Heuristic algorithms for the unrelated parallel machine scheduling problem with one scarce additional resource

Fulgencia Villa, Eva Vallada, Luis Fanjul-Peyro

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: mfuvilju@eio.upv.es, evallada@eio.upv.es, lfpeyro@hotmail.com

Abstract

In this paper, we study the unrelated parallel machine scheduling problem with one scarce additional resource to minimise the maximum completion time of the jobs or makespan. Several heuristics are proposed following two strategies: the first one is based on the consideration of the resource constraint during the whole solution construction process. The second one starts from several assignment rules without considering the resource constraint, and repairs the non feasible assignments in order to obtain a feasible solution. Several computation experiments are carried out over an extensive benchmark. A comparative evaluation against previously proposed mathematical models and matheuristics (combination of mathematical models and heuristics) is carried out. From the results, we can conclude that our methods outperform the existing ones, and the second strategy performs better, especially for large instances.

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

1. Introduction and problem definition

In the unrelated parallel machine scheduling problem (UPM), a set of n jobs has to be processed on exactly one machine out of a set of m machines. In this variant of the problem, processing times of the jobs differ according to the machine the job is assigned to. This problem has been extensively studied in the literature over recent decades (Fanjul-Peyro and Ruiz (2010), Fanjul-Peyro and Ruiz (2011), Vallada and Ruiz (2011), Rodriguez et al. (2013) and

Arroyo and Leung (2017), among others). However, machines should not be considered as the only resource in a real manufacturing environment. In this case, a job needs, in addition to a machine, an amount of one or more additional resources. These additional resources (human resources, tools, etc.) are limited, so it is necessary to consider them when the jobs are assigned to the machines. Moreover, the amount of resources a job needs differs according to the machine it is assigned to. This extension to a more realistic problem is needed in a competitive world, where the objective of the companies is to increase their profit. Therefore, this approach to the problem is the unrelated parallel machine scheduling problem with additional resources (UPMR), which has been the focus of far fewer studies in the research community. Regarding the optimisation objective, the most studied in the scheduling literature is the minimisation of the maximum completion of the jobs, known as makespan ($C_{\max}$). In the UPM problem, there is only an assignment problem, since the sequence inside each machine does not affect the makespan. However, in the UPMR the problem consists of finding the best assignment of n jobs to m machines, and at the same time, the best sequence for each machine so that at any time the resource constraint is satisfied. In this paper, several heuristic methods are proposed for the UPMR problem with one renewable additional resource with the objective of minimising the makespan. The additional resource is considered as: renewable, since after a job is processed the resource is again available; discrete, the amount of resource needed by a job is a positive integer; processing, the resource is needed just during the processing of the job. Both, processing times and resource consumption of the jobs are different depending on the machine the job is assigned to. This consideration is especially important in real manufacturing environments since different machines are used at the same time. The processing time or resource consumption of a job is expected to be different on an old machine from that on a new one.

In a formal way, the problem consists of scheduling a set of n jobs (indexed by j) on one machine of a set of m machines (indexed by i). Moreover, one additional resource is considered. The following assumptions are considered: each job is processed by exactly one machine, each machine can process one job at a time, preemption of jobs is not allowed and each job needs an amount of the additional resource during its entire process. A solution or sequence for this problem consists of a list of jobs for each machine. The order of the jobs on the list represents the processing order on the machine. Note that a solution or sequence implies the computation of the starting and finishing

times of each job on the machines. Depending on the resource availability, idle times might be necessary in order to obtain a feasible solution. The following notation is introduced:

- p_{ij}: processing time of job j on machine i.
- r_{ij}: resource consumption of job j on machine i.
- $Mach_i$: list of jobs assigned to machine i and processed following the order of the list.
- $J_{i[l]}$: job in position l of machine i.
- $R_{\max}$: maximum availability of resource.
- R_t: remaining units of resource at time t.
- RD_0: total resource demand at time 0.
- $r_{i[0]}$: resource consumption of the job in position 0 on machine i.
- C_i: completion time of machine i.
- $C_{\max}$: maximum completion time, $max\{C_1, C_2, \cdots, C_i\}$.
- $i_{\max}$: machine with the maximum completion time or makespan ($C_{\max}$).

The main contribution of this paper is that the proposed heuristics can efficiently solve instances of different sizes, especially large ones, something that, at present, does not exist in the literature. The described problem is solved, in this paper, by two different approaches. The first one is based on ordering the jobs to construct a solution taking into account the resource constraint during the process. The second approach is based on assigning jobs to machines without considering the resource constraint and applying a repair mechanism in case the assignment is not feasible with respect the resource.

The rest of the paper is organised as follows: In Section 2, an overview of the literature is presented. Sections 3 and 4 are focused on the proposed heuristic methods. In Section 5 computational results are provided. Finally, in Section 6 some conclusions and future research are given.

2. Literature review

The unrelated parallel machine scheduling problem, or very similar variations, has been widely considered over recent decades. Some recent results can be found in Zeidi and MohammadHosseini (2015) and Chen (2015). A review of parallel machine scheduling problems can be found in Kravchenko and Werner (2011). According to Lenstra et al. (1977), this problem is NP-hard even for the simplest version with two identical parallel machines. The

consideration of additional resources has been studied significantly less. One of the first works is Garey and Johnson (1975), where the authors examine the complexity of identical parallel machine scheduling problems with additional resources. The conclusion was that the problem is NP-complete even with only one additional resource. A classification of the complexity of scheduling problems subject to resource constraints can be found in Błażewicz et al. (1983). In Błażewicz et al. (1987), the identical parallel machine scheduling problem with one additional resource is studied with the objective of minimising the total flow time. The authors showed that the problem is NP-hard in the strong sense, even with only 2 machines and one type of resource. Also for the identical machines case, Ventura and Daecheol (2000) studied one additional resource so that jobs need one or zero units with an objective related to due dates. A more recent work with the same one/zero assumption is Zheng and Wang (2016). Several works can be found for dynamic versions of the problem where the processing times of the jobs depend on the number of resources assigned (Grigoriev et al. (2005), Grigoriev et al. (2007), Kellerer (2008), Yin et al. (2014), Hsu and Yang (2014) and Hsieh et al. (2015)). The additional resource constraint is an important consideration from a real industrial environment point of view. Some works related to manufacturing systems can be found in Edis and Ozkarahan (2012), Bitar et al. (2016) and Ventura and D. (2003). More recently, in Edis and Ozkarahan (2011) and Edis and Oguz (2012), mathematical models for different variations of the problem are proposed. In Edis et al. (2013) a classification and a summary of solution methods for parallel machine scheduling problems with additional resources are provided. However, most of the papers study a simplification of the problem: a fixed amount of resource for each machine during the whole production horizon (Daniels et al. (1999), Ruiz-Torres et al. (2007)), resource consumption of a job independent of the machine (Bitar et al. (2016)) or the resource consumption of a job is one or zero units (Ventura and D. (2003), Zheng and Wang (2016)). As a result, heuristic methods for the general problem where both processing times and resource consumption of the jobs are different according to the machine to which jobs are assigned, have not been proposed yet in the literature. A recent study, where an adaptation of the model presented in Edis and Oguz (2012) is given, together with other different MILP mathematical models and matheuristics methods for the more general problem studied in this paper, can be found in Fanjul-Peyro et al. (2017). A matheuristic is a combination of a mathematical model with a heuristic, some variables of the model are fixed according to the solution

provided by a heuristic method. Therefore, a matheuristic can not guarantee the optimal solution. Matheuristics have been gaining more attention the last years. Some examples related to scheduling problems are Croce et al. (2014) and Billaut et al. (2015). Therefore, the objective of this paper is to propose efficient (in terms of CPU time) and effective (in terms of quality of the solution) heuristic methods for the general problem, so that they are able to solve much larger instances than the methods presented in Fanjul-Peyro et al. (2017), where only small and medium instances are considered.

