{\max} + r_{\min}}{2} = m\frac{9 + 1}{2} = 5m,$$

where m is the number of machines available.

To summarize, four factors are considered when designing the instances, with the following levels: $p_{ij} \in \{U(1, 100), U(10, 100), U(100, 200), CorrJob, CorrMach\}$, $n \in \{8, 12, 16, 20, 25, 30\}$, $m \in \{2, 4, 6\}$, $r_{ij} \in \{U(1, 9), Intervals\}$. We replicate all possible combinations of these four factors five times. Therefore, the total number of instances to be tested is $(5 \times 6 \times 3 \times 2) \times 5 = 900$ (450 small, 450 medium).

5.2. $K_{\max}$ in the experiments

For the MILP program UPMR-S and the matheuristic JMR the maximum value of the time index k allowed was set to a trivial upper bound equal to the makespan value assuming only one machine is available, that is, $K_{\max} = \min_i \sum_j p_{ij}$. Although not reported here due to space limitations, we tried other much tighter bounds with mixed results, as CPLEX had difficulty finding feasible solutions.

In the MAF method, the $K_{\max}$ used to solve the UPMR-S model was set equal to the sum of the processing times of the job-machine assignments fixed by the solution of the corresponding UPM, that is, $K_{\max} = \sum_{i,j} p_{ij} x^*_{ij}$, where x^* is an optimal solution to the UPM problem, as the $K_{\max}$ firstly defined might be infeasible when the job-machine assignment is fixed.

There are also some specific details for the $K_{\max}$ used in the GBF method when paired with the UPMR-S model. Since at each iteration only a fraction of the jobs are considered, the $K_{\max}$ is updated accordingly. For example, in the first iteration, the UPMR-S model is solved with the $K_{\max}$ calculated only with the first g jobs involved. For subsequent iterations, the time index (k) is defined only from the makespan value of the previous jobs to the calculated $K_{\max}$ considering also the $g-1$ new jobs to be assigned. By doing so, the UPMR-S model to be solved at each iteration of the GBF method is much smaller, having a significantly reduced number of variables, which results in a much faster execution.

5.3. Experimental settings

All proposed algorithms are run in the aforementioned virtual machines with a maximum stopping time of 3600 seconds. This is the maximum allowed time for the UPMR-S and UPMR-P mathematical models. Some additional details are needed for the other tested matheuristics.

The Machine-assignment fixing (MAF) procedure first solves the UPM model and then uses either the UPMR-S or UPMR-P models with fewer variables, as explained before. The CPU time in solving the UPM model is ignored as it is, on average, negligible. Therefore, the MAF procedure itself is run for one hour.

For the job-machine reduction (JMR), the reduced model after fixing ℓ is also run for one hour as a maximum stopping time. After some calibration, the value given to ℓ in the experiments is set to $\lfloor m/2 \rfloor + 1$, for each instance, i.e., for the instances with 6 machines, the variables corresponding to the fastest 4 machines for each job are defined. For instances with 4 machines, 3 are selected. Note that for 2 machines, both machines are selected so in all instances with 2 machines, JMR is no different from the regular models for this value of ℓ.

For the Greedy-based fixing (GBF) method the parameter g is set to 8 after some calibrations. Notice that 8 is the same size as the smallest instances. Therefore, for these instances, GBF will obtain the same results as the mathematical models. The details of the calibration are not given due

n	$C_{\max}(x^*)$	LB(UPMR-P)	LB(UPMR-S)
8	88	147	137
12	109	52	2
16	139	9	56
20	149	2	21
25	150	0	6
30	150	0	1

Table 1: Absolute frequencies of best lower bound found.

to reasons of space, but suffice to say, simple analyses of variance experiments were carried out on some calibration instances (different from the final test instances). The maximum CPU time given for each iteration is 3600 seconds divided by the number of iterations in the GBF algorithm.

After all these explanations, we now test the two mathematical models proposed, UPMR-S and UPMR-P, and the three matheuristic methods, each one tested with the two MILP models. Therefore, we test MAF-S, MAF-P, JMR-S, JMR-P, GBF-S and GBF-P. All methods are tested over the 900 aforementioned instances, which results in $8 \times 900 = 7200$ results. The total CPU time needed to do all the experiments was 3550 hours or equivalently, almost 148 days.

