A bin packing reformulation and matheuristics for the unrelated parallel machines scheduling problem with resources

Rubén Ruiz[1], Luis Fanjul[1], and Federico Perea[1]

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.
rruiz@eio.upv.es, lfpeyro@gmail.com, perea@eio.upv.es

Abstract. This short paper studies a parallel machines scheduling problem with the consideration of resources. Each job needs a number of resources at each machine for its completion. The objective studied 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, whereas the other exploits the resemblance of our problem with the well-known strip packing problems. Since none of these models is capable of solving to optimality medium-sized instances, we propose three matheuristic strategies. All algorithms proposed are run over randomly generated instances of small-medium size. Results show that matheuristics outperform the mathematical models by a large margin that widens with instance size.

Keywords: scheduling, unrelated parallel machines, bin packing, additional resources, matheuristics, makespan

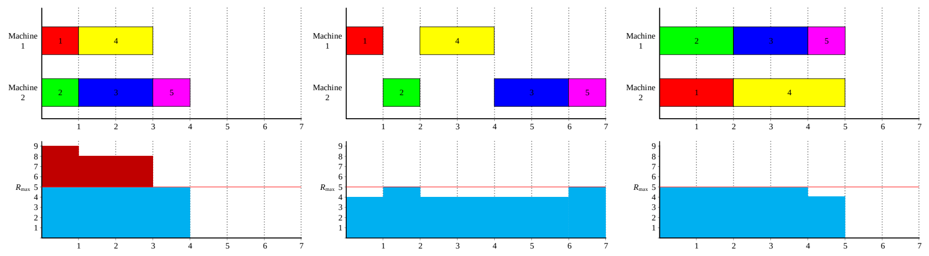
1 Introduction and motivation

Production tasks are largely carried out by machines. Therefore, the need for intelligent organization becomes a must. In this paper we study a generalization of the classical *unrelated parallel machines scheduling problem* (UPM) in which we have to process a number of jobs in a number of parallel machines that are available in the production shop. The most common objective in this problem is to find an assignment of jobs to machines, so that the latest job being processed finishes as soon as possible, the minimization of the so-called *makespan*. Machines can process jobs simultaneously in *parallel*, and processing times need not be the same for all *unrelated* machines. The UPM arises in production systems in which two or more tasks need to be processed. [8] is one the first papers dealing with this topic. Ever since then, the interest of the scientific community on the UPM has not stopped increasing. Most of the treated scheduling problems on parallel machines do not consider that the machines need an extra resource for their functioning. Such extra resource could be, for example, the human operators needed to manipulate machines. In this paper we assume that a fixed number of operators are needed to process jobs in machines. More specifically, 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. We refer to the resulting problem as the *Unspecified Dynamic Unrelated Parallel Machine Scheduling problem with additional Resources* (UPMR). The UPMR problem takes the following input data: 1) A list of m available machines, indexed by i and i'. 2) A list of n jobs to be processed, indexed by j and j'. 3) $R_{\max}$ units of a certain resource. 4) $p_{ij} \in \mathbb{Z}^+$ units of time and $r_{ij} \in \mathbb{Z}^+$ units of the resource, are needed to process job j at machine i, $\forall\ i = 1, ..., m, j = 1, ..., n$. The additional constraint to satisfy is that no more than $R_{\max}$ units of the resource can be used at any time. The aim is to assign jobs to machines and to decide when they will be processed, so that the maximum completion time of the job is minimized. This objective is commonly referred to as makespan or $C_{\max}$. The following example illustrates our problem (UPMR) and the differences with respect to the unrelated parallel machine scheduling problem (UPM).

Example 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 violated by four, between time 0 and time 1, and by three units between time 1 and time 3. Constructing a resource-feasible solution from the optimum solution without resources, 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 optimum solution considering resources for the same problem we obtain the solution given in Figure 1c.



(a) Opt. sol. (b) Sol. with resources. (c) Opt. sol. with resources.

Fig. 1: Optimum solution for the example in different scenarios.

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 than the solution without resources.

Note that the regular parallel machines problem is just an assignment problem, and the only decision to be made is to which machine each job must be assigned to. Assigned jobs are processed in any order and the machines are never idle in between jobs. The version with resources is much more complex as the assignment, the sequence of the assigned jobs at each machine, and also the starting and completion times, must be determined. Besides, at times machines might be unable to process the next job due to resource shortage and idle times might appear.