3. Heuristics based on resources

This first approach consists of three phases: job ordering, construction of the solution and improvement of the solution. During the last two phases the resource constraint is considered, that is, total resource consumption at any time must be lower than or equal to $R_{\max}$.

3.1. Ordering of the jobs

In this phase the jobs are ordered without considering resource constraint according to one of these rules:

- Based on processing time (Rule P): for each job, the lowest k values ($k = 1, 2, 3$) of its processing times (p_{ij}) are added up and jobs are ordered in non increasing order according to this sum. A different version where jobs are ordered in non decreasing order according to the sum is also proposed.
- Based on resource consumption (Rule R): similar to Rule P, but using r_{ij} instead of p_{ij}.
- Based on machines (Rule M): on each machine, jobs are ordered in non decreasing order of p_{ij}. For each machine a group with the k first jobs is considered ($k = 1, 2, 3$). A hypothetical or virtual sequence is constructed and the group of k jobs with the hypothetical minimum completion time is included in the ordered list of jobs. Previous hypothetical completion times are considered during the whole process when a new group of k jobs are considered. A different version using r_{ij} instead of p_{ij} is also proposed. More details are given later following an example.

Regarding ties, several mechanisms are proposed in order to break them. In Figure 1, a summary of the rules and how the ties are broken is shown. Therefore, for Rules P and R, on a first level, two order options are considered. On a second level, two mechanisms for breaking ties and on the third level, two ways for the final order. That is 8 variants for each rule. Rule M is very similar, two order options are considered (non decreasing order of p_{ij} and non decreasing order of r_{ij}). The second and third level are the same as Rules P and R. Therefore, there are 8 variants for Rule M as well.

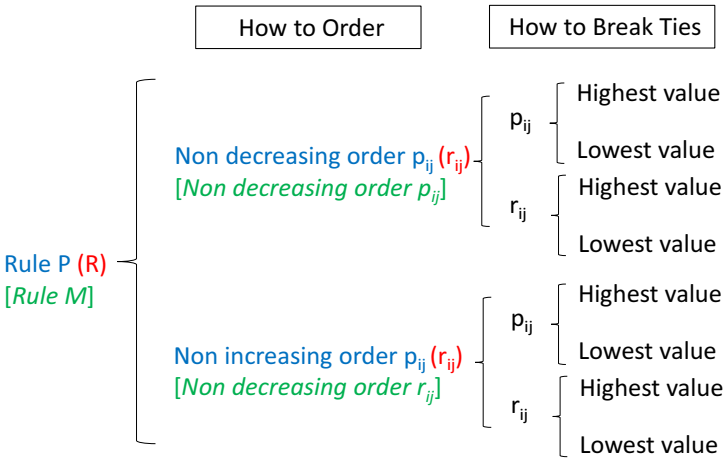

Figure 1: Outline for Rules P, R and M.

The following example illustrates a small instance.

Example 3.1. *Consider the following instance of a UPMR with two machines ($m = 2$), six jobs ($n = 6$), ten units of a scarce resource ($R_{\max} = 10$) and the following processing times and resource needs:*

$$(p_{ij}) = \begin{pmatrix} 1 & 2 & 4 & 3 & 3 & 1 \\ 3 & 1 & 4 & 4 & 1 & 2 \end{pmatrix}; \quad (r_{ij}) = \begin{pmatrix} 2 & 6 & 4 & 7 & 1 & 5 \\ 8 & 6 & 3 & 3 & 2 & 1 \end{pmatrix}$$

If we apply Rule P (non increasing order, $k = 2$), the result can be found in Table 1. In the last row, the addition of the first k ($k = 2$) values for each job are computed. The final sorted list of the jobs is provided by ordering in non increasing order of this addition, that is, $\{J3, J4, J1, J5, J2, J6\}$. There are two ties to be broken: between jobs $J1$ and $J5$ and between jobs $J2$ and $J6$. In this case, following Figure 1, ties are broken according to highest value

of r_{ij}. In more detail, between jobs $J1$ and $J5$ the highest r_{ij} is computed as $max\{r_{11}, r_{21}, r_{15}, r_{25}\} = r_{21} = 8$. Then, $J1$ is ordered before $J5$. The same procedure is applied between $J2$ and $J6$.

In the same Table 1, results for Rule R (non increasing order, $k = 2$) can be found. Ties are broken in the same way as the previous example and the final sorted list is $\{J2, J1, J4, J3, J6, J5\}$.

Regarding Rule M (non decreasing order of p_{ij}, $k = 1$), the final sorted list is obtained according to a hypothetical sequence for the machines. Then, a hypothetical sequence of jobs for each machine is computed. Following the same Example 3.1, the first step is to order jobs on each machine in non decreasing order of p_{ij}, then machine $Mach_1 = \{J1, J6, J2, J4, J5, J3\}$ and machine $Mach_2 = \{J2, J5, J6, J1, J3, J4\}$. The first job of each machine is considered ($k = 1$), in this case $J1$ and $J2$. A hypothetical sequence is computed and the job with the minimum completion time is included in the final sorted list. Specifically, if $J1$ is assigned to machine 1, then $C_1 = 1$ and if $J2$ is assigned to machine 2, then $C_2 = 1$. In this case ties are broken according to the maximum value of p_{ij}, that is, $max\{p_{11}, p_{21}, p_{12}, p_{22}\} = p_{21} = 3$. Then $J1$ is selected to be included on the final sorted list. Therefore, the new order list for each machine is $Mach_1 = \{J6, J2, J4, J5, J3\}$ and machine $Mach_2 = \{J2, J5, J6, J3, J4\}$ and $J6$ and $J2$ are the next jobs to be considered. Note that to compute the hypothetical sequence, jobs already sequenced ($J1$ on machine 1 in this case) are considered in order to obtain the completion times of the machines. Then, $C_1 = p_{11} + p_{16} = 1 + 1 = 2$ and $C_2 = p_{22} = 1$. In this case, there is no tie, then $J2$ is included in the final sorted list. Following the same procedure for all jobs, the final sorted list is $\{J1, J2, J5, J6, J4, J3\}$.

Table 1: Ordering rules example.

	Rule P ($k = 2$)							**Rule** R ($k = 2$)					
	$J1$	$J2$	$J3$	$J4$	$J5$	$J6$		$J1$	$J2$	$J3$	$J4$	$J5$	$J6$
	1	1	4	3	1	1		2	6	3	3	1	1
	3	2	4	4	3	2		8	6	4	7	2	5
Sum	4	3	8	7	4	3	Sum	10	12	7	10	3	6

3.2. Construction of the solution

Once the jobs are ordered, a constructive procedure is applied to obtain a feasible solution considering the resource constraint. Two constructive procedures are proposed, denoted as NEH_{res} and SWA respectively.

- Construction based on NEH (NEH_{res}). This is based on the well known heuristic proposed by Nawaz et al. (1983), considered to be the best one for the flowshop scheduling problem and successfully used for the parallel machine problem (Vallada and Ruiz (2011)). Every job from the ordered list is selected and inserted into all possible positions of each machine. The job is finally located in the position where the makespan is minimum. The resource constraint is considered for every insertion of the jobs. Note that the worst case computational complexity is, as in the original NEH, $\mathcal{O}(n^3m)$. Following with the example 3.1, we consider the final sorted list provided by Rule M ($\{J1, J2, J5, J6, J4, J3\}$). The first job to be considered is job $J1$ has to be inserted in the first position of all the machines, two in this case. The job is finally placed on the machine resulting in the minimum $C_{\max}$, machine 1 in this case. Now, $C_1 = 1$, $C_2 = 0$ and $C_{\max} = 1$. Then, job $J2$ is selected and has to be inserted in all possible positions: before job $J1$ on machine 1, after job $J1$ on machine 1 and on machine 2. Note that in all cases, the resource constraint is satisfied. Finally, job $J2$ is placed on machine 2, since the $C_{\max}$ is minimum in this way. Now, $C_1 = 1$, $C_2 = 1$ and $C_{\max} = 1$. Following this procedure with the remaining jobs, the final solution (list of jobs for each machine representing the processing order on the machine) is $Mach_1 = \{J6, J1, J4\}$ and $Mach_2 = \{J5, J2, J3\}$. The $C_{\max}$ value is $C_2 = 6$.