5.4. Response variables measured

The response variable is the relative percentage deviation (RPD) over a calculated lower bound. The lower bound considered (LB) is the maximum among the optimal solution to the problem without resources UPM, (denoted by $C_{\max}(x^*)$), the MIP lower bound when solving UPMR-S (denoted by LB(UPMR-S)), and the MIP lower bound when solving UPMR-P (denoted by , LB(UPMR-P). Therefore,

$$LB = \max\{C_{\max}(x^*), LB(\text{UPMR-S}), LB(\text{UPMR-P})\}.$$

The number of instances in which each of the three lower bounds considered was the best (i.e., the maximum) is summarized in Table 1, for each value of n. One can see that, specially when the number of jobs considered increases, the lower bound obtained by solving the instance without considering the additional resource (denoted by $C_{\max}(x^*)$) is the best one.

All UPM models for the small and medium instances are quickly solved with CPLEX in a few seconds. As a matter of fact, the average CPU time used by CPLEX for solving the UPM models among the 900 instances is just 0.24 seconds, 8.92 seconds being the maximum recorded for one of the largest instances in the medium set with 30 jobs and 6 machines. This response variable for any of the tested instances is therefore measured as:

$$RPD = 100 \cdot \frac{Method_{sol} - LB}{LB},$$

where $Method_{sol}$ is the solution obtained by the algorithm tested. It should be noted that LB is a hypothetical lower bound, and therefore we will sometimes be comparing our algorithms with values that cannot be reached.

The first important result to be mentioned after the experimentation is that not all models and matheuristics are capable of obtaining feasible solutions in all cases, even for the small instances of 12 jobs. This is in stark contrast with the results of the regular UPM model which, as mentioned, needs less than 9 seconds in the worst case to obtain an optimal solution. A first salient conclusion is that adding resources to the unrelated parallel machine problems results in a problem that is significantly more complex to solve. This was a somewhat expected result as the UPM model is basically an assignment problem where the processing order of the jobs at the machines is not important when minimizing the $C_{\max}$. In the UPMR problem however, the sequence of the jobs on each machine is important, as it also represents the start and finish times in order to satisfy the resource availability constraint. Table 2 shows, for each tested method, and grouped by the number of jobs n, a summary of the results. For the two MIP models tested, (MIP Gap = 0) is the number of instances solved to optimality within the time limit. For all methods, column (Opt.) shows the number of instances in which the solution returned is optimal. This fact is checked by comparing this solution with the guaranteed optimal solution provided by one of the MIP models, or by comparing it with the solution to the UPM problem. Note that, for the MIP models, a solution that satisfies MIP gap = 0 is optimal, but the contrary is not always true. (Feas.) is the number of instances in which a feasible but non-optimal result is given (MIP gap strictly positive after the time limit for the mathematical programming models), and (No Sol.) is the number of instances in which no solution was returned. For the MIP models, the sum of the columns (MIP Gap = 0), (Feas) and (No sol.) is 150. For the matheuristic algorithms, the sum of (Opt.), (Feas) and (No sol.) is 150. It

should be highlighted that there are 150 instances for each value of n.

As we can see, the UPMR problem is hard to solve optimally. Only the smallest instances can be solved to optimality and not in all cases. The model UPMR-S obtains more optimal solutions than model UPMR-P but at the same time it also has a large number of no solutions. The model UPMR-P always gives a solution, which is a much desired trait. As regards the matheuristics, the only ones that are not always able to find solutions are MAF-S and JMR-S, both based on UPMR-S. Of particular interest is MAF-P, which is able to find solutions that are later proved to be optimal in almost 50% of the cases. In this scenario it is hard to compare methods when not all of them have given solutions for all instances as we risk comparing apples with oranges. Therefore, in the following experiments we carry out two measurements both based on the RPD. In the first case we remove all cases in which one or more methods failed to give a solution from the comparison, i.e., if for an instance all methods except one provided a solution, this instance is removed from the comparison. This measurement is denoted as RPD_1. In the second case we consider all instances, but substitute all results in which there is no solution, for the $K_{\max}$ value. This is to say that, when a method is not able to give a solution, we assume this solution has a makespan equal to the $K_{\max}$. This measurement is denoted as RPD_2. We now describe the values of these two measurements both for the set of small instances and for the set of medium instances.

5.5. Small instances results

The Average Relative Percentage Deviation ($\overline{RPD}$) calculated for both measurements and grouped by the different values of n for the small instances (8, 12, 16), is given in Table 3. We also show the number instances considered (Count), the maximum RPD (Max RPD) and the average CPU time in seconds (Av. Time).