Although not as old as the version without additional resources, the UPMR (or close variations of it) has been studied in the literature for the last three decades. Back in the eighties, [1] study a UPMR problem in which all machines are identical, and therefore requirements of additional resources and processing times only depends on the job. There are some papers about the static version of the problem, which assumes that the allocation of resources to machines is given and fixed during the whole time horizon. [2] study another simplified version of the UPMR, in which the number of resources needed to process a job does not depend on the machine in which the job is to be processed. [4] propose some models for the UPMR problem with machine eligibility constraints (not all jobs can be processed in all machines). These models are later applied to a real instance by the same authors in [5].

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

2 MILP modelling

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 bin packing problem.

2.1 A MILP based on previous research

In this section we adapt the MILP program introduced by [3] to the UPMR studied in this paper. The model assumes that the number of resources assigned affects the processing times. This is different from our problem, which assumes that a fixed amount of resources is needed for processing jobs in machines, and this number may not be changed. Besides 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 at which a job can be processed, denoted by $K_{\max}$, is not trivial. Further discussions about $K_{\max}$ are given in the experiments 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 and $C_{\max}$ is the makespan. Note that the x_{ijk} variable only exists for $k \geq p_{ij}$. Model UPMR-S consists of minimizing $C_{\max}$, subject to the following constraints:

$$\sum_{k,i} k x_{ijk} \leq C_{\max},\ \forall\ j \tag{1}$$

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

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

$$\sum_{j,i,s:s\in\{k,\ldots,k+p_{ij}-1\}} r_{ij} x_{ijs} \leq R_{\max}, \forall\ k \geq p_{ij} \tag{4}$$

Constraints (1) determine the makespan. Constraints (2) impose 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) impose that no more than $R_{\max}$ units of resource are used at any time.

2.2 A bin packing problem based model

We use the ideas obtained from bin-packing formulations to model our UPMR as a linear programming program. In 2D bin-packing problems, the objective is to place a set of rectangular items into a rectangular case. When one of the dimensions of the case is fixed, the problem is known as a strip packing problem where the objective is to place a set of rectangles into the case so that the length of the other dimension is minimized. We will consider that the height of the case (denoted by H, which will denote the resources) is fixed, and the width (denoted by W, which will denote the makespan) is to be minimized, so that all items fit in the case.

The resemblance between this problem and the specified version of the UMPR, in which the assignment of jobs to machines is fixed a priori, can be stated as follows. 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 and height 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_k and y_k (the x-axis will represent time and the y-axis will represent the resources). In UPMR, these variables will be x_{ij} and y_{ij}. 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. Note that this is a very specific strip-packing problem for which no specific notation, like that provided in [9] is valid. The specified UPMR problem can be modeled as a strip-packing problem as follows:

$$\text{minimize} \quad C_{\max} \tag{5}$$

$$p_{ij} \leq x_{ij} \leq C_{\max}; \quad r_{ij} \leq y_{ij} \leq R_{\max}; \qquad \forall\ j \tag{6}$$

$$\begin{array}{c} x_{ij} - x_{i'j'} \geq p_{ij} \ \ \mathit{or} \ \ x_{i'j'} - x_{ij} \geq p_{i'j'} \\ \mathit{or} \ \ y_{ij} - y_{i'j'} \geq r_{ij} \ \ \mathit{or} \ \ y_{i'j'} - y_{ij} \geq r_{i'j'} \end{array} \quad \forall\ j, j' > j+1 \tag{7}$$