- Construction with swapping (SWA). In this case jobs are selected from the ordered list and placed in the last position of each machine considering the resource constraint. The job is finally located in the machine with the minimum C_i after the job placement. After the insertion, a swap can be carried out which consists of interchanging the last job inserted with all the last jobs already placed on the remaining machines. In order to decide if the swap movement is finally carried out, two strategies are considered: the movement is accepted if the completion time of both machines is decreased (strict swap) or the movement is accepted if the sum of both completion times after the movement is lower than before (global swap). In addition, the method without swapping movements is also considered (no swap). In this case, the worst case computational complexity is $\mathcal{O}(n^2m)$. Following with the example 3.1, we consider the final sorted list provided by Rule M ($\{J1, J2, J5, J6, J4, J3\}$). The first job to be considered is job $J1$

has to be inserted in the last position of all machines (first position in this case since the machines are empty). The job is finally placed on the machine resulting in the minimum $C_{\max}$, machine 1 in this case. Now, $C_1 = 1$, $C_2 = 0$ and $C_{\max} = 1$. Then, job $J2$ is selected and has to be inserted on the last position of both machines: after job $J1$ on machine 1 and on machine 2. Finally, job $J2$ is placed on machine 2, since the $C_{\max}$ is the minimum. Now, $C_1 = 1$, $C_2 = 1$ and $C_{\max} = 1$. Following this procedure with the remaining jobs, the final solution is $Mach_1 = \{J1, J6, J4\}$ and $Mach_2 = \{J2, J5, J3\}$. The $C_{\max}$ value is $C_2 = 6$. After each insertion, swap movements can be carried out, interchanging the last job of each machine with the last jobs of the remaining machines. In this case, none of the interchanging movements are accepted, since conditions for strict swap or global swap are not satisfied.

3.3. Improvement of the solution

Once a complete and feasible solution is obtained a local search procedure is applied. The improvement method consists of interchanging all the jobs from the machine with the maximum completion time or $C_{\max}$ with all the jobs on the other machines. After testing all combinations, the movement with the best $C_{\max}$ is carried out if it is lower than the original one. Every time a movement is made the procedure starts from the beginning (local search until local optima).

In Algorithm 1 a general description of the algorithm can be found. The local search is not applied in the same way since there are more combinations for the SWA algorithm than the NEH_{res}. Moreover, the SWA construction method includes a small local search. Then, for the SWA algorithm, local search is applied to the best solution obtained at the end of the process, once per each ordering rule. However, for the NEH_{res} method a local search is applied for every combination.

4. Heuristics based on assignment

In this section five multi-pass heuristics are proposed. In this case resource constraint is not considered during the whole process, so it is very likely to obtain non feasible solutions. All the heuristics have a common constructive phase but a different local search (LS). The common constructive phase starts

Algorithm 1: Heuristics based on resources.

```
1  for r = P, R, M do
2      for k = 1, ..., 3 do
3          for tie = 1, ..., 8 do
4              order := Rule(r,k,tie);
5              if SWA is selected as constructor then
6                  for sw = 1, ..., 3(1:=No swap; 2:=strict; 3:=global) do
7                      SWA(sw,order) and Update the best solution;
8              else
9                  NEH_res(order);
10                 Interchange Local Search and Update the best solution
11     if SWA is selected then
12         Interchange Local Search and Update best solution
```

from an assignment (A) provided by a rule and, depending on the local search applied, a different multi-pass heuristic is obtained.

In Algorithm 2 the general pseudocode of the multi-pass heuristics is shown (S denotes a feasible sequence or solution, considering resource constraint). Algorithm 2 is repeated for all assignment rules and the best solution provided by each multi-pass is later compared.

4.1. Construction of the solution

The first step of the constructive phase is the application of the assignment rules, which do not consider resource constraints, so it is very likely to obtain a non feasible solution. The following rules, based on both processing times and resource consumption of the jobs are applied:

- Rule 1: each job j is assigned to the machine i with the minimum p_{ij}. Ties are broken by minimum r_{ij}.
- Rule 2: each job j is assigned to the machine i with the minimum r_{ij}. Ties are broken by minimum p_{ij}.
- Rule 3: each job j is assigned to the machine i with the minimum $p_{ij} \cdot r_{ij}$. Ties are broken by minimum p_{ij}.

Algorithm 2: Multi-pass heuristics based on assignment:general scheme.

```
for a = 1, ..., 8 (For each Assignment Rule) do
    A:=Assignment Rule a;
    Check Feasibility(A);
    if Feasible(A) then
        S := A;
        LS(S) considering resources
    else
        S:=Repairing Mechanism(A)
    LS(S) without considering resources (except for multi-pass 1);
    Update S* := Best Solution
```

- Rule 4: the average consumption value is computed $D = (R_{\max}/m)$. For each job j, the set of machines so that resource consumption of the job is not greater than D is determined and denoted as MD_j. Job is assigned to the minimum processing time machine from MD_j. Ties are broken by minimum r_{ij}. If MD_j is empty, the job is assigned to the minimum r_{ij} machine. Ties are broken by minimum p_{ij}.
- Rule 5: all jobs are tentatively assigned to all machines. On each machine, jobs are ordered in non decreasing order of their processing times. Ties are not initially considered. The job is finally assigned to the machine in which the position of the job is the lowest. Ties for the final assignment are broken by minimum p_{ij}.
- Rule 6: similar to Rule 5, but jobs are ordered in non decreasing order of resource consumption. Ties are broken by minimum r_{ij}.
- Rule 7: similar to Rule 5, but jobs are ordered in non decreasing order of $p_{ij} \cdot r_{ij}$. Ties are broken by minimum r_{ij}.
- Rule 8: a combination of the previous rules based on the algorithm proposed in Alvarez-Valdes et al. (2015) for the parallel machine problem minimising earliness and tardiness. In this case, seven assignments for each machine are available from the previous rules. For each machine the assignment with the minimum completion time is selected. Note that it is very likely to find missing or repeated jobs since a different rule can be selected for each machine. Repeated jobs are assigned to the machine i with the minimum resource consumption. Ties are broken

by minimum C_i. Missing jobs are assigned to the machine i with the minimum C_i.

None of the rules is dominated by the rest, all of them collaborate to the results obtained by the multi-pass heuristics.

Following the Example 3.1, in Table 2 the final assignments using Rules 1 to 4 are shown, as well as completion times (C_i) for each machine.

Rule	Machine 1	C_1	Machine 2	C_2
Rule 1 (min p_{ij})	{J1,J4,J6}	5	{J2,J3,J5}	6
Rule 2 (min r_{ij})	{J1,J5}	4	{J2,J3,J4,J6}	11
Rule 3 (min $p_{ij} \cdot r_{ij}$)	{J1}	1	{J2,J3,J4,J5,J6}	12
Rule 4 ($\leq R_{\max}/m$)	{J1,J6}	2	{J2,J3,J4,J5}	10

Table 2: Assignment of the jobs using Rules 1 to 4.

Rule 5 requires some intermediate steps: all jobs are sorted into non decreasing order of p_{ij} for each machine, that is, $Mach_1 = \{J1, J6, J2, J4, J5, J3\}$ and $Mach_2 = \{J2, J5, J6, J1, J3, J4\}$. Each job is finally placed on the machine where the position of the job is the lowest. For example, Job 1 is assigned to machine 1. Rules 6 and 7 are similar but jobs are sorted using the rules explained above. In Table 3, the final assignments for Rules 5 to 7 are shown.

