GRASP algorithms for the unrelated parallel machines scheduling problem with additional resources during processing and setups

Axel Lopez-Esteve[1], Federico Perea[2] and Juan C. Yepes-Borrero[3]
[1]axloes@inf.upv.es, Universitat Politècnica de València.
Camino de Vera sn. 46022, Valencia, Spain
[2] Corresponding author: perea@us.es, Universidad de Sevilla.
Departamento de Matemática Aplicada II, EPS,
Calle Virgen de África 7, 41011, Sevilla, Spain.
[3] juanca.yepes@urosario.edu.co, Universidad del Rosario.
Escuela de Ingeniería, Ciencia y Tecnología,
Cl. 12c N 6-25, 111711, Bogotá, Colombia

February 22, 2026

Abstract

This paper addresses an unrelated parallel machines scheduling problem with the need of additional resources during the processing of the jobs, as well as during the setups that machines need between the processing of any two jobs. This problem is highly complex, and therefore in this paper we propose several constructive heuristics to solve it. In order to improve the performance of these heuristics, we propose several variations, including randomization with different probability distributions and a local search phase, having this way GRASP algorithms. The results of extensive experiments over randomly generated instances show several findings on the different parameters that characterize our constructive algorithms. In particular, we highlight the fact that non-uniform probability distributions might be advisable for choosing elements of a restricted candidate list in GRASP algorithms.

Keywords: Parallel machines; Scheduling; Sequence dependent setup times; Additional resources; Metaheuristics; GRASP.

1 Introduction

In the current competing world, efficiency is a key aspect for all companies, either public or private. Delivering products on time, correctly managing the staff, or keeping inventory at a correct level, may well be the difference between success and bankruptcy. Within the variety of problems found in the logistics of a company, those regarding production planning are attracting more and more attention both in the industry and in the scientific literature. Out of the many problems found in production planning, in this paper we focus on a particular unrelated parallel machines scheduling problem. The simplest version of this problem consists of assigning machines to jobs in a production plant. Traditionally, most papers in the literature assume that machines process the jobs automatically, and therefore the machines are the only limiting resource. However, it is often realistic to assume that the functioning of the machines must be supervised by additional resources, e.g. workers. Therefore, when jobs are assigned to machines, the availability of personnel becomes another extra constraint, which makes the scheduling problem even harder to solve. Besides, in this paper we also consider that the necessary adjustment of the machines between the processing of two consecutive jobs (typically referred to as *setup*), must also be supervised by the additional resources, adding this way even more complexity.

Ideally, one would like to have the optimal solution to these optimization problems. For this, a solver that is able to deal with large mathematical programming problems is needed, often under commercial license. However, in many production plants where this type of scheduling problems may arise, managers and engineers do not have access to such commercial solvers. This is an added advantage to the use of heuristics and metaheuristics, apart from the considerable savings in CPU time obtained since they normally are able to return a feasible solution in much less time than an engine that solves a mathematical problem. Therefore, the design of algorithms which do not rely on mathematical programming models is needed.

Although there are some exact algorithms proposed for scheduling problems with additional resources, the high computational complexity of these problem makes that most efforts in the related literature go into the direction

of heuristics and metaheuristics.

It is essential for production planners to have decision-making tools that allow them to plan their tasks, or reschedule them in (nearly) zero time. Arguably, exact algorithms take longer to provide a solution than heuristics and metaheuristics. Besides, as will be proven in the experiments section, metaheuristics can provide nearly-optimal solutions in very short CPU times when they are correctly calibrated.

The main contributions of this paper are:

1. the design of efficient heuristics and metaheuristics for the Unrelated Parallel Machines Scheduling problem with additional Resources during Processing and Setups (problem that we denote as UPMSR-PS).

2. the empirical proof of the importance of several factors to be considered in the design of such algorithms: randomization, consideration of the resources in the constructive phase, assigning different probabilities to the elements in the restricted candidate list, allowed CPU time, and local search.

The rest of this paper is structured as follows. In Section 2 we summarize the related literature. Although the UPMSR-PS treated in this paper has been proposed before, we formally describe it in Section 3 for the sake of completeness. The heuristics and metaheuristics proposed for the UPMSR-PS are presented in Section 4. In Section 5, these algorithms are tested and compared with an exact decomposition approach proposed in the literature.

2 Literature Review

In scheduling problems, a set of jobs to be processed is given, as well as a set of machines that can process the jobs. The Unrelated Parallel Machines scheduling problem (UPM), consists of assigning one of the available machines to each of the jobs to be processed, so that a certain objective is optimized. One of the objectives most commonly found in these problems is the *makespan*, defined as the total duration of the schedule. When minimizing the makespan, the UPM problem is already NP-hard even for the case of two machines, as proven in Garey and Johnson (1979). The literature on the UPM problem is quite extensive. The interested reader is referred to Fanjul-Peyro and Ruiz (2010), Fanjul-Peyro and Ruiz (2011), Fanjul-Peyro and Ruiz (2012) and the references therein.