Equation (5) is the objective: the minimization of the free dimension of the strip (which corresponds to the makespan). Constraints (6) set the bounds for x_{ij} and y_{ij}. Constraints (7) prevent the overlap of any two rectangles, by forcing the difference between each pair of rectangles to be larger than or equal to the width (horizontal axis) or height (vertical axis), of the rectangle located more to the right and higher, respectively. Besides, we also need to impose to this model that, when jobs are assigned to the same machine ($i = i'$), there should be no overlap between the rectangles associated to these jobs in the horizontal axis, because the same machine cannot process more than one job at the same time. In the UPMR problem jobs are not pre-assigned to any machine and we need to define a set of m rectangles for each job: one for each machine. Because jobs are processed in one machine, for each job we must select only one rectangle out of this set, and schedule them in order to minimize the $C_{\max}$. For this purpose, we need to make $x_{ij} = y_{ij} = 0$ in case job j is not assigned to machine i, and the overlap-constraints (7) only need to be applied to pairs of rectangles that have been selected out of its respective set. Therefore, for the UPMR problem model based on bin-packing, denoted by UPMR-P, we need the binary variable s_{ij}. This variable takes value 1 when job j is assigned to machine i, and zero otherwise. The generic UPMR-P model consists of minimizing the makespan subject to:

$$\sum_{i=1}^{m} s_{ij} = 1 \quad \forall\ j \tag{8}$$

$$\begin{array}{r} \text{if } \ s_{ij} = 1 \Rightarrow \ \ p_{ij} \leq x_{ij} \leq C_{\max}; \quad r_{ij} \leq y_{ij} \leq R_{\max}; \\ \text{if } \ s_{ij} = 0 \Rightarrow \ \ x_{ij} = y_{ij} = 0; \end{array} \quad \forall\ i, j \tag{9}$$

$$\begin{array}{r} \forall\ i, j, i', j' \geq j+1 \ \text{ such that } \ \ s_{ij} = 1 \ \text{ and } \ s_{i'j'} = 1 \Rightarrow \\ x_{ij} - x_{i'j'} \geq p_{ij} \ \text{ or } \ x_{i'j'} - x_{ij} \geq p_{i'j'} \\ \text{or } \ y_{ij} - y_{i'j'} \geq r_{ij} \ \text{ or } \ y_{i'j'} - y_{ij} \geq r_{i'j'} \quad \text{(vertical axis only if } \ i \neq i') \end{array} \tag{10}$$

Constraints (8) impose that each job j is processed by one machine. Constraints (9) impose the bounds for x_{ij} and y_{ij}, and fix these variables to zero if job j is not assigned to machine i. Constraints (10) prevent the overlap of rectangles ij and $i'j'$, and are only imposed if both rectangles are selected. They also impose that, in case $i = i'$ (same machine), there should be no overlap in the horizontal axis.

Our next step is to write the previous model as an integer linear programming model. For this aim, we need to write the bound constraints (9) and the overlap constraints (10) as linear constraints. Out of several possibilities tested, the one that showed the best performance is now described.

$$\min \; C_{\max} + \frac{1}{M}\sum_{j=1}^{n}\sum_{i=1}^{m} x_{ij} \tag{11}$$

$$\text{s.t.:} \qquad \sum_{i=1}^{m} s_{ij} = 1 \;\forall\; j, \tag{12}$$

$$s_{ij}p_{ij} \le x_{ij} \le C_{\max}, \quad s_{ij}r_{ij} \le y_{ij} \le R_{\max}s_{ij} \quad \forall\; i,j, \tag{13}$$

$$Mu_{iji'j'} + x_{ij} - x_{i'j'} \ge p_{ij}, \quad Mu_{i'j'ij} + x_{i'j'} - x_{ij} \ge p_{i'j'} \tag{14}$$

$$u_{i'j'ij} + u_{iji'j'} \le 1 + (1 - s_{ij}) + (1 - s_{i'j'}), \quad (\text{if} \;\; i' = i) \tag{15}$$

$$Mv_{iji'j'} + y_{ij} - y_{i'j'} \ge r_{ij}, \quad Mv_{i'j'ij} + y_{i'j'} - y_{ij} \ge r_{i'j'}, \quad (\text{if} \;\; i' \ne i) \tag{16}$$

$$\begin{aligned} u_{iji'j'} + u_{i'j'ij} + v_{iji'j'} + v_{i'j'ij} \le 3 + (1 - s_{ij}) + (1 - s_{i'j'}) + \\ (1 - s_{ij}) + (1 - s_{i'j'}), \quad (\text{if} \;\; i' \ne i) \end{aligned} \tag{17}$$

Where the binary variables u and v are used to determine which of the no-overlap constraints are activated. Constraints (14), (15), (16) and (17) being defined for all $i, j, i', j' \ge j + 1$.