Rule	Machine 1	C_1	Machine 2	C_2
Rule 5 (non decr. p_{ij})	{J1,J4,J6}	5	{J2,J3,J5}	6
Rule 6 (non decr. r_{ij})	{J1,J2,J5}	6	{J3,J4,J6}	10
Rule 7 (non decr. $p_{ij} \cdot r_{ij}$)	{J1}	1	{J2,J3,J4,J5,J6}	12

Table 3: Assignment of the jobs using Rules 5 to 7.

Rule 8 is a combination of the seven previous ones. In this case, for each machine, the assignment so that the completion time of the machine is the lowest is selected. That is, for machine 1, Rule 3 or Rule 7 assignment is chosen ($C_1 = 1$). Regarding machine 2, assignment from Rule 1 is selected ($C_2 = 6$). Then, $Mach_1 = \{J1\}$ and $Mach_2 = \{J2, J3, J5\}$. The missing jobs ($J4, J6$) are allocated to the machine i with the minimum C_i. So, the final assignment for each machine is $Mach_1 = \{J1, J4, J6\}$ with $C_1 = 5$ and $Mach_2 = \{J2, J3, J5\}$ with $C_2 = 6$.

4.1.1. Check feasibility

Now we check if a given assignment obtained by the previous rules satisfies the resource constraint. Resource Demand is computed at time $t = 0$ (RD_0) after ordering jobs on each machine in non increasing order of r_{ij} (Algorithm 3).

Algorithm 3: Check Feasibility.

```
for i = 1, ..., m do
    Order Mach_i in non increasing order of r_ij;
    Compute RD_0 as the total demand of resources at t = 0;
    if RD_0 ≤ R_max then
        S := A
    else
        S:=Repair Mechanism(A)
```

The same Example 3.1 is used to illustrate the feasibility procedure. Assignment from Rule 2 is selected in order to check the feasibility: $Mach_1 = \{J1, J5\}$, $Mach_2 = \{J2, J3, J4, J6\}$. In this case jobs are already ordered in non increasing order of r_{ij}. Once the jobs are sorted, the resource consumption of every first job of each machine is added. Resource Demand at time $t = 0$ is computed as $RD_0 = r_{11} + r_{22} = 2 + 6 = 8$. Therefore, as $R_{\max} = 10$, $RD_0 < R_{\max}$ and the assignment is feasible. We can state that if the first job (maximum r_{ij}) of every machine can be processed at the same time, then the rest of the sequence is also feasible.

4.1.2. Repairing mechanism

The objective of this procedure is to obtain a feasible solution S, from a given unfeasible assignment A. A new set of jobs denoted as Pending Jobs (PJ) is created. The idea is to move jobs from A to PJ until the partial assignment of A is feasible.

Using Example 3.1, the assignment from Rule 1 is selected in order to check the feasibility: $Mach_1 = \{J1, J4, J6\}$, $Mach_2 = \{J2, J3, J5\}$. In this case, jobs are not ordered in non increasing order of r_{ij}, so the ordered assignment is $Mach_1 = \{J4, J6, J1\}$, $Mach_2 = \{J2, J3, J5\}$. Resource Demand at time $t = 0$ is computed as $RD_0 = r_{11} + r_{22} = 7 + 6 = 13$. Since $RD_0 > R_{\max}$, the assignment is not feasible and the repairing mechanism is applied. The job

placed in first position with the maximum resource consumption is selected to be removed from the assignment. This job is placed in the Pending Jobs set. If there is more than one job with maximum resource consumption, the one placed on the machine with the largest completion time (C_i) is chosen. The process is repeated until $RD_0 \leq R_{\max}$. In our example, the first job selected to be moved from the assignment to the Pending Jobs set is job $J4$. In this case, $r_{14} = 7$ and $r_{22} = 6$, so $J4$ is removed. The partial assignment is $Mach_1 = \{J6, J1\}$, $Mach_2 = \{J2, J3, J5\}$ and $PJ = \{J4\}$. Now, $RD_0 = r_{16} + r_{22} = 5 + 6 = 11$, so the solution is still unfeasible and job $J2$ is selected to be part of the PJ set. The partial assignment is $Mach_1 = \{J6, J1\}$, $Mach_2 = \{J3, J5\}$ and $PJ = \{J4, J2\}$ and it is feasible ($RD_0 = r_{16} + r_{23} = 5 + 3 = 8$). Note that every time a job is moved from the assignment to the PJ set, all jobs are shifted left.

At this point there is, on one hand, a partially feasible assignment and on the other hand a Pending Jobs set to be placed in the sequence satisfying the resource constraint. In Algorithm 4, pseudocode to schedule the Pending Jobs is given. Note that index i^* refers to the machine where the job j was originally assigned. First, we try to schedule the Pending Jobs at the beginning of the machine where they were assigned if there are enough resources. Otherwise, Pending Jobs are assigned at the end of a machine following two strategies: at the end of the machine where the job was originally assigned and at the end of the machine so that the completion time of the job (C_j) is the minimum. In both cases the resource constraint must be satisfied and the solution with the minimum makespan value is finally selected. Once a pending job is scheduled a swap operation is carried out with the previous job, if there are no idle times on the machine and the resource consumption of the previous job is lower than the resource consumption of the pending job scheduled. The swapping operation is carried out as many times as possible, that is, a swap between the scheduled pending job and the previous one is carried out while the sequence after swapping is still feasible.

Algorithm 4: Repair mechanism: Pending Jobs scheduling.

```
for j = 1, ..., |PJ| do
    if R_0 + r_{i*[0]} ≥ r_{i*j} then
        S := S ∪ j;
        Order Mach_i^* in non increasing order of resource consumption;
        PJ := PJ − j;
        Update R_0;
Order PJ in non decreasing order of r_ij;
S' := S;
for j = 1, ..., |PJ| do
    S : = Schedule job j in S at the end of the machine where it was
     originally assigned (earliest time satisfying resource constraint);
    while resource constraint is satisfied do
        Swap the scheduled job with the previous one;
    S' : = Schedule job j in S' at the end of the machine where the
     completion time of the job is the minimum (earliest time satisfying
     resource constraint);
    while resource constraint is satisfied do
        Swap the scheduled job with the previous one;
S* := Select schedule with the lowest C_max(S, S');
```

Following with Example 3.1, $PJ = \{J4, J2\}$. Recall that $Mach_1 = \{J6, J1\}$, and $Mach_2 = \{J3, J5\}$. Algorithm 4 is applied to the current solution. First, we try to schedule job $J4$ at the beginning of machine 1. At time 0, the total consumption is $r_{16} + r_{23}$, that is $5 + 3 = 8$. Then, $R_0 = 10 - 8 = 2$ (available resources at time 0). The first step consists of trying to introduce Pending Jobs at the beginning if there are enough resources. Job $J4$ needs 7 resource units on machine 1 (where it was originally assigned), therefore we have to check if there are enough resources at time 0 to sequence job $J4$ at the beginning of machine 1. If the available units of resource at time 0 (R_0) plus the resource consumption of the first job already scheduled on the machine is equal to or greater than the resource consumption of the pending job, it means that the pending job can be scheduled at the beginning of the machine. In this case, $R_0 = 2$ and the resource consumption of the first job already scheduled on machine 1 is $r_{16} = 5$. The resource consumption of the pending job $J4$ on machine 1 is $r_{14} = 7$. Then, $R_0 + r_{16} = 2 + 5 = 7$

and $r_{14} = 7$. Therefore, pending job $J4$ can be scheduled at the beginning of machine 1. So $Mach_1 = \{J4, J6, J1\}$, $Mach_2 = \{J3, J5\}$ and $PJ = \{J2\}$. Job $J2$ can not be scheduled at the beginning. Job $J2$ is sequenced at the end following the two strategies explained before: in the first one, the job is assigned to the machine where it was originally assigned, and in the second, the job is assigned to the machine resulting in the minimum C_i. So, in the first case, job $J2$ is assigned at the end of machine 2, then $Mach_1 = \{J4, J6, J1\}$, $Mach_2 = \{J3, J5, J2\}$ and $C_{\max} = 6$. In this case, it is possible to swap job $J2$ with the previous job $J5$, since there are no idle times, the resource constraint is satisfied and the resource consumption of job $J5$ is lower than the resource consumption of job $J2$. No more swaps involving job $J2$ are allowed since the resource constraint is not satisfied. Swapping the jobs allows for more available resources at the end of the machine in order to sequence new Pending Jobs. For the second strategy, $J2$ is sequenced in machine 2 as well. Then, the final solution is $Mach_1 = \{J4, J6, J1\}$, $Mach_2 = \{J3, J2, J5\}$ and $C_{\max} = 6$. Note that there are no idle times in the solution to this example.