A more realistic version of the unrelated parallel machines scheduling problem also considers setups between consecutive jobs (denoted as UPMS). The number of research papers in the scientific literature dealing with variants of this problem is also large, as shown by the review paper Allahverdi (2015), where hundreds of articles are cited. Due to the fact that the UPMS is quite hard to solve, most of the efforts have gone into the direction of heuristics and metaheuristics (see e.g. Diana et al. (2014) and Avalos-Rosales et al. (2015) for two of the most successful metaheuristics for this problem). However, the interested reader can also find some references that propose exact approaches, which have successfully solved large instances of the UPMS. Two examples of such approaches are found in Tran et al. (2016), where decomposition techniques are proposed, and Fanjul-Peyro et al. (2019), who refine existing models and algorithms for the UPMS.

Another even more realistic problem arises when machines need additional resources in the production plant. In other words, machines cannot process jobs automatically and/or being adjusted without the help of additional resources, such as workers. Although the literature that considers additional resources is less extensive than the literature for scheduling problems that only consider the machines as the limiting resource, in the last years more and more research is being done in this direction. For example, Ozer and Sarac (2019) model an identical parallel machine scheduling problem using mixed integer programming, and a genetic algorithm is proposed for solving it more efficiently. In Prata et al. (2021), several constructive heuristics in combination with local search heuristics for a single-machine scheduling problem with periodical resource constraints are presented. However, such problems with additional resources and more than one unrelated parallel machine are not so common in the literature. The problem in which (several) machines need additional resources in order to process jobs, denoted as UPMR, has been addressed in only a few papers since it was proposed by Edis et al. (2013). Among the few papers that have addressed this problem, the following are somehow similar to the one we study here: the new mathematical models and three matheuristics in Fanjul-Peyro et al. (2017), the heuristic approaches in Villa et al. (2018), and metaheuristics algorithms in Vallada et al. (2019). Specific cases of this problem are also treated in Fleszar and Hindi (2018).

Other papers dealing with similar scheduling problems of parallel machines under the presence of additional resources are the following. Abdeljaoued et al. (2020) assume that the processing of each job requires one specific single additional resource. The additional resource in Çanakoglu and Muter

(2020) consists of components with discrete levels. In Chen et al. (2018), processing times can be reduced thanks to the available resources. In Fu et al. (2019), the additional resource is allocated to machines in advance, and the processing times depend on the amount of allocated resources. The problem treated by Lee et al. (2020) assumes that jobs can be split into different sections, which in turn can be processed on different machines under the requirements of setup operators. Li et al. (2019) add more complexity by assuming uncertain processing times. Sekkal and Belkaid (2020) assume that machines are subject to deterioration, and consider two resource consumption constraints with the objective of simultaneously optimizing both the makespan and the resource costs. Yunusoglu and Topaloglu Yildiz (2021) consider a multi-resource problem with setups and other operational constraints, and develop algorithms based on constraint programming.

Even less references about the unrelated parallel machines scheduling problem with setups and the need of additional resources during setups (which we denote as UPMSR-S) are found in the literature. This problem was introduced in Yepes et al. (2020), where a GRASP algorithm was also proposed and tested. A multiobjective approach is proposed in Yepes et al. (2021), where both the number of additional resources and the makespan are optimized simultaneously. As far as we know, no more papers on this problem have been published so far.

The problem addressed in this paper adds another realistic feature, by combining UPMR and UPMSR-S. Namely, machines now need additional resources both for processing jobs (as in UPMR) and for the setups (as in UPMSR-S). This new problem is denoted as UPMRS-PS, and was proposed in Fanjul-Peyro (2020), together with mathematical models and a decomposition exact approach. In that paper, seven different variants of the UPMRS-PS are proposed, depending on the tasks that the resources may do (processing and/or setups).

3 Problem statement

In this section, we formally describe the specifications of the Unrelated Parallel Machines Scheduling problem with additional Resources during Processing and Setups (UPMSR-PS), as follows:

- Let $N = \{1, ..., n\}$ be the set of jobs to be processed, indexed by j and

k. For modelling purposes, we define $N_0 = N \cup \{0\}$, where $\{0\}$ is a dummy job.

- Let $M = \{1, ..., m\}$ be the set of available machines, indexed by i. Once a machine starts processing a job, it cannot be assigned to another job until such processing is finished (i.e., no preemption is allowed). Machines can process at most one job at a time, although they can work in parallel (different machines might be processing different jobs at the same instant).
- Let $p_{ij} \geq 0$ be the processing time of job j on machine i.
- Let s_{ijk} be the setup time needed to reconfigure machine i between the processing of jobs j and k, in that order. We assume that once a setup begins, it cannot be stopped until its completion.
- Let $r^P_{\max}$ be the maximum number of resources that can be used for processing at the same time.
- Let $r^S_{\max}$ be the maximum number of resources that can be used for setups at the same time.
- Let r^P_{ij} be the necessary number of resources for processing job j on machine i.
- Let r^S_{ijk} be the necessary number of resources for the setup of machine i between jobs j and k.

The problem consists of deciding:

- Which machine processes each job.
- The sequence in which jobs are processed on each machine.
- The initial time of the processing of each job.
- The initial time of the setup of each machine, between the processing of any two consecutive jobs on the same machine.

Note that, whereas in the UPM the only decision to make is which machine processes each job, and in the UPMS you also need to decide the sequence in which jobs are processed, in the UPMSR-PS you also need to decide the time

at which both the processings and the setups are done. Therefore, UPMSR-PS is a problem that extends both UPM and UPMS.