In column Count, within group RPD_1, the reader may note that in 20 of the 450 small instances, at least one of the methods did not find a solution in the allowed computation time which affects the RPD_1 measurement. However, the distribution of these *unsolved* instances depends on the n values: for $n = 8$ all methods found a solution, for $n = 12$ there is one instance (which represents 0.67%) in which at least one method could not find a feasible solution, and this number increases to 19 (which represents 12.67%) when $n = 16$. As we can see, both mathematical models are very capable in relation to the smallest instances of 8 jobs with UPMR-P having a slight advantage.

	UPMR-S				UPMR-P			
n	MIP Gap = 0	Opt.	Feas.	No Sol.	MIP Gap = 0	Opt.	Feas.	No Sol.
8	137	147	13	0	146	150	4	0
12	84	97	65	1	47	85	103	0
16	53	65	78	19	1	26	149	0
20	21	45	108	21	0	7	150	0
25	6	18	114	30	0	1	150	0
30	1	5	106	43	0	0	150	0
Total	302	377	484	114	194	269	706	0

	MAF-S			MAF-P		
n	Opt.	Feas.	No Sol.	Opt.	Feas.	No Sol.
8	75	75	0	75	75	0
12	72	78	0	72	78	0
16	64	85	1	71	79	0
20	64	85	1	67	83	0
25	48	99	3	60	90	0
30	50	98	2	76	74	0
Total	373	520	7	421	479	0

	JMR-S			JMR-P		
n	Opt.	Feas.	No Sol.	Opt.	Feas.	No Sol.
8	123	27	0	126	24	0
12	81	69	0	86	64	0
16	51	93	6	31	119	0
20	37	93	20	13	137	0
25	17	103	30	2	148	0
30	9	108	33	0	150	0
Total	318	493	89	258	642	0

	GBF-S			GBF-P		
n	Opt.	Feas.	No Sol.	Opt.	Feas.	No Sol.
8	147	3	0	150	0	0
12	36	114	0	55	95	0
16	8	142	0	18	132	0
20	3	147	0	9	141	0
25	0	150	0	5	145	0
30	0	150	0	3	147	0
Total	194	706	0	240	660	0

Table 2: Study about the solutions given by the methods proposed for all instances.

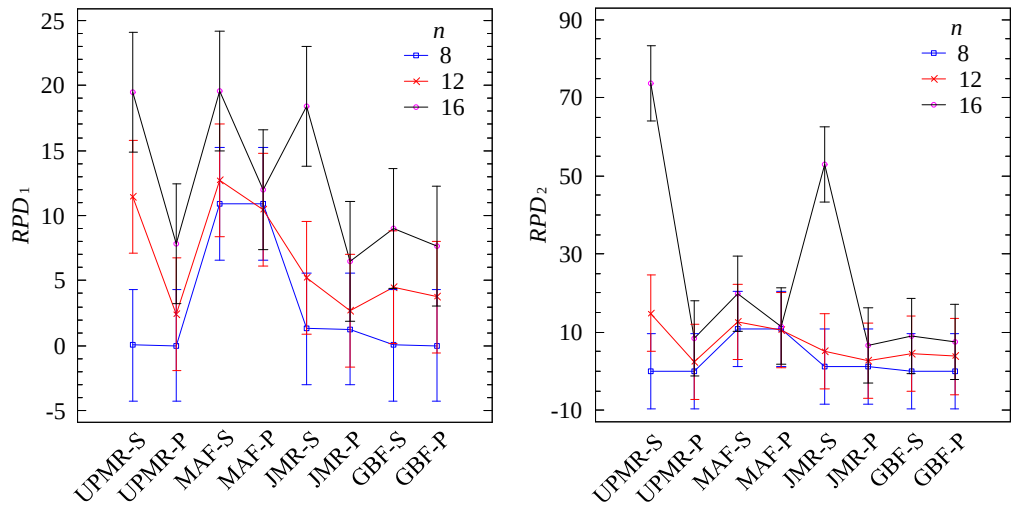
		RPD_1				RPD_2			
Method	n	Count	$\overline{RPD_1}$	Max RPD_1	Av. Time	Count	$\overline{RPD_2}$	Max RPD_2	Av. Time
UPMR-S	8	150	0.02	1.99	482.74	150	0.02	1.99	482.74
	12	149	11.43	526.24	1907.38	150	14.86	526.24	1918.66
	16	131	19.52	332.79	2443.81	150	73.70	628.31	2590.26
		430	9.92	526.24	1573.84	450	29.53	628.31	1663.89
UPMR-P	8	150	**0.00**	**0.00**	254.11	150	**0.00**	**0.00**	254.11
	12	149	**2.40**	**31.93**	2770.52	150	**2.38**	**31.93**	2776.05
	16	131	7.82	**39.18**	3594.08	150	8.28	**64.66**	3594.85
		430	3.21	39.18	2143.60	450	3.55	64.66	2208.34
MAF-S	8	150	10.92	89.77	238.88	150	10.92	89.77	238.88
	12	149	12.71	98.60	812.98	150	12.62	98.60	807.84
	16	131	19.53	99.27	1650.78	150	19.86	247.88	1621.26
		430	14.16	99.27	867.95	450	14.47	247.88	889.33
MAF-P	8	150	10.92	89.77	**1.07**	150	10.92	89.77	**1.07**
	12	149	10.48	71.56	**268.16**	150	10.41	71.56	**266.38**
	16	131	12.02	69.42	1299.44	150	11.54	80.86	1303.49
		430	11.10	89.77	489.17	450	10.95	89.77	523.65
JMR-S	8	150	1.30	31.13	484.96	150	1.30	31.13	484.96
	12	149	5.23	121.81	1656.52	150	5.20	121.81	1655.83
	16	131	18.36	299.62	2399.24	150	52.85	591.08	2554.71
		430	7.86	299.62	1474.11	450	19.78	591.08	1565.17
JMR-P	8	150	1.27	31.13	117.08	150	1.27	31.13	117.08
	12	149	2.71	**31.93**	2250.08	150	2.69	**31.93**	2245.54
	16	131	**6.45**	**39.18**	3539.47	150	**6.60**	**64.66**	3547.15
		430	3.35	39.18	1898.82	450	3.52	64.66	1969.92
GBF-S	8	150	0.02	1.99	482.74	150	0.02	1.99	482.74
	12	149	4.48	32.15	884.00	150	4.46	32.15	894.24
	16	131	9.05	44.83	666.65	150	8.98	65.03	880.92
		430	4.32	44.83	677.81	450	4.49	65.03	752.63
GBF-P	8	150	**0.00**	**0.00**	254.11	150	**0.00**	**0.00**	254.11
	12	149	3.74	**31.93**	480.56	150	3.74	**31.93**	478.58
	16	131	7.67	**39.18**	**614.72**	150	7.54	64.91	**756.67**
		430	3.63	39.18	442.44	450	3.76	64.91	496.45