4.2. Local search

Local search methods are widely applied to parallel machine scheduling problems in order to improve solutions obtained by constructive procedures (see Fanjul-Peyro and Ruiz (2010), Fanjul-Peyro and Ruiz (2011), Vallada and Ruiz (2011) and Rodriguez et al. (2013), among others). Once the assignment A is transformed into a feasible solution S, a local search procedure is applied. Depending on the multi-pass heuristic, a local search is applied in a different way. In all cases, a local search is first applied to the machine resulting with the maximum completion time (makespan machine) and then to all machines, including again the makespan machine. More details about local search procedures are given in following subsections. Note that all local searches are applied consecutively and in the end, if there is at least one movement, the local search process starts again. The general structure for the local search is the following:

1. Insertion of jobs from the $C_{\max}$ machine(s).
2. Insertion of all jobs into all machines.
3. Swap of jobs from the $C_{\max}$ machine(s).
4. Swap of all jobs to all machines.

4.2.1. Local search considering resources

This local search is only applied if the original assignment A is already feasible, that is, there are no Pending Jobs. In this case, it is possible to apply a fast local search based on both insertion and swap neighbourhoods. In Algorithm 5 and Algorithm 6 pseudocodes for the local search considering resources are shown. Line 8 (Algorithm 5) and Line 10 (Algorithm 6) ensure the feasibility of the solution after an insertion/swap movement. In both cases, a local search is applied until the makespan value can not be improved.

Algorithm 5: Insertion considering resources ($C_{\max}$ machine).

```
1  for i = 1, ..., m do
2      if C_i = C_max then
3          Order Mach_i in non increasing order of processing times on
             machine i;
4          for j = 1, ..., |Mach_i| do
5              M_j := list of machines sorted by non decreasing p_ij;
6              for k = 1, ..., |M_j| do
7                  if k ≠ i then
8                      if R_0 + r_k[0] ≥ r_kj then
9                          Insert job j in machine k if C_k + p_kj < C_max;
10                         Order Mach_k in non increasing order of r_kj;
11                         Update R_0, C_k, C_max;
12                         break (all For loops);
```

After the local search based on insertion neighbourhood for the $C_{\max}$ machine, the same local search is also applied for all machines in a similar way to Algorithm 5 but without checking condition in Line 2. After the insertion neighbourhood, a local search based on swap neighbourhood is also applied as before, first to the $C_{\max}$ machine (Algorithm 6) and later to all machines. In both cases, the local search is applied while there is an improvement. Moreover, for both, insertion/swap considering $C_{\max}$ machine, first improvement strategy is applied. However, when the insertion/swap is applied to all machines, best improvement strategy is carried out.

Algorithm 6: Swap considering resources ($C_{\max}$ machine).

```
1  i_max := First machine so that its completion time is equal to C_max;
2  Order Mach_{i_max} in non increasing order of processing times;
3  for j = 1, ..., |Mach_{i_max}| do
4      M_{C_i} := List of machines sorted by non decreasing order of C_i;
5      for k = 1, ..., |M_{C_i}| do
6          Order Mach_k in non decreasing order of processing times on
             machine i_max;
7          for l = 1, ..., |Mach_k| do
8              NewC_max := C_max after swapping jobs j and l;
9              if NewC_max < C_max then
10                 if r_{i_max[0]} ≥ r_{i*l} and r_{k[0]} ≥ r_{kj} then
11                     Swap jobs j and l; Update i_max, C_k, C_max;
12                     Order Mach_k and Mach_{i_max} in non increasing order
                         of resource consumption;
13                     break (all For loops);
```

4.2.2. Local search without considering resources

The previous local search is only applied when there are no Pending Jobs in the original assignment, that is, the assignment is also a feasible solution. If the original assignment is not feasible, a different local search also based on insertion and swap neighbourhood is applied after applying the repair mechanism (Algorithm 4) to the assignment. In this local search, movements do not consider the resource constraint, that is, Lines 8 and 10 are not checked in Algorithms 5 and 6 respectively. In this way, the local search procedure is faster and several movements can be carried out using substantially less CPU time than when considering the resource constraint all along. In this local search, only the assignment is used, that is, if there are idle times due to the resource constraint, these idle times are removed and the schedule is compact. A feasible solution S is transformed into a non feasible assignment A. Then, two strategies can be used:

1. **Intensive local search:** every time a movement is carried out in a neighbourhood, the assignment is repaired to obtain a feasible solution following Algorithm 3.

2. **Non intensive local search:** according to the structure explained at the beginning of Section 4.2, Algorithm 3 is applied to the best solution obtained at the end of each type of local search. In this case, additional conditions are checked before carrying out a movement, in order to avoid implementing as much as possible, the repair mechanism. Specifically, for the insertion neighbourhood from $C_{\max}$ machine, movements are applied only if the resource consumption of the job in the new machine is not greater than the original resource consumption of the job. When swapping jobs from the $C_{\max}$ machine, if the total consumption of the swapped jobs after the movement is lower than the initial consumption of the same jobs, the movement is applied.

4.3. Multi-pass heuristics

Each multi-pass proposed heuristic consists of a combination of the elements explained above. Each multi-pass is run for each assignment rule, so at the end of each multi-pass method, 8 solutions are available and the best one is selected. More specifically, the following combinations are considered:

- *Multi-pass 1 (M1)*: After the constructive phase, the *local search considering resources* is applied only if the initial assignment is feasible.
- *Multi-pass 2 (M2)*: After *M1*, every solution is improved by applying the *non intensive local search*.
- *Multi-pass 3 (M3)*: After *M2*, every solution is improved by applying a procedure in order to *unbalance* the makespan machine. Local search procedures can be exhausted, machines are too *balanced* and it is not possible to improve the makespan by inserting/swapping jobs. *Balance* means that all C_i are very similar and it is not possible to improve the solution. A procedure to *unbalace* the makespan machine introducing *diversification* is carried out. The *unbalance* approach consists of different steps:
    1. Obtain the assignment A from the sequence S (eliminate idle times from S).
    2. Pending Jobs construction.
    3. Insert Pending Jobs in the makespan machine ($i_{\max}$).
    4. Improve the current solution by means of the non intensive local search.

If there are no Pending Jobs to insert in machine $i_{\max}$, the *unbalanced* procedure inserts a job j in machine $i_{\max}$ if this machine is the best one from the processing time ($p_{i_{\max}j}$) or resource consumption ($r_{i_{\max}j}$) point of view, $j = 1, ..., n$.

- *Multi-pass 4 (M4)*: This method is run in order to obtain two separate solutions: on the one hand, the solution provided by *M3*. On the other hand, the solution provided by *M1* which is improved by the intensive local search. At the end of the process, two solutions are obtained and the best one is selected.

- *Multi-pass 5 (M5)*: After *M3*, every solution is improved by applying the intensive local search.

Note that the worst case computational complexity is the same for all cases, $\mathcal{O}(n^2m)$.