The objective considered is the minimization of the *makespan*, denoted as $C_{\max}$, defined as the completion time of the last job. The problem we treat in this paper is denoted in Fanjul-Peyro (2020) as UPMSR-P+S, because we consider specific processing resources (they can only be used for processing) and specific setup resources (they can only be used for setups).

MILPs (mixed integer linear programs) are proposed in Fanjul-Peyro (2020), although only results on the small set of instances are reported. We include in the Appendix an adaptation of these mathematical models to the UPMSR-P+S.

Besides, a three-phase exact algorithm with satisfactory results is also proposed in Fanjul-Peyro (2020), which we will use in order to test the quality of the solutions returned by the GRASP proposed in this paper.

3.1 Example

In this section we introduce an illustrative example of the problem treated. We consider $n = 4$ jobs (J1,J2,J3,J4) and $m = 2$ machines (M1,M2), and available number of resources for setups and processing $R^S_{\max} = R^P_{\max} = 5$. Processing times and setup times, as well as resources needed for processing and setups, are detailed in Table 1. Note that the main diagonal of the s and r matrices represents the initial setup of machines, assuming the corresponding job is the first to be processed there.

In Figure 1 we represent two different solutions for this example. The horizontal axis represents time units. The two machines are in the vertical axis. The sequence of each machine is shown in such a way that setups are in blue and jobs processing are in red. Below each of them we see the amount of resources required.

We see how the top schedule is infeasible, since the consumption of processing resources at time 18 is 6, greater than the maximum allowed ($R^P_{\max} = 5$). However, in the bottom schedule, and thanks to the inclusion of idle times, the consumption of resources is always below the maximum available, and the solution is therefore feasible. Note as well how the makespan has increased from 22 in the (unfeasible) top schedule to 24 in the (feasible) bottom schedule.

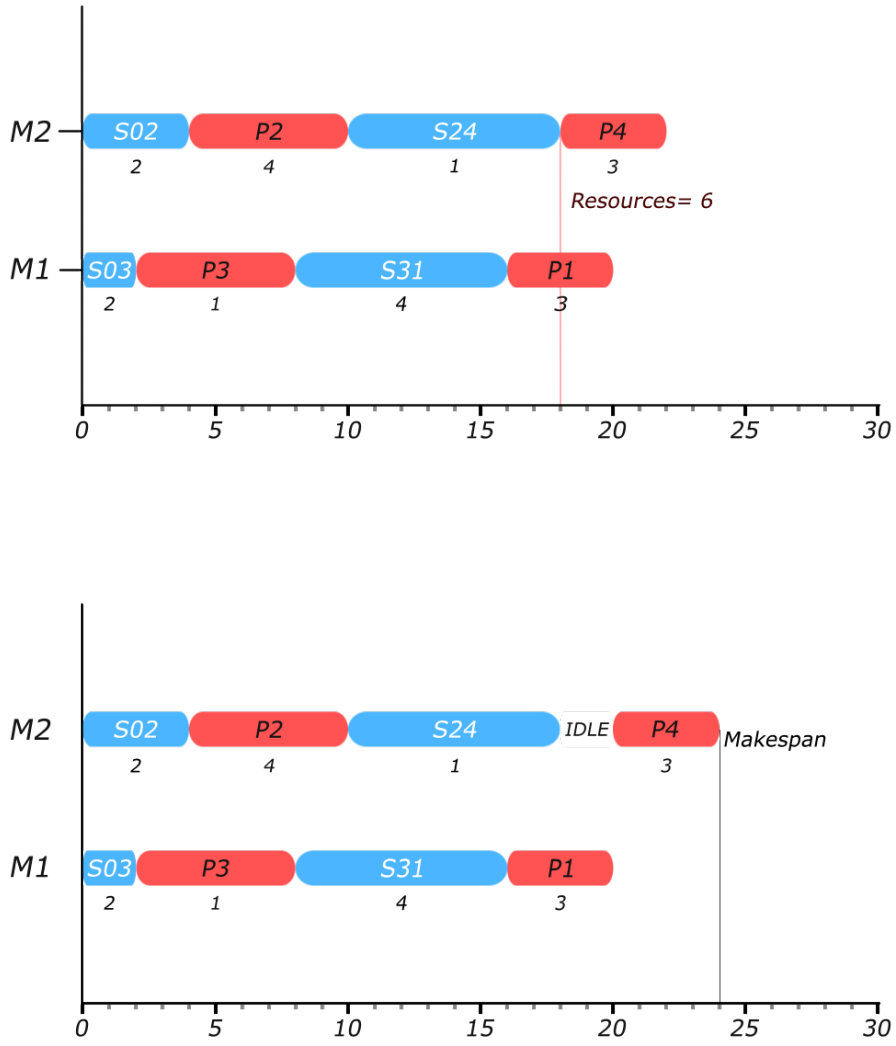

Figure 1: One infeasible solution (top schedule) and one feasible solution (bottom schedule), $R^S_{\max} = R^P_{\max} = 5$.