The objective function (11) minimizes $C_{\max}$ and forces x_{ij} to zero whenever this is possible (if job j is not assigned to machine i). (12) make that each job j is assigned only to one machine. Constraints (13) set the bounds for the horizontal and vertical axis, setting this bound for the vertical axes to zero if s_{ij} is zero. Constraints (14) and (16) are the no overlap constraints, which are imposed for all pairs in the horizontal axis, and only for pairs that do not share the same machine in the vertical axis ($i \ne i'$). Constraints (15) and (17) are the boundary equations for the no overlap restrictions. The first set applies to pairs sharing the same machine, and the second set applies to pairs not sharing the same machine. Notice that these constraints are relaxed when s_{ij} or $s_{i'j'}$ are zero.

3 Matheuristic strategies

Matheuristics are algorithms in which metaheuristics techniques are combined with mathematical programming models. Although a relatively new concept, it is gaining more and more attention in the last years. We briefly describe three heuristic strategies based on the MILP models introduced in Section 2. The first one is based on exploiting the specified version of the problem, in which the job to machine assignment is fixed in a first step as the solution to the relaxed UPM problem, to then solve the UPMR having such 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 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 it yielded in the UPM, see [7]. The third one is a greedy strategy, in which the problem is sequentially solved for smaller subsets of jobs, keeping fixed the assignments found in previous iterations. This strategy was chosen both for its simplicity and for the extensive use, many times successful, that greedy algorithms and its variants have had in the literature.

3.1 Machine-assignment fixing

This algorithm firstly solves the regular unrelated parallel machines problem minimizing the makespan (UPM problem without the resource constraints), which can be done very rapidly in a solver as shown in [6]. As shown in the experiments section, 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. In a second phase, the MILP program of the UPMR is relaxed in such a way that the variables that assume any other job-assignment than the one obtained by UPM are fixed to zero. This method is referred to as MAF.

3.2 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 here mean 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 model. For example, for each job j we set variables to zero for all machines i such 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 refereed to as job-machine reduction (JMR).

3.3 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 have a scheduled start time. This strategy closely resembles the K-greedy algorithms. 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 in the machine generating the makespan and $g-1$ new unscheduled jobs are added to the problem and 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. There might be a final iteration with the $\big((n-g) \bmod (g-1)\big)+1$ final unscheduled jobs.

4 Computational and statistical evaluation

In order to generate the instances, several factors are considered. Apart from number of jobs n and number of machines m, the magnitude and dispersion of the processing times and the units available of the resource and the consumption of these by the jobs are considered. The processing times are generated in five different ways (see [6] for more details). We consider the following additional 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 as 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: $r_{ij} = U(1,9)$ and by intervals, increasing with the processing times. We divide the set of job-machine feasible assignments into different groups, according to their processing times. We assign a number of resources to every group, increasingly depending on the processing times. Then the number of resources for each job-machine pair is computed as the sum of the number assigned before and a $U(-3,3)$, truncated between 1 and 9, if necessary. As a result, if $U_{\max}$ and $U_{\min}$ are the theoretical maximum and minimum processing times and if R_{up} and R_{lo} are the theoretical maximum and minimum number of resources needed by a job-machine assignment, then $r_{ij} = \left\lfloor \frac{p_{ij}-U_{\min}}{d} \right\rfloor + R_{lo} + U(-3,3)$, where $d = \frac{U_{\max}-U_{\min}}{R_{up}-R_{lo}+1}$ and $\lfloor x \rfloor$ means we take the integer part of x (rounded down). 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). For the MILP program UPMR-S, the maximum k allowed was set to a trivial upper bound equal to the makespan value assuming only one machine is available, that is, $K_{\max} = \min_i \sum_j p_{ij}$. Note that for the proposed matheuristics the values of the $K_{\max}$ were set accordingly.

The solver used has been IBM ILOG-CPLEX 12.6. All experiments have been carried out in a computational cluster formed by 30 blade servers each with two Intel XEON E5420 processors running at 2.5 GHz and 16 Gbytes of RAM memory. Virtual machines of 1 processor and 2 GBytes of RAM are used. All proposed algorithms are run in the aforementioned computers with a maximum stopping time of 1 hour each. We test the two proposed mathematical models, 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. In total we have two MILP models and 6 matheuristics. 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 (if the experiments were performed in a single computer).