5. Computational Results

In this section, comprehensive computational results on a benchmark originally proposed by Fanjul-Peyro et al. (2017) are given. The proposed benchmark consists of three sets of small, medium and large instances. Regarding the size of the instances, the following values are considered: small set, $n = \{8, 12, 16\}$, $m = \{2, 4, 6\}$; medium set, $n = \{20, 25, 30\}$, $m = \{2, 4, 6\}$; large set, $n = \{50, 150, 250, 350\}$, $m = \{10, 20, 30\}$. For each combination of number of jobs and number of machines, there are five sets according to processing times and resource consumption generation. Specifically, for processing times, uniform and correlated distributions are considered ($U(1, 100)$, $U(10, 100)$, $U(100, 200)$, Correlated Jobs, Correlated Machines). Regarding resource consumptions, two sets are generated: 1) following a uniform distribution ($U(1, 9)$) and 2) correlated values (the highest processing time, the highest resource consumption). Finally, the total number of the resource is computed as $R_{\max} = 5 \cdot m$. All the details about the benchmark generation can be found in Fanjul-Peyro et al. (2017). There are five replicates for each combination, therefore in total there are 450 small instances, 450 medium instances and 600 large instances. All of them are available at `http://soa.iti.es`, as well as the generator to reproduce exactly the same benchmark. All methods have been run on 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. The specific tests are performed on virtual machines running on this cluster. Each one runs a Microsoft Windows 7 64 bit operating system and has one single virtual processor and 2 GBytes of RAM. Therefore, the instances are randomly distributed to virtual machines in order to speed-up the experiments. The platform used for the codes is Microsoft Visual Studio 2013 and the methods were coded in C# under the same .NET Framework (4.6.1).

Regarding the effectiveness measure, the Relative Percentage Deviation (RPD) is computed for each instance according to the following expression:

$$\text{Relative Percentage Deviation}(RPD) = \frac{Heu_{sol} - Best_{sol}}{Best_{sol}} \cdot 100, \quad (1)$$

where Heu_{sol} is the solution obtained with a given proposed heuristic and $Best_{sol}$ is the best known solution for a given instance. For small and medium instances $Best_{sol}$ is a lower bound recently provided by Fanjul-Peyro et al. (2017), where several matheuristic methods along with mathematical models are proposed for the same problem. Specifically, the lower bound is computed as the maximum value between the optimal solution or lower bound provided by the mathematical models and the lower bound provided by the mathematical models for the same problem without considering the resource constraint. For large instances, $Best_{sol}$ is the best known solution obtained after all the experiments have been carried out throughout the paper.

Both heuristic approaches are tested. For the first one (Section 3), the best combination for each constructive method (Section 3.2) is tested. Then, two final heuristics are provided, denoted as SWA and NEH_{res}. Regarding the second approach (Section 4), the five multi-pass heuristics are tested (Section 4.3), denoted as $M1$, $M2$, $M3$, $M4$ and $M5$. Moreover, a simple adaptation of the heuristic NEH proposed by Nawaz et al. (1983) is included in the comparative evaluation and denoted as NEH_{st}. This adaptation consists of sorting the jobs into non increasing order of the sum of the p_{ij}, for each job j on all machines. Then, jobs are selected in this order and inserted on the machine so that the makespan value is the minimum with the resource constraint being considered throughout the entire insertion process. Note that the order of the jobs is completely different from NEH_{res}.

5.1. Small and medium instances

First, a comparative evaluation of all the proposed methods using small and medium instances is carried out. Moreover, the best mathematical model $UPMR-P$ and the best matheuristic $JMR-P$ presented by Fanjul-Peyro et al. (2017) are also included, and here denoted as $UPMR$ and JMR respectively. In Tables 4 and 5, the Average Relative Percentage Deviation for each $n \times m$ size group is shown for small and medium instances respectively. All methods are tested under the same conditions and using the same computational resources, being the same used in Fanjul-Peyro et al. (2017). Multi-pass heuristics $M4$ and $M5$ and NEH_{res} obtained the best results. This is expected since these methods use local search procedures to improve the solution. In order to check the results, an analysis of variance (ANOVA) (Montgomery (2012)) is applied to validate if the differences in the RPD values are statistically significant. We can see in Figure 2 the means plot with LSD intervals (α=0.05), where it is shown that there are no statistically significant differences (overlapped intervals) among $M4$, $M5$ and NEH_{res}, that is, on average the three methods perform similarly . For the remaining methods, the mathematical model ($UPMR$) and the matheuristic (JMR) perform well, similarly to $M2$ and $M3$ multi-pass heuristics. It is clear that $M1$ heuristic and the standard adaptation of the NEH (NEH_{st}) perform the worst since they are the simplest methods. These results are consistent regardless of the size of the instance and generation of processing and resource consumption times. However, according to the average CPU times needed by each method (Table 4), multi-pass heuristics are much more efficient than the NEH_{res} method. So, in conclusion, $M4$ and $M5$ multi-pass heuristics perform similarly while using much less computation time. The CPU time needed by the mathematical model ($UPMR$) and the matheuristic was a maximum of 1 hour for each one, using IBM ILOG-CPLEX 12.6 to solve the instances according to Fanjul-Peyro et al. (2017). Regarding the number of optimal solutions, NEH_{res} obtained 291 over 450 small instances, while $M4$ and $M5$ got 267 and 257 respectively. Methods proposed by Fanjul-Peyro et al. (2017) were able to optimally solve 261 small instances ($UPMR$) and 243 small instances (JMR). In more detail, the mathematical model $UPMR$ is able to optimally solve all 8 jobs instances (150 instances), 85 over 150 instances with 12 jobs and just 26 over 150 instances with 16 jobs. Regarding the matheuristic JMR, it is able to solve 126 over 150 instances with 8 jobs, 86 over 150 instances with 12 jobs and just 31 over 150 instances with 16 jobs. Therefore, both methods are especially sensitive to the size of the instance.

None of the heuristic methods can optimally solve all 8 jobs instances, but their performance improves as the size of the instance increases.

Instance group	$UPMR$	JMR	NEH_{st}	SWA	NEH_{res}	$M1$	$M2$	$M3$	$M4$	$M5$
8×2	**0.00**	**0.00**	2.50	1.11	0.25	3.61	1.10	1.04	0.10	0.24
8×4	**0.00**	1.94	4.33	1.00	0.21	8.37	1.92	1.71	0.73	1.17
8×6	**0.00**	1.88	4.71	0.24	0.19	2.81	2.10	2.04	1.06	0.96
12×2	**3.80**	**3.80**	7.34	5.02	4.09	7.50	5.04	5.00	4.41	4.49
12×4	1.89	3.20	7.46	3.43	**1.75**	11.11	4.22	3.87	2.62	2.91
12×6	1.46	1.07	6.58	2.21	**0.65**	5.74	3.08	2.90	1.16	1.19
16×2	**8.10**	**8.10**	12.50	11.26	8.61	13.04	9.55	9.41	8.50	8.58
16×4	7.80	5.52	9.37	4.11	**2.13**	10.33	3.67	3.56	2.65	2.98
16×6	8.92	6.19	7.92	2.68	**1.01**	8.30	3.82	3.27	1.59	1.79
$Av.RPD$	3.55	3.52	6.97	3.45	**2.10**	7.87	3.83	3.64	2.53	2.70
Av.Time	2208.34	1969.92	4.84	62.59	115.26	2.64	7.97	16.44	44.9	35.22

Table 4: Average Relative Percentage Deviation (RPD) and Average CPU Time in milliseconds for small instances (in seconds for $UPMR$ and JMR).