Table 3: Average Relative Percentage Deviation ($\overline{RPD}$, two measurements) for the tested methods in the small instances. Time in seconds and best values highlighted in bold.

However, the performance for $n = 12$ and $n = 16$ is much worse and the average deviations quickly escalate to two digits. For the second measurement, RPD_2, UPMR-S deviates 73.70% from the lower bounds when $n = 16$, with too large maximum RPD deviations. The reformulation carried out in the UPMR-P model really shows its worth. First of all, both measurements yield similar results for this model and the average and maximum values of RPD_2 are one order of magnitude lower than those of UPMR-S (3.55% and 64.67% versus 29.53% and 628.31%, respectively). The CPU times are, however, larger than those of UPMR-S, especially for $n = 16$. This is due to the fact that, UPMR-S closes the gap more often than UPMR-P, and therefore the average CPU time is lower. However, we want to highlight that UPMR-P finds feasible solutions more often than UPMR-S. As regards the matheuristic methods, JMR-P, GBF-P and GBF-S are competitive with UPMR-P. Actually, for the larger instances of this set, $n = 16$, JMR-P is better, having CPU times comparable to those of UPMR-P. Therefore, this first analysis shows that some of the proposed matheuristics are competitive or better than their corresponding mathematical models in our set of small instances. The best CPU times correspond to MAF-P and GBF-P.

Although not shown here due to reasons of space (the interested reader can download all detailed results and tables from the on-line materials accompanying this paper), there are some interesting results if we group by other instance factors. For example, model UPMR-S performs very badly in instances where processing times follow the distribution $U(100, 200)$. The explanation is that the processing times calculated this way are on average larger than when calculated using the other methods proposed, which means many more variables due to the time-index k in the variables. However, model UPMR-P is not affected by the distributions since the amount of variables of this model does not depend on this time index.

While tables with average results are useful for analyzing and comparing the performance of algorithms, they are insufficient for ascertaining if the observed differences in the averages are statistically significant. We therefore analyze the results with the Analysis of Variance (ANOVA) technique, a simple but powerful parametric statistical tool. All instance characteristics (n, m, distribution of processing times –referred to as type of instance–, distribution of resource requirements –res–) are considered as non-controllable blocking factors in a full factorial experiment. Additionally, the type of algorithm is another factor with 8 levels (all tested algorithms). The response variables are the two RPD measurements. Being parametric, there are three



(a) RPD_1 measure. (b) RPD_2 measure.