Figure 1 Alt Text: In the top schedule we see an infeasible solution, because at instant 18 the number of resources used is 6, exceeding the availability. In the bottom schedule the solution is feasible, as the availability of resources is never exceeded.

p_{ij}	M1	M2
J1	4	2
J2	5	6
J3	6	4
J4	8	4

s_{1ij}	J1	J2	J3	J4
J1	2	7	4	5
J2	5	2	6	9
J3	8	3	2	7
J4	8	8	4	2

s_{2ij}	J1	J2	J3	J4
J1	8	8	6	5
J2	10	4	4	8
J3	3	3	1	7
J4	4	2	8	4

r_{ij}^P	M1	M2
J1	3	1
J2	4	4
J3	1	2
J4	1	3

r_{1ij}^S	J1	J2	J3	J4
J1	2	3	1	1
J2	3	2	5	1
J3	4	1	2	2
J4	1	3	5	1

r_{2ij}^S	J1	J2	J3	J4
J1	1	2	1	4
J2	4	2	2	1
J3	4	4	2	2
J4	5	1	2	2

Table 1: Input data of example.

4 Heuristics and metaheuristics

Due to the complexity of the UPMSR-PS, and the need for good quality solutions in a reasonably small amount of time, efficient algorithms are needed. In this section we propose some heuristics and metaheuristics, which are based on constructing a sequence of jobs for each machine without idle times, and then repairing it (if necessary) by postponing jobs or setups, so that no more resources than those available are used at any point of time. We first explain some deterministic heuristic strategies, which are later randomized so they do not get stuck in local optima.

4.1 Heuristic strategies

The heuristic strategies proposed have two phases:

1. Constructive: In this phase, a solution to the corresponding UPMS problem is obtained, that is to say, the UPMSR-PS without considering the additional resources as a hard constraint. This phase will then yield a sequence of jobs for each machine, so that all jobs are processed and setups are done. Note that this solution may be unfeasible, as the amount of resources received may exceed the availability. Besides, in the experiments section we will test a variant of this strategy in which resources are somehow considered during this constructive phase. This phase will be further explained in Section 4.1.1.

2. Repairing: Since the sequences given in the constructive phase may not be feasible, because they may use more resources than those available at some point in time, a second phase is needed. In this phase, we first check if the sequence is feasible and, in case it is not, the sequence is repaired. This phase is further explained in Section 4.1.2.

We will design two variants of each phase: denoted as C1, C2 (constructive phases) and R1, R2 (repairing phases). By combining them, we have four different heuristics, denoted as: H1 (C1 + R1), H2 (C1 + R2), H3 (C2 + R1), H4 (C2 + R2). These four heuristic strategies will be tested in the experiments section. We now explain the two variants of each phase in detail. Since we have to deal with both the processing of jobs and the machine setups, and they are managed in similar ways, for the sake of space we will describe the algorithms for the jobs processing, indicating between parenthesis when we can also refer to setups.

4.1.1 Constructive phases

This is an iterative phase. Initially, the list of jobs that need to be assigned (denoted as *pending* jobs) is equal to N, and the makespan is zero. The constructive phase proposed consists of an initial step and an iterative process.

- Initial step. The job that consumes the largest number of resources for each machine is assigned to such machine.
- Iterative process. In each subsequent iteration, a job is assigned to one of the available machines, so that the overall makespan of the new solution is minimum. In other words, at each iteration we search for the job-machine assignment that implies the lowest increase in the makespan, with respect to the makespan of the iteration before. The corresponding job is removed from the list of pending jobs, and the next job-machine assignment is looked for. This phase finishes when there are no more pending jobs. Note that, when processing a new job, the makespan may remain constant with respect to the makespan in the iteration before.

We denote this constructive phase as C1. We will also test another variant of this constructive phase, denoted as C2, in which the initial step is not considered.

Algorithm 1 summarizes the two constructive phases, where N^* denotes the list of pending jobs, C^i denotes the makespan of machine i, C^{ij} denotes the makespan if job j was assigned to machine i on that iteration, and j_i denotes the last job processed on machine i at each iteration. While no jobs have been processed on a machine, we denote its corresponding j_i as $\emptyset$. Note that both phases only differ in the initial step, which in C1 is applied and in C2 is not. In either case, the result is a sequence of jobs processed on each machine. Note that no information about the resources is used in this phase. However, in the experiments section we will test a variant in which the choice of the job-machine assignment is based not only on the new makespan, but also on the resources required. Since this algorithm consists only of basic operations, except for the arg min and arg max which we assume linear on the length of the corresponding array, it can be proven that its computational complexity is $O(n^2m)$.

Algorithm 1: First constructive phase: C1. C2 is obtained by removing the first for loop (lines 4-9).