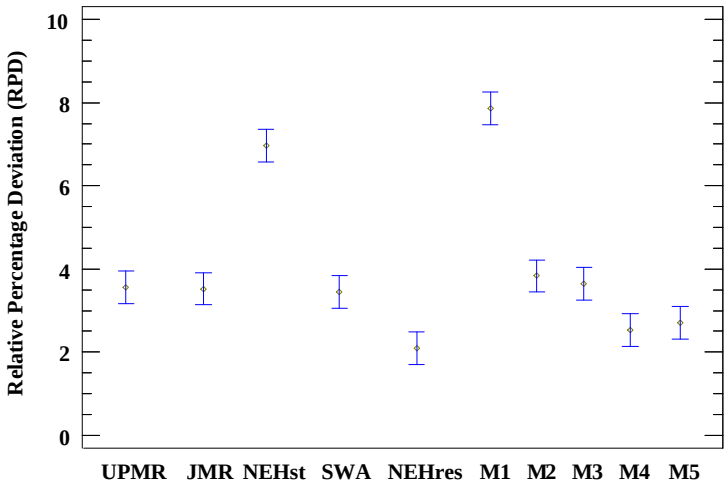

Figure 2: Means Plot and LSD intervals at the 95% confidence level for the heuristic methods (small instances)

Regarding medium instances, results are shown in Table 5 and the statistical analysis in Figure 3. The picture is slightly different from that of small instances. In the medium size case, the mathematical model ($UPMR$) and the matheuristic (JMR) do not perform well since the size of the instance has a large negative impact for these methods. Regarding the best methods, again

Instance group	$UPMR$	JMR	NEH_{st}	SWA	NEH_{res}	$M1$	$M2$	$M3$	$M4$	$M5$
20×2	**6.98**	**6.98**	10.45	9.74	**6.98**	12.36	8.75	8.33	7.18	7.42
20×4	13.91	10.93	8.14	4.56	2.48	9.50	3.86	2.96	2.48	**2.33**
20×6	14.69	9.72	7.45	3.55	1.57	7.28	2.97	2.52	1.35	**1.29**
25×2	13.21	13.21	12.80	12.60	10.15	14.38	11.02	10.88	**9.95**	9.99
25×4	21.02	15.24	8.69	4.53	**2.62**	10.26	4.26	3.80	2.66	2.71
25×6	25.00	19.39	8.62	4.11	**2.05**	7.93	3.14	2.58	2.09	2.12
30×2	15.25	15.25	8.36	8.02	**5.30**	10.64	6.50	6.37	5.40	5.57
30×4	28.26	21.52	7.53	3.70	1.88	8.64	2.42	2.26	**1.28**	1.41
30×6	61.24	30.27	10.24	5.23	2.89	10.14	3.45	2.49	1.89	**1.59**
$Av.RPD$	22.17	15.84	9.14	6.23	3.99	10.12	5.15	4.69	**3.81**	3.83
Av.Time	3600	3600	45.97	371.08	1458.53	6.13	26.18	51.34	196.50	146.56

Table 5: Average Relative Percentage Deviation (RPD) and Average CPU Time in milliseconds for medium instances (in seconds for $UPMR$ and JMR).

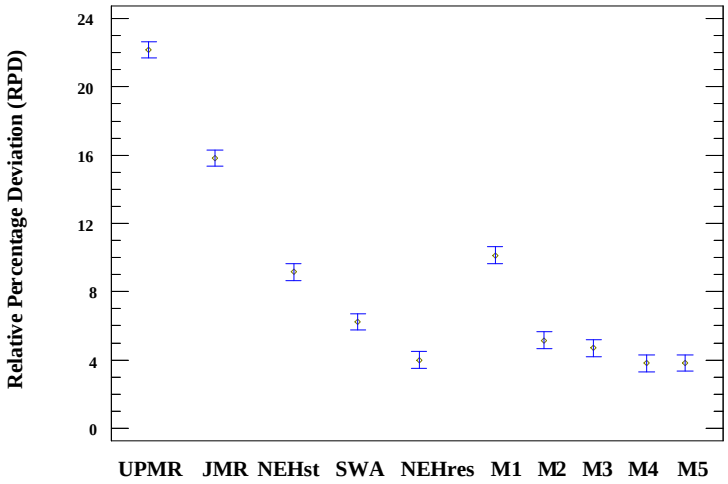

Figure 3: Means Plot and LSD intervals at the 95% confidence level for the heuristic methods (medium instances)

$M4$, $M5$ and NEH_{res} demonstrate the best performance without statistically significant differences. In conclusion, the three methods are statistically similar but multi-pass heuristics ($M4$ and $M5$) need much less computational effort (Table 5). Again these results are consistent regardless of the size of the instance and generation of processing and resource consumption times. If we focus our attention on the number of optimal solutions obtained by each method, $M4$ and $M5$ are able to obtain 143 and 131 over 450 medium instances respectively. NEH_{res} is only able to get 44 optimal solutions. The mathematical model ($UPMR$) and the matheuristic (JMR) can solve just 8 instances each one, far inferior to the best heuristic methods. Note that in this case, optimal solutions for all medium size instances are not known, so the number of optimal solutions obtained by the methods is a lower bound, that is, at least they are able to optimally solve this amount of medium instances. In conclusion, we can state that the bigger the size of the instance, the better the performance is of the heuristics compared to $UPMR$ and JMR.

5.2. Large instances

For large instances, NEH_{res} is not tested since the computational time needed is too long and the results are not competitive. Finally, the mathematical model ($UPMR$) and the matheuristic (JMR) are not considered since they are not able to solve large instances. In Table 6, the results for the rest of the methods are shown. Again, the $M4$ and $M5$ heuristics perform the best, obtaining on average a smaller Relative Percentage Deviation than the rest of the methods. According to Table 6, it seems that $M5$ works better than $M4$, however if we apply an analysis of the variance (ANOVA) (Figure 4), both methods are overlapped, so there are no statistically significant differences, that is, $M4$ and $M5$, on average, are similar. If we pay attention to the CPU times (Table 6), on average, $M5$ requires less computational effort than $M4$. Therefore, $M5$ can be considered the best method if we take into account both the quality of the solution and efficiency. For the remaining methods, $M1$ and the SWA method do not work well. The effectiveness of the $M2$ and $M3$ heuristics is not as good but both methods are much faster (less than 10 seconds on average) than $M4$ and $M5$. For large instances, optimal solutions are not known for the problem considering the resource constraint. However, it is possible to compare the solutions obtained by the heuristics against a lower bound for the Unrelated Parallel Machine Scheduling Problem (UPM), that is, without considering the resource. If the heuristic methods are able to obtain the same optimal makespan value as that in the UPM case, it means

the solution is also optimal for the $UPMR$ problem which takes resources into consideration. The SWA method is able to obtain 22 optimal solutions. Regarding multi-pass heuristics, 12, 24 and 30 optimal solutions are obtained by $M1$, $M2$ and $M3$ respectively. Multi-pass $M4$ and $M5$, the best ones, are able to optimally solve 50 and 51 large instances respectively. Note that the same makespan values do not mean the same solution. Actually, it is very likely that the optimal solution for the UPM problem is non feasible for the $UPMR$ one. Therefore, even if makespan values are similar and optimal for UPM and $UPMR$, the sequence of the jobs for each machine has not to be similar. Then, the proposed heuristic methods obtain the optimal solution for $UPMR$ after applying all the procedures explained above in order to obtain a feasible solution.

Instance group	SWA	M1	M2	M3	M4	M5
50×10	4.58	7.51	1.95	1.22	**0.24**	0.28
50×20	3.59	4.66	2.05	1.73	0.92	**0.51**
50×30	2.12	3.05	2.02	1.57	**0.24**	0.28
150×10	5.37	6.19	1.00	0.74	0.35	**0.08**
150×20	5.63	6.43	0.68	0.42	**0.11**	0.16
150×30	4.98	7.04	1.18	0.71	0.35	**0.18**
250×10	4.46	6.31	0.82	0.57	0.26	**0.04**
250×20	5.50	5.63	0.65	0.35	0.20	**0.01**
250×30	5.22	5.47	0.79	0.54	0.29	**0.06**
350×10	4.29	5.41	0.75	0.57	0.19	**0.04**
350×20	5.59	5.16	0.58	0.40	0.24	**0.03**
350×30	5.17	4.69	0.50	0.35	0.14	**0.01**
$Av.RPD$	4.71	5.63	1.08	0.76	0.29	**0.14**
Av.Time (sec.)	46.3	1.60	5.25	9.84	184.93	102.25

Table 6: Average Relative Percentage Deviation (RPD) and Average CPU Time in seconds for large instances.