Figure 4: Interactions between the proposed methods and number of jobs n. All means with Tukey's Honest Significant Difference (HSD) 95% confidence intervals. Small instances.

hypotheses that must be checked. These are normality, homoscedasticity and independence of the residuals. These were all checked and met. The complete results of the ANOVA are not given due to space considerations, but some multiple pairwise comparison plots are shown. Figure 4 shows the means plots for the methods proposed and the number of jobs n for both measures. The intervals have the observed average RPD in the center and are calculated according to Tukey's Honest Significant Difference (HSD) method, with 95% confidence level. Overlapping intervals between any two means imply not statistically significant differences.

As can be seen in Figure 4, the proposed mathematical models are only statistically better than some proposed matheuristics for the smallest instances of 8 jobs. For 12 jobs most matheuristics can be considered statistically equivalent. UPMR-P is shown to be statistically better than UPMR-S for the larger instances as well.

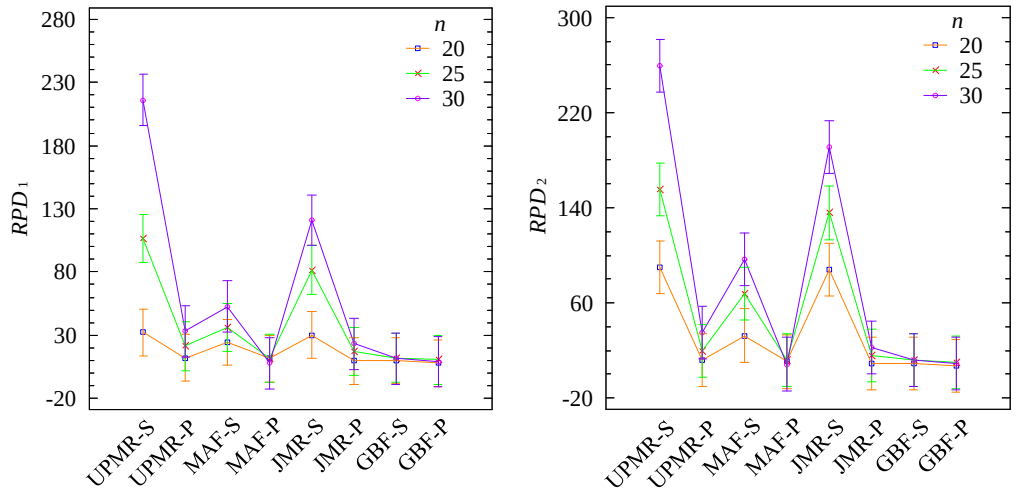
5.6. Medium instances results

The results for the medium instances ($n = \{20, 25, 30\}$) are reported in Table 4.

Note that for the RPD_1 measurement, 100 out of the 450 instances tested

		RPD_1				RPD_2			
Method	n	Count	$\overline{RPD}$	Max RPD	Av. Time	Count	$\overline{RPD}$	Max RPD	Av. Time
UPMR-S	20	128	32.10	293.20	3197.14	150	89.87	592.86	3256.25
	25	117	106.49	553.07	3530.17	150	155.08	586.78	3545.56
	30	105	216.01	1508.86	3610.25	150	259.38	1508.86	3607.19
		350	112.14	1508.86	3432.40	450	168.11	1508.86	3469.67
UPMR-P	20	128	11.92	48.54	3600.21	150	11.86	48.54	3600.28
	25	117	21.20	72.88	3600.45	150	19.74	72.88	3600.47
	30	105	33.10	117.75	3600.85	150	34.92	129.74	3600.88
		350	21.38	117.75	3600.48	450	22.17	129.74	3600.54
MAF-S	20	128	24.14	146.20	2123.61	150	32.60	461.45	2164.94
	25	117	35.88	292.00	2429.80	150	67.68	450.82	2583.92
	30	105	52.45	435.46	2151.21	150	96.34	486.15	2513.37
		350	36.56	435.46	2234.24	450	65.54	486.15	2420.74
MAF-P	20	128	11.30	72.92	2528.08	150	10.47	72.92	2401.75
	25	117	11.88	63.26	2850.54	150	11.79	69.58	2922.33
	30	105	**7.90**	57.70	2916.03	150	**8.46**	57.70	2901.68
		350	10.47	72.92	2752.26	450	10.24	72.92	2741.92
JMR-S	20	128	30.07	312.77	3212.05	150	88.08	592.86	3270.72
	25	117	81.64	514.34	3525.09	150	135.69	586.78	3541.59
	30	105	120.86	637.85	3584.31	150	191.31	640.83	3591.93
		350	74.55	637.85	3428.37	450	138.36	640.83	3468.08
JMR-P	20	128	9.54	38.38	3600.08	150	9.21	38.38	3600.08
	25	117	17.16	60.71	3600.20	150	15.95	60.71	3600.19
	30	105	23.15	52.07	3595.52	150	22.35	67.21	3597.06
		350	16.17	60.71	3598.75	450	15.84	67.21	3599.11
GBF-S	20	128	9.80	39.89	**735.06**	150	9.24	39.89	**860.77**
	25	117	12.01	59.33	**629.38**	150	11.97	59.33	**1144.47**
	30	105	11.52	**37.28**	**717.40**	150	12.20	79.22	**1210.75**
		350	11.05	59.33	694.44	450	11.14	79.22	1072.00
GBF-P	20	128	**7.83**	**38.18**	788.76	150	**7.18**	**38.18**	926.57
	25	117	**10.39**	**58.55**	1074.18	150	**10.17**	**58.55**	1263.48
	30	105	9.07	39.05	1020.02	150	9.13	**39.05**	1391.58
		350	9.06	58.55	953.55	450	8.82	58.55	1193.87