1 $N^* \leftarrow N$;
2 $j_i \leftarrow \emptyset$;
3 $C_{\max} \leftarrow 0$; $C^i \leftarrow 0$;
4 **foreach** $i \in M$ **do**
5 | Let $j_i = \arg\max_{j \in N^*} r^P_{ij}$;
6 | Process j_i on machine i;
7 | $N^* \leftarrow N^* \setminus \{j_i\}$; $C^i \leftarrow p_{ij_i}$
8 | $C_{\max} \leftarrow \max\{C_{\max}, C^i\}$
9 **end**
10 **while** $N^* \neq \emptyset$ **do**
11 | **foreach** $j \in N^*$ **do**
12 | | **foreach** $i \in M$ **do**
13 | | | Find the new makespan in case job j is processed on machine i (considering the setup between j_i and j) denoted as $C^{ij} := \max\{C_{\max}, C^i + s_{ij_ij} + p_{ij}\}$;
14 | | **end**
15 | **end**
16 | $(i^*, j^*) = \arg\min_{i,j}\{C^{ij}\}$;
17 | Process j^* on i^* after job j_{i^*}, and update the makespan:
18 | $C^{i^*} \leftarrow C^{i^*j^*}$;
19 | $C_{\max} \leftarrow \max\{C^{i^*}, C_{\max}\}$;
20 | $N^* \leftarrow N^* \setminus \{j^*\}$;
21 | $j_{i^*} \leftarrow j^*$
22 **end**

4.1.2 Repairing phase

The repairing phase analyzes the sequence obtained in the constructive phase, and repairs such sequence in case the number of available resources is exceeded at some instant. The procedure is the following: if the number of available processing resources (setup resources) is exceeded at instant t, we check all machines that are processing jobs (doing setups) at the same instant t. In such case, we propose two different strategies to reduce the consumption of resources at instant t:

1. R1. In the first strategy, we delay the processing (setup) of one of

the jobs (setups) that is being processed (done) at instant t, until the other jobs (setups) are finished. We repeat this operation with all jobs (setups) that overlap at instant t, and we keep the delay that implies the lowest increase in the makespan. If more resources than available are still used, we repeat the same operation with the other jobs (setups).

2. In the second strategy, instead of delaying jobs (setups) until the end of the other jobs (setups), we postpone one of them one unit of time, and check if the number of resources used exceeds the availability on instant t. If the excess on instant t has been resolved, we check instant $t + 1$. We keep on doing that (one unit of time by one unit of time) until the use of resources is not exceeded. We repeat the same with all jobs (setups) that are processed (done) at instant t, and we keep the movement that implies the lowest increase in the makespan. If still more resources than available are used, we repeat the same operation with the other jobs (setups).

We repeat this process from $t = 0$ until the latest completion time. Algorithm 2 shows a pseudocode of these phases. The only difference between them is in the two *foreach* loops. In the pseudocode, M_t^P and M_t^S denote the set of machines that are processing a job at instant t, or doing a setup at instant t, respectively. R_t^P and R_t^S denote the number of resources devoted to jobs processing and the number of resources devoted to setups at instant t, respectively.

Algorithm 2: Repairing phases R1 and R2

$j_i \leftarrow \emptyset$;
$t \leftarrow 0$;
while $t \leq C_{\max}$ **do**
  if $R_t^P > R_{\max}^P$ **then**
    foreach $i \in M_t^P$ **do**
      Let j_i be the job being processed on i;
      R1: Compute the new makespan $C_{\max}^i$ if the processing of j_i was postponed until the other jobs whose processing overlaps with that of j_i are finished;
      R2: Compute the new makespan $C_{\max}^i$ if the processing of j_i is postponed one unit of time;
    end
    Let $i^* = \arg\min_{i \in M_t^P} C_{\max}^i$;
    Postpone the process at machine i and update $C_{\max}$;
  end
  if $R_t^S > R_{\max}^S$ **then**
    foreach $i \in M_t^S$ **do**
      R1: Compute the new makespan $C_{\max}^i$ if its setup was postponed until the other setups that overlap are finished;
      R2: Compute the new makespan $C_{\max}^i$ if its setup was postponed one unit of time;
    end
    Let $i^* = \arg\min_{i \in M_t^S} C_{\max}^i$;
    Postpone the setup at machine i and update $C_{\max}$;
  end
  Update $R_{t'}^P, R_{t'}^S, \ \forall\ t' \geq t$;
  $t \leftarrow t + 1$;
end

4.2 Randomization

As will be shown in the experiments section, the makespan obtained by the best of the heuristic strategies tested is more than 7% far from the solutions returned by the MILP, on average. In order to lower this difference, we also propose a randomization of the constructive phases in order to obtain a larger variety of solutions. The randomization strategy follows a standard

GRASP algorithm (see Feo and Resende (1989)). For this, we define the restricted candidate list (RCL), as the list of job-machine assignments that can be included in the solution. The process to build the RCL is as follows. We first sort all job-machine assignments in increasing order according to the makespan of the sequence if they were added. The first s elements in that list will constitute the RCL (the s assignments that would imply the lowest makespans). Then, instead of choosing the job-machine assignment that minimizes the makespan at each iteration (as proposed in the heuristics before), we randomly choose one of the job-machine assignments in the RCL, and remove the corresponding job from the list of pending jobs.

More specifically, line 16 in Algorithm 1 is substituted by the lines in Algorithm 3.

Algorithm 3: Randomization part of constructive phase, instead of line 16.