6. Conclusions and future research

In this paper, the unrelated parallel machine scheduling problem with additional resources is considered. Several heuristics are proposed following

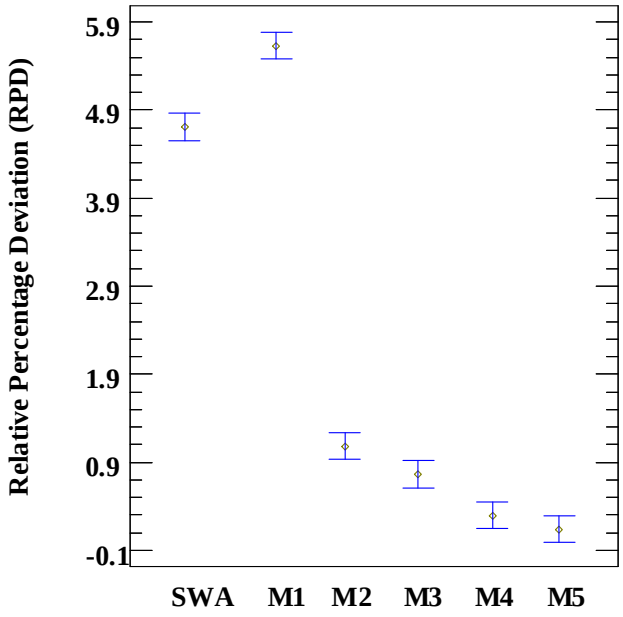

Figure 4: Means Plot and LSD intervals at the 95% confidence level for the heuristic methods (large instances)

two different approaches: the first one considers the resource constraint during the whole process while the second one works with non feasible solutions as regards resources, applying a procedure to repair the solutions. Two heuristics are tested according to the first approach based on resources and five multi-pass heuristics are proposed according to the second one based on assignment. An exhaustive comparative evaluation is carried out under an extensive benchmark of small, medium and large instances. From the results obtained, we can conclude that the heuristics from the first approach together with multi-pass $M4$ and $M5$ from the second approach, demonstrate the best performance in small and medium instances. In terms of efficiency, heuristics from the second approach are far superior to those of the first one. Regarding large instances, multi-pass heuristics $M4$ and $M5$ obtained the best results, improving on all other tested methods. The main contribution of the paper is that the proposed heuristics can efficiently solve instances of different sizes, especially large instances, something that, at present, does not exist in the literature. As a conclusion, we can state that the second approach is more suitable than the first one, since the results obtained by the multi-pass heuristics are equal to or better than those obtained by the

methods of the first approach whilst using much less CPU time. Moreover, heuristics improve on the only methods proposed until now for the same problem: the mathematical models and matheuristics presented by Fanjul-Peyro et al. (2017), from the point of view of efficiency (small instances) and effectiveness and efficiency (large and medium instances). Regarding future research, considering the good performance of the proposed heuristics, the authors are starting to work in metaheuristic methods. These methods could start from the solutions provided by the heuristics presented in this paper, with the objective of improving the results.

Acknowledgments

The authors are supported by the Spanish Ministry of Economy and Competitiveness, under the projects "SCHEYARD - Optimization of Scheduling Problems in Container Yards" (No. DPI2015-65895-R) and "OPTEMAC - Optimización de Procesos en Terminales Marítimas de Contenedores" (No. DPI2014-53665-P), all of them partially financed with FEDER funds. The authors are also partially supported by the EU Horizon 2020 research and innovation programme under grant agreement no. 731932 "Transforming Transport: Big Data Value in Mobility and Logistics". Interested readers can download contents from `http://soa.iti.es`, like the instances used and a software for generating further instances. Source codes are available upon justified request from the authors.

References

Alvarez-Valdes, R., Tamarit, J. M., Villa, F., 2015. Minimizing weighted earliness-tardiness on parallel machines using hybrid metaheuristics. Computers and Operations Research 54, 1–11.

Arroyo, J. E. C., Leung, J. Y.-T., 2017. Scheduling unrelated parallel batch processing machines with non-identical job sizes and unequal ready times. Computers and Operations Research 78, 117–128.

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, 75–81.

Bitar, A., Dauzère-Pérès, S., Yugma, C., Roussel, R., 2016. A memetic algorithm to solve an unrelated parallel machine scheduling problem with auxiliary resources in semiconductor manufacturing. Journal of Scheduling 19, 367–376.

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

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

Chen, J., 2015. Unrelated parallel-machine scheduling to minimize total weighted completion time. Journal of Intelligent Manufacturing 26, 1099–1112.

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.

Daniels, R. L., Hua, S. Y., Webster, S., 1999. Heuristics for parallel-machine flexible resource scheduling problems with unspecified job assignment. Computers and Operations Research 26, 143–155.

Edis, E. B., Oguz, C., 2012. Parallel machine scheduling with flexible resources. Computers and Industrial Engineering 63, 433–447.

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, 449–463.

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, 135–157.

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

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

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

Fanjul-Peyro, L., Ruiz, R., 2011. Size-reduction heuristics for the unrelated parallel machines scheduling problem. Computers and Operations Research 38, 301–309.

Garey, M., Johnson, D., 1975. Complexity results for multiprocessor scheduling under resource constraints. SIAM Journal on Computing 4, 397–411.

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.

Grigoriev, A., Sviridenko, M., Uetz, M., 2007. Machine scheduling with resource dependent processing times. Mathematical Programming Series B 110, 209–228.

Hsieh, P. H., Yang, S. J., Yang, D. L., 2015. Decision support for unrelated parallel machine scheduling with discrete controllable processing times. Applied Soft Computing 30, 475–483.

Hsu, Chou, J., Yang, D. L., 2014. Unrelated parallel-machine scheduling with position-dependent deteriorating jobs and resource-dependent processing time. Optimization Letters 8, 519–531.

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

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

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

Montgomery, D. C., 2012. Design and Analysis of Experiments, eigth Edition. John Wiley & Sons, New York.

Nawaz, M., Enscore, Jr, E. E., Ham, I., 1983. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. OMEGA, The International Journal of Management Science 11, 91–95.

Rodriguez, F. J., Lozano, M., Blum, C., García-Martínez, C., 2013. An iterated greedy algorithm for the large-scale unrelated parallel machines scheduling problem. Computers and Operations Research 40, 1829–1841.

Ruiz-Torres, A. J., López, F. J., Ho, J. C., 2007. Scheduling uniform parallel machines subject to a secondary resource to minimize the number of tardy jobs. European Journal of Operational Research 179, 302–315.

Vallada, E., Ruiz, R., 2011. A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. European Journal of Operational Research 211, 612–622.

Ventura, J. A., D., K., 2003. Parallel machine scheduling with earliness-tardiness penalties and additional resource constraints. Computers and Operations Research 30, 1945–1958.

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

Yin, N., Kang, L., Sun, T. C., Yue, C., Wang, X. R., 2014. Unrelated parallel machines scheduling with deteriorating jobs and resource dependent processing times. Applied Mathematical Modelling 38, 4747–4755.

Zeidi, J., MohammadHosseini, S., 2015. Scheduling unrelated parallel machines with sequence-dependent setup times. International Journal of Advanced Manufacturing Technology 81, 1487–1496.

Zheng, X., Wang, L., 2016. A two-stage adaptive fruit fly optimization algorithm for unrelated parallel machine scheduling problem with additional resource constraints. Expert Systems with Applications 65, 28–39.