Table 4: Average Relative Percentage Deviation ($\overline{RPD}$, two measurements) for the tested methods in the medium instances. Time in seconds and best values highlighted in bold.



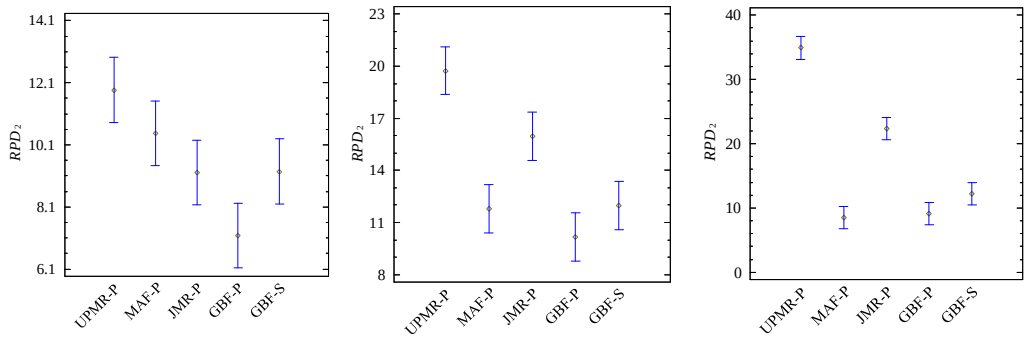
(a) RPD_1 measure.

(b) RPD_2 measure.

Figure 5: Interactions between the proposed methods and number of jobs n. All means with Tukey's Honest Significant Difference (HSD) 95% confidence intervals. Medium instances.

are not considered, as one or more methods did not provide a solution. This explains the large variation in results for the medium instances when both measurements are compared. The first and probably most important result is that for the medium instances the mathematical models, in particular UPMR-S, are no longer competitive and the average deviations go from 32.10% to 259.38%, depending on the measurement. UPMR-P performs better than UPMR-S, yielding better averages and lower max RPD while using comparable CPU times. For larger n values, most proposed matheuristics outperform UPMR-P. For example, MAF-P gives average relative percentage deviations equal to 7.90% and 8.46% for $n = 30$, which are several times lower than those of UPMR-P while using considerably less CPU time. We also show the graphical results of the ANOVA in Figure 5.

As we can see, most of the larger observed differences between any two means are statistically significant, especially for the RPD_2 measure. As a matter of fact, measure RPD_1 favours the methods that produce fewer solutions more than RPD_2, as the former does not consider them and the latter measurement assigns them a "bad" value. For the medium instances, model UPMR-P is by a large margin statistically better than model UPMR-S. Moreover, the matheuristics based on the UPMR-S model perform badly,



(a) $n = 20$. (b) $n = 25$. (c) $n = 30$.

Figure 6: RPD_2 measure for the best algorithms as a function of the number of jobs n. All means with Tukey's Honest Significant Difference (HSD) 95% confidence intervals. Medium instances.

except for GBF-S. The large differences in the plotted averages of Figure 5 do not allow us to observe the small differences between the high performing methods. Figure 6 shows, for the three values of n in the medium instances, the performance of the best methods (removing UPMR-S, JMR-S and MAF-S) for measure RPD_2.