1 Sort $C^{ij}\ \forall\ i \in N^*$ such that $C^{i^1,j^1} \leq C^{i^2,j^2} \leq ... \leq C^{i^{|N^*|},j^{|N^*|}}$;
2 Define $RCL = \{(i,j) : C^{ij} \leq C^{i^s,j^s}\}$;
3 Select $(i^*, j^*) \in RCL$ chosen randomly;

The probability distribution chosen to select the elements in the RCL is a parameter of this algorithm, and will be tested in the experiments section. This process is repeated until the list of pending jobs is empty. (Note that, if RCL consists of only one element, $s = 1$, then we have the constructive phases in 1).

4.3 Local search

In order to avoid local optima, we also propose a local search phase. This phase consists of removing at random one of the assigned jobs on the makespan machine (the machine that finishes processing jobs last), and testing its reinsertion into all other possible positions on all machines. At each reinsertion, we compute the new makespan, and we keep the solution with the lowest makespan. We repeat this process until there are 50 consecutive iterations with no improvement in the makespan, like in Yepes et al. (2021). Algorithm 4 shows a pseudocode of this local search.

Algorithm 4: Summary of the local search.

```
1  BestSolution = C_max
2  NoImprovementIterations = 0
3  ImprovementFlag = 0
4  while NoImprovementIterations < 50 do
5      Remove at random one job from the makespan machine
6      foreach i ∈ M do
7          foreach k ∈ i do
8              Reinsert the job and compute the new makespan
9              if new makespan<BestSolution then
10                 BestSolution = new makespan
11                 ImprovementFlag = 1
12             end
13         end
14     end
15     NoImprovementIterations++
16     if ImprovementFlag= 1 then
17         NoImprovementIterations = 0
18         ImprovementFlag = 0
19     end
20 end
21 Return BestSolution
```

Afterwards, the repairing phase in Section 4.1.2 is applied, having this way a feasible solution. The same process is repeated until we run out of available time. The final solution returned by the algorithm is, among all the feasible solutions found, the one with minimum makespan. A high-level pseudocode of the metaheuristic is given in Algorithm 5.

Algorithm 5: Summary of the metaheuristic.

```
1 while TimeUsed < TimeAvailable do
2     Do constructive phase with |RCL| = s
3     Do Local search
4     Do repairing phase
5     Return a feasible solution
6 end
7 Return the feasible solution found with the lowest makespan.
```

The reader may note that this algorithm has several steps that influence

its performance. More specifically, we will calibrate:

- the RCL size, s.
- the maximum running time available ($TimeAvailable$).
- the probabilities for the job-machine assignments in the RCL of being chosen.
- the consideration of resources in the constructive phase.
- the effectiveness of the local search phase.

5 Experiments

In this section we show the results obtained by the GRASP we propose, over the randomly generated instances in Fanjul-Peyro (2020). These instances are divided into three sets: small, medium, and large. In the small set, the number of jobs varies in $n \in \{10, 20, 30, 40, 50\}$ and the number of machines varies in $m \in \{4, 6, 8\}$. Each of the $5 \times 3 = 15$ possible combinations of n and m is replicated five times as follows. Let $U(a, b)$ denote a discrete uniform probability distribution between a and b, with $a \leq b$. In each of the replicates, the processing time p_{ij} follows a $U(50, 100)$, the setup time s_{ijk} follows a $U(50, 100)$, the number of necessary resources for processing (r^P_{ij}) and for setups (r^S_{ijk}) both follow a $U(1, 9)$, $\forall\ i, j, k$. Finally, the number of available resources is set to $R^P_{\max} = R^S_{\max} = 5m$. Therefore, in total we have $5 \times 3 \times 5 = 75$ different instances in the small set. In the medium set, the number of jobs varies in $n \in \{60, 80, 100, 120, 140\}$. In the large set, the number of jobs varies in $n \in \{200, 250, 300, 350, 400\}$. The other input data varies as in the small set, having in total another 75 instances in each set. Therefore, in total we test $75 \times 3 = 225$ instances.

Our metaheuristics have been coded in Microsoft Visual Studio 2019 using C#, and run on virtual machines with 2 virtual processors and 8 gigabytes of RAM memory each under Windows 10 Enterprise 64 Bits. Complete results are available in the supplementary material accompanying this paper. C# codes are available from the authors upon request.

5.1 Reference solutions

In order to test the quality of our metaheuristics, we use the solutions in (Fanjul-Peyro, 2020), where an average 6.14% gap is reported for the small instances, 4.47% for the medium instances, and 2.26% for the large instances. These results were obtained on virtual machines with 2 cores, 16 GB or RAM memory, Windows 10 as operating system.

The quality measure we apply is the Relative Percent Deviation (RPD) with respect to the solutions reported in Fanjul-Peyro (2020). For each instance, and each algorithm, we define its RPD as:

$$RPD = 100\frac{z^M - z^F}{z^M}, \tag{1}$$

where z^M is the makespan of the solution returned by our metaheuristic, and z^F is the makespan of the solution in Fanjul-Peyro (2020). For the calibration of the heuristics over the small instance we will use the MILP. For the metaheursitics, since they are tested over all instances, we will use the Three-Phase exact algorithm. Note that, a positive value of RPD implies that the makespan of our metaheuristic is larger (worse) than the makespan of the other algorithm, whereas a negative value implies the opposite.