The response variable is the relative percentage deviation (RPD) over a calculated lower bound. The lower bound contains three components, the first one is the optimum solution of the problem without resources, i.e., the UPM problem, obtained by solving the simple and well known assignment problem shown, among others, in [6]. All UPM models for the small and medium instances are quickly solved with CPLEX in a few seconds. The other two components are taken from the lower bounds given by the solver after solving the two proposed models with resources, i.e., UPMR-S and UPMR-P. This response variable RPD 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 any of the tested models or matheuristic methods for a given instance and LB is the three component lower bound.

The first important result 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. In this scenario it is hard to compare methods when not all of them have given solutions for all instances. Therefore, in the following experiments we carry out two measurements, both based on the RPD. In the first case we remove from the comparison all cases in which one or more methods failed to give a solution, or where the solution had an absurdly high makespan value (RPD_1). In the second case we consider all instances, but substitute all results in which there is no solution, or there is a bad solution, by the $K_{\max}$ value (RPD_2). The Average Relative Percentage Deviation ($\overline{RPD}$) calculated for both measurements is given in Table 1. We are also showing the number of cases at each row (Counts), the maximum $\overline{RPD}$ (Max RPD) and the average CPU time in seconds (Av. Time).

For the small instances, in 20 of the 450 instances not all methods could find a solution. Both mathematical models are very capable. For RPD_2, UPMR-S deviates almost a 30% from the lower bounds. The reformulation carried out in the UPMR-P model is much better with maximum values of RPD_2 being one order of magnitude lower than those of UPMR-S. As regards the matheuristic methods, only JMR-P and GBF-P are competitive with UPMR-P. The best CPU times correspond to GBF-P. This is a very interesting result as we have a very fast method with good performance. For the medium instances ($n = \{20, 25, 30\}$) we have that, for the RPD_1 measurement, 100 out of 450 instances do not have solution for all methods. The most important result is that the mathematical models, in particular UPMR-S, are no longer competitive and the average deviations go up to more than 160% depending on the measurement. UPMR-P is definitely much better than UPMR-S in all measurements while using comparable CPU times. Most proposed matheuristics dominate UPMR-P,

Small instances

	RPD_1				RPD_2			
Method	Count	$\overline{RPD}$	Max RPD	Av. Time	Count	$\overline{RPD}$	Max RPD	Av. Time
UPMR-S	430	9.92	526.24	1573.84	450	29.53	628.31	1663.89
UPMR-P	430	**3.21**	**39.18**	2143.60	450	3.55	**64.66**	2208.34
MAF-S	430	14.16	99.27	867.95	450	14.47	247.88	889.33
MAF-P	430	11.10	89.77	489.17	450	10.95	89.77	523.65
JMR-S	430	7.86	299.62	1474.11	450	19.78	591.08	1565.17
JMR-P	430	3.35	**39.18**	1898.82	450	**3.52**	**64.66**	1969.92
GBF-S	430	4.32	44.83	677.81	450	4.49	65.03	752.63
GBF-P	430	3.63	**39.18**	**442.44**	450	3.76	64.91	**496.45**

Medium instances

	RPD_1				RPD_2			
Method	Count	$\overline{RPD}$	Max RPD	Av. Time	Count	$\overline{RPD}$	Max RPD	Av. Time
UPMR-S	350	112.14	1508.86	3432.40	450	160.11	1508.86	3469.67
UPMR-P	350	21.38	117.75	3600.48	450	22.17	129.74	3600.54
MAF-S	350	36.56	435.46	2234.24	450	65.54	486.15	2420.74
MAF-P	350	10.47	72.92	2752.26	450	10.24	72.92	2741.92
JMR-S	350	74.55	637.85	3428.37	450	138.36	640.83	3468.08
JMR-P	350	16.17	60.71	3598.75	450	15.84	67.21	3599.11
GBF-S	350	11.05	59.33	**694.44**	450	11.14	79.22	**1072.00**
GBF-P	350	**9.06**	**58.55**	953.55	450	**8.82**	**58.55**	1193.87

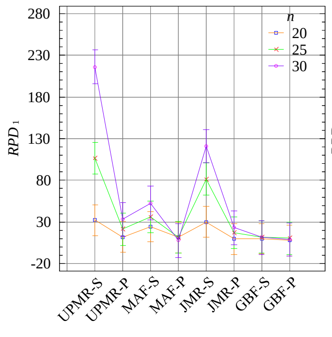
Table 1: Average Relative Percentage Deviation ($\overline{RPD}$, two measurements) for the tested methods. Time in seconds.

i.e., for example MAF-P gives average relative percentage deviations under both measurements that are more than three times lower than those of UPMR-P while using considerable less CPU times.