From Figure 6 we observe that, as the number of jobs increases, the observed differences among the best methods get larger. For $n = 25$ the best methods are MAF-P, GBF-P or GBF-S and the difference from UPMR-P is clear. For $n = 30$ there is a large margin of difference. Since GBF-P has good results in small and medium instances with small CPU times, it seems the best option among the matheuristics. In general, all these results show that the modelization of the UPMR as a special packing problem results in powerful methods and that the matheuristics presented clearly outperform the state-of-the-art solver CPLEX when solving the packing model.

6. Conclusions

This paper studies a realistic unrelated parallel machine problem with the additional consideration of resources that are used when jobs are being processed on the machines. We have presented an adaptation of an existing formulation (named UPMR-S) as well as a novel reformulation of the problem inspired by the strip packing model (named UPMR-P), which, as opposed to UPMR-S, does not need a time index. In the experiments, we have shown

how the UPMR-P outperforms model UPMR-S significantly as the size of the solved problems increases. However, and as expected, because the UPMR problem has previously been identified as $\mathcal{NP}$-Hard, neither of these MIP models is able to cope with medium-sized instances. For these cases, we have presented matheuristic strategies and have shown them to be competitive with the MIP models in small instances. In medium-sized instances, the matheuristics clearly outperform the MIP models. Furthermore, each of the three matheuristic algorithms obtained from model UPMR-P are superior to their corresponding counterparts obtained from model UPMR-S which is another indication that the strip-packing-based model is better than the other one.

Initial tests over larger sized instances (around 50 jobs and 15 machines), have shown us that more research needs to be done, as the matheuristics proposed do not always succeed in providing a good solution. Therefore, the complexity of this problem, exponentially increasing with the size of the instance, calls for the design of further heuristics and metaheuristics which shall be addressed in further research.

Acknowledgments

The authors are supported by the Spanish Ministry of Economy and Competitiveness, under project "SCHEYARD - Optimization of Scheduling Problems in Container Yards" (No. DPI2015-65895-R), partially financed with FEDER funds. Thanks are due to our colleagues Eva Vallada and Ful Villa, for their useful suggestions. Special thanks are due to three anonymous referees which have significantly contributed to the improvement of the manuscript. Apart from accompanying on-line materials, interested readers can download more contents from `http://soa.iti.es/problem-instances`, like the instances used, software for generating instances and all the binaries of the algorithms tested in this paper. We also provide complete solutions, full tables of results and the statistics software files to replicate all results and plots. Additional explanations are also provided in "how-to" text files.

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

[2] Billaut, J.-C., Croce, F. D., Grosso, A., 2015. A single machine scheduling

problem with two-dimensional vector packing constraints. European Journal of Operational Research 243 (1), 75–81.

[3] Błażewicz, J., Ecker, K. H., Pesch, E., Schmidt, G., Weglarz, J., 2007. Handbook on Scheduling. Springer-Verlag, New York, USA.

[4] Błażewicz, J., Kubiak, W., Röck, H., Szwarcfiter, J., 1987. Minimizing Mean Flow-Time with Parallel Processors and Resource Constraints. Acta Informatica 24 (5), 513–524.

[5] Błażewicz, J., Lenstra, J. K., Rinnooy Kan, A. H. G., 1983. Scheduling subject to resource constraints: classification and complexity. Discrete Applied Mathematics 5 (1), 11–24.

[6] Croce, F. D., Salassa, F., T'kindt, V., 2014. A hybrid heuristic approach for single machine scheduling with release times. Computers & Operations Research 45, 7–11.

[7] Daniels, R. L., Hua, S. Y., Webster, S., 1999. Heuristics for parallel-machine flexible-resource scheduling problems with unspecified job assignment. Computers & Operations Research 26 (2), 143–155.

[8] Edis, E. B., Oguz, C., 2011. Parallel Machine Scheduling with Additional Resources: A Lagrangian-Based Constraint Programming Approach. Vol. 6697. Springer-Verlag Berling Heidelberg, Ch. Lecture Notes in Computer Science, CPAIOR 2011, pp. 92–98.

[9] Edis, E. B., Oguz, C., 2012. Parallel machine scheduling with flexible resources. Computers and Industrial Engineering 63 (2), 433–447.

[10] Edis, E. B., Oguz, C., Ozkarahan, I., 2013. Parallel machine scheduling with additional resources: Notation, classification, models and solution methods. European Journal of Operational Research 230 (3), 449–463.