5.2 Calibration of heuristics over small set

In this section we show the results obtained by the four heuristic strategies tested: H1 (C1 + R1), H2 (C1 + R2), H3 (C2 + R1), H4 (C2 + R2). The RPDs of the heuristics are not satisfactory, and therefore we only report results over the small set in Table 2. We observe how the best results (lowest RPD) are found using strategy H4 (C2 + R2), which combines the second constructive phase with the second repairing phase.

We observe how the average RPD decreases with the size of the instance, which is typical in these problems as the quality of the solution returned by the MILP decreases. However, even if it yields the best results, the RPD returned by H4 is 7.52% on average, these deviations can be improved. In the next section we show the results obtained after a randomization of H4 in its constructive phase, which yields much lower relative percent deviations both in the small and the medium sets of instances.

n	H1	H2	H3	H4
10	21.21	11.30	9.06	8.45
20	18.15	12.06	9.95	9.86
30	17.62	11.68	8.94	9.44
40	12.80	9.43	6.56	6.55
50	8.46	5.67	3.08	3.02
Average	15.65	10.03	7.52	7.46

Table 2: Average RPD of the four heuristic strategies tested per group and overal average.

5.3 Calibration of GRASP over small and medium sets

In this section we analyze the results of a GRASP algorithm based on H4, as explained in Section 4.2. In the resulting metaheuristic, there are several factors that may influence the performance of the algorithm and we therefore need to calibrate them. Such factors, and the levels we test, are specified below.

- The first factor considered is the size of the restricted candidate list of the algorithm. We test three different values: $RCL \in \{2, 3, 4\}$. Note the abuse of notation when using RCL both to denote the restricted candidate list (a set) and its size (a number).

- The second factor considered is the allowed CPU time for the metaheuristic. We make such maximum time depend on the size of the instance, so larger instances are allowed more time than smaller instances, according to the following formula (in seconds):

$$t_{\max} = n \times w. \tag{2}$$

  We test five different levels: $w \in \{1, 2, 3, 4, 5\}$.

- The third factor tested is the probability distribution followed when choosing one element of the restricted candidate list. We test two different distributions: uniform, decreasing. The "uniform" distribution assigns equal probability to all elements in the RCL. The "decreasing" distribution assigns a different probability to each element of the RCL as follows: We sort the elements of the RCL according to the makespan

they would provoke, in an increasing way (first assignment is the best, ..., last assignment is the worst), and we give to each of them the probabilities as shown in Table 3, where s is the size of the RCL. Note that the assignment in position k has a probability of being chosen equal to $(s-k+1)\frac{2}{s(s+1)}$, and that the sum of these probabilities equals one. We denote this factor and its levels as $P \in \{U, D\}$ (U stands for "Uniform" and D stands for "Decreasing").

- The fourth factor is the criteria followed for choosing the "best" element in the RCL. In Yepes et al. (2020) it is empirically proven that considering the need of resources during the constructive phase yields better results than only considering the makespan. Therefore, we now propose to also consider the number of resources needed when choosing the best job-machine assignment, according to the following formula:

$$\alpha \times makespan + (1-\alpha) \times resources,\ \alpha \in [0,1]. \tag{3}$$

  In words, we sort the job-machine assignments according to a convex combination of the increase that such an assignment would produce in the makespan and the number of resources needed (processing plus setups) by the assignment. In Algorithm 1 (line 16), we would have $(i^*, j^*) = \arg\min_{i,j}\{\alpha C^{ij} + (1-\alpha)(r^P_{ij} + r^S_{ij_i j})\}$, instead of $(i^*, j^*) = \arg\min_{i,j}\{C^{ij}\}$. In order to calibrate the value of α, we test H4 over the 75 test instances in the small set, varying α between 0 and 1 with step 0.1. The results of this experiment are shown in Table 4.

  The reader may note that the lowest average RPD is achieved with $\alpha = 0.7$, improving from 7.52% (obtained when considering only the makespan, $\alpha = 1$) down to 6.97%.

  Therefore, the fourth factor considered consists of whether or not using information about resources during the constructive phase. This factor has two levels (YES and NO), that we denote $R \in \{Y, N\}$. YES implies that the resources are considered in the constructive phase with $\alpha = 0.7$, and NO implies that the resources are not considered in the constructive phase, which implies $\alpha = 1$.

- The fifth factor considered consists of whether or not using the local search after the constructive phase. As before, this factor has two levels (YES and NO), that we denote $LS \in \{Y, N\}$.

To summarize, we have the following factors with its levels: $RCL \in \{2, 3, 4\}, w \in \{1, ..., 5\}, P \in \{U, D\}, R \in \{Y, N\}$, $LS \in \{Y, N\}$.

In order to find the best combination of factors, we run a full factorial design testing all possible combinations: $3 \times 5 \times 2 \times 2 \times 2 = 120$. We will test all possible combinations over the 150 instances in the small and medium sets. Therefore, in this part of the experiments we run $120 \times 150 = 18000$ times our algorithms. Note that this implies a total of 3510000 seconds of computation time, which means around 41 days of CPU time to complete the calibration. The two best combinations of factors will be further tested over the large set of instances (see Section 5.4). For this set of small instances, the MILP finds solutions in 180 minutes which are not always optimal, yielding an average gap of 5.31% (Table 2 in Fanjul-Peyro (2020)). For some groups of these instances, our algorithm finds solutions which are, on average, better than the MILP. This is why in the following tables we observe some negative relative percent deviations (RPD).