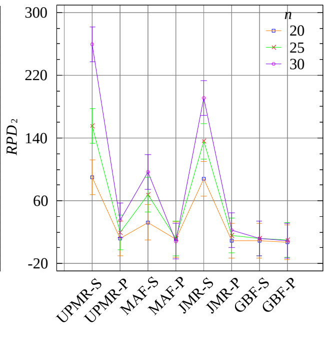
All results are checked for statistical significance with the ANOVA technique. The complete results of the ANOVA are not given due to space considerations (and are omitted for the small instances). Figure 2 shows the means plots for the interaction between the proposed methods and the number of jobs n for both measures in the medium instances. The intervals have the observed average RPD in the center and are calculated according to Tukey's Honest Significant Difference (HSD) method. The intervals have a 95% confidence level. Overlapping intervals between any two means imply statistical insignificance in the observed differences. As we can see, most of the larger observed differences between any two means are statistically significant, specially for the RPD_2 measure. For the medium instances, model UPMR-P is statistically better, and by a wide margin, than model UPMR-S. Also, most matheuristics based on the UPMR-S model also perform bad, with the exception of GBF-S.

5 Conclusions

We have shown how the formulation of the UPMR problem using a strip-packing problem based model outperforms an adaptation of a model previously introduced in the literature. However, and as expected because the UPMR problem has previously been termed NP-hard, neither of these ILP models are able to cope with



(a) RPD_1 measure.



(b) RPD_2 measure.

Fig. 2: Interactions between the methods and n. All means with Tukey's Honest Significant Difference (HSD) 95% confidence intervals. Medium instances.

medium-sized instances. The matheuristic strategies proposed are competitive with the ILP models in small instances, and clearly outperform them in medium-sized instances. Besides, each of the three matheuristic algorithms obtained from model UPMR-P is superior to their corresponding counterparts obtained from model UPMR-S. The complexity of this problem, exponentially increasing with the size of the instance, calls for the design of heuristics and metaheuristics, which shall be addressed in further research.

References

1. Błażewicz, J., Kubiak, W., Röck, H., Szwarcfiter, J.: Minimizing mean flow-time with parallel processors and resource constraints. Acta Informatica 24, 513–524 (1987)
2. Edis, E.B., Oguz, C.: Parallel Machine Sccheduling with Additional Resources: A Lagrangian-Based Constraint Programming Approach. Lecture Notes in Computer Science 6697, 92–98 (2011)
3. Edis, E.B., Oguz, C.: Parallel machine scheduling with flexible resources. Computers and Industrial Engineering 63, 433–447 (2012)
4. Edis, E.B., Ozkarahan, I.: A combined integer/constraint programming approach to a resource-constrained parallel machine scheduling problem with machine eligibility restrictions. Engineering Optimization 43(2), 135–157 (2011)
5. Edis, E.B., Ozkarahan, I.: Solution approaches for a real-llife resource constrained parallel machine scheduling problem. International Journal of Advanced Manufacturing Technology 9(12), 1141–1153 (2012)
6. Fanjul-Peyro, L., Ruiz, R.: Iterated greedy local search methods for unrelated parallel machine scheduling. European Journal of Operational Research 207(1), 55–69 (2010)
7. Fanjul-Peyro, L., Ruiz, R.: Size-reduction heuristics for the unrelated parallel machines scheduling problem. Computers & Operations Research 38(1), 301–309 (2011)
8. McNaughton, R.: Scheduling with deadlines and loss functions. Management Science 6(1), 1–12 (1959)
9. Wäscher, G., Haußner, H., Schumann, H.: An improved typology of cutting and packing problems. European Journal of Operational Research 183(3), 1109–1130 (2007)