[11] Edis, E. B., Ozkarahan, I., 2011. A combined integer/constraint programming approach to a resource-constrained parallel machine scheduling problem with machine eligibility restrictions. Engineering Optimization 43 (2), 135–157.

[12] Edis, E. B., Ozkarahan, I., 2012. Solution approaches for a real-llife resource constrained parallel machine scheduling problem. International Journal of Advanced Manufacturing Technology 9 (12), 1141–1153.

[13] Fanjul-Peyro, L., Ruiz, R., 2010. Iterated greedy local search methods for unrelated parallel machine scheduling. European Journal of Operational Research 207 (1), 55–69.

[14] Fanjul-Peyro, L., Ruiz, R., 2011. Size-reduction heuristics for the unre-

lated parallel machines scheduling problem. Computers & Operations Research 38 (1), 301–309.

[15] Ghirardi, M., Potts, C. N., 2005. Makespan minimization for scheduling unrelated parallel machines: A recovering beam search approach. European Journal of Operational Research 165 (2), 457–467.

[16] Grigoriev, A., Sviridenko, M., Uetz, M., 2005. Unrelated parallel machine scheduling with resource dependent processing. In: Jünger, M., Kaibel, V. (Eds.), TIMES, Proceedings of the 11th conference on integer programming and combinatorial optimization. Vol. 3509 of Lecture Notes in Computer Science. Springer, Berlin-Heidelberg, pp. 182–195.

[17] Grigoriev, A., Sviridenko, M., Uetz, M., 2007. Machine scheduling with resource dependent processing times. Mathematical Programming Series B 110 (1), 209–228.

[18] Kellerer, H., 2008. An approximation algorithm for identical parallel machine scheduling with resource dependent processing times. Operations Research Letters 36 (2), 157–159.

[19] Kellerer, H., Strusevich, V. A., 2003. Scheduling parallel dedicated machines under a single non-shared resource. European Journal of Operational Research 147 (2), 345–364.

[20] Kellerer, H., Strusevich, V. A., 2003. Scheduling problems for parallel dedicated machines under multiple resource constraints. Discrete Applied Mathematics 133 (1–3), 45–68.

[21] Kravchenko, S., Werner, F., 2011. Parallel machine problems with equal processing times: A survey. Journal of Scheduling 14 (5), 435–444.

[22] Lenstra, J. K., Rinnooy Kan, A. H. G., and Brucker, P., 1977. Complexity of machine scheduling problems. Annals of Discrete Mathematics 1, 343–362.

[23] Lodi, A., Martello, S., Monaci, M., 2002. Two-dimensional packing problems: A survey. European Journal of Operational Research 141 (2), 241–252.

[24] Martello, S., Soumis, F., Toth, P., 1997. Exact and approximation algorithms for makespan minimization on unrelated parallel machines. Discrete Applied Mathematics 75 (2), 169–188.

[25] McNaughton, R., 1959. Scheduling with deadlines and loss functions. Management Science 6 (1), 1–12.

[26] Mokotoff, E., Jimeno, J. L., 2002. Heuristics based on partial enumera-

tion for the unrelated parallel processor scheduling problem. Annals of Operations Research 117 (1-4), 133–150.

[27] Perea, F., de Waard, H. W., 2011. Greedy and k-greedy algorithms for multidimensional data association. IEEE Transactions on Aerospace and Electronic Systems 47 (3), 1915–1925.

[28] Pinedo, M. L., 2005. Planning and scheduling in manufacturing and services. Springer series in operations research. Springer, New York, USA.

[29] Pinedo, M. L., 2016. Scheduling: theory, algorithms and systems, 5th Edition. Springer, New York,USA.

[30] Słowiński, R., 1980. Two approaches to problems of resource allocation among project activities, a comparative study. Journal of the Operational Research Society 31 (8), 711–723.

[31] Stefanello, F., de Araújo, O. C. B., Müller, F. M., 2015. Matheuristics for the capacitated p-median problem. International Transactions in Operational Research 22 (1), 149–167.

[32] Sule, D. R., 2008. Production planning and industrial scheduling: examples, case studies and applications, 2nd Edition. CRC Press, Boca Raton, USA.

[33] Toffolo, T. A. M., Santos, H. G., Carvalho, M. A. M., Soares, J. A., 2015. An integer programming approach to the multimode resource-constrained multiproject scheduling problem. Journal of Scheduling (Online).

[34] Ventura, J. A., Daecheol, K., 2000. Parallel machine scheduling about an unrestricted due date and additional resource constraints. IIE Transactions 32 (2), 147–153.