We now present the results of all 120 combinations, divided into four groups, according to factors P and R, in Tables 5 to 8. In the four tables, we test each combination of the other three factors: RCL, w and LS . For each combination of factors, we show the average RPD per group of instances according to the number of jobs n, and the global average per set. Note that a negative value implies that the GRASP obtains better objective values, on average, for that group. For the sake of space, we now present the results for both the small and medium instances in one single table.

Table 5 presents the results for the metaheuristic in which elements of the RCL are chosen following a uniform distribution ($P = U$) and the resources are not considered in the constructive phase ($R = N$). We observe how the best results in small instances are obtained for the largest size of the RCL, while for medium instances the best results are obtained with $RCL = 2$. In both sizes we observe that the algorithm yields better results with local search. We also observe how the average RPD is best when allowing the algorithm the maximum time ($t = n \times 5$). It is also interesting to note that the RDP is a little lower when the size of the instance increases, meaning that the quality of the solution returned by the metaheursitics does not get worse when the problem gets more complicated.

Table 6 presents the results for the metaheuristic in which the elements of the RCL are chosen following a decreasing distribution ($P = D$) and the resources are not considered in the constructive phase ($R = N$). We observe how the best results in both groups of instances are obtained for RCL = 4.

Position in RCL	1	2	...	s
Probability	$s\frac{2}{s(s+1)}$	$(s-1)\frac{2}{s(s+1)}$	...	$\frac{2}{s(s+1)}$

Table 3: Probabilities of the distribution $P = D$.

α	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
Avg. RPD	50.81	17.84	12.29	9.31	7.49	7.68	7.24	6.97	7.26	7.42	7.52

Table 4: Average RPD for H4, considering different values of α.

			Small						Medium					
w	RCL	LS	10	20	30	40	50	Avg.	60	80	100	120	140	Avg.
1	2	Yes	4.07	5.12	3.78	2.95	0.02	3.19	2.92	2.67	2.32	2.44	2.03	2.48
		No	5.57	7.65	5.32	4.45	0.76	4.75	3.50	2.65	2.61	2.46	2.23	2.69
	3	Yes	2.96	4.66	3.64	2.53	-0.26	2.71	2.84	2.60	2.41	2.54	2.27	2.53
		No	5.20	6.82	4.63	3.49	0.27	4.08	3.21	2.81	2.52	2.57	2.44	2.71
	4	Yes	2.93	4.45	3.67	2.39	-0.16	2.66	2.73	2.59	2.49	2.50	2.37	2.54
		No	4.46	5.79	4.40	3.12	0.15	3.59	3.09	2.77	2.72	2.63	2.72	2.79
2	2	Yes	4.07	4.97	3.87	2.79	0.03	3.15	2.85	2.62	2.22	2.38	1.98	2.41
		No	5.54	7.65	5.32	4.42	0.76	4.74	3.50	2.82	2.59	2.44	2.16	2.70
	3	Yes	2.69	4.66	3.61	2.45	-0.21	2.64	2.81	2.62	2.39	2.41	2.21	2.49
		No	4.98	6.73	4.63	3.49	0.27	4.02	3.12	2.79	2.38	2.48	2.37	2.63
	4	Yes	2.78	4.39	3.62	2.26	-0.35	2.54	2.69	2.60	2.44	2.48	2.34	2.51
		No	4.41	5.79	4.40	3.12	0.15	3.58	3.08	2.76	2.53	2.50	2.50	2.67
3	2	Yes	4.07	4.91	3.81	2.72	-0.06	3.09	2.77	2.55	2.15	2.28	1.95	2.34
		No	5.45	7.65	5.32	4.45	0.69	4.71	3.37	2.75	2.53	2.42	2.05	2.62
	3	Yes	2.69	4.56	3.61	2.44	-0.25	2.61	2.79	2.59	2.21	2.41	2.04	2.41
		No	5.00	6.73	4.63	3.44	0.23	4.01	3.12	2.67	2.43	2.43	2.33	2.60
	4	Yes	2.61	4.16	3.62	2.26	-0.33	2.46	2.69	2.57	2.43	2.39	2.39	2.49
		No	4.10	5.79	4.40	3.12	0.15	3.51	3.09	2.63	2.48	2.42	2.42	2.61
4	2	Yes	3.93	5.05	3.76	2.75	-0.03	3.09	2.86	2.53	2.19	2.33	1.84	2.35
		No	5.45	7.65	5.32	4.29	0.65	4.67	3.49	2.61	2.52	2.38	2.06	2.61
	3	Yes	2.81	4.56	3.56	2.40	-0.27	2.61	2.83	2.55	2.26	2.36	2.13	2.43
		No	4.93	6.73	4.63	3.49	0.20	3.99	3.15	2.66	2.30	2.38	2.21	2.54
	4	Yes	2.14	4.43	3.61	2.21	-0.23	2.43	2.60	2.54	2.40	2.36	2.33	2.44
		No	3.95	5.79	4.40	3.12	0.15	3.48	3.09	2.68	2.47	2.40	2.48	2.62
5	2	Yes	3.93	4.88	3.87	2.68	-0.09	3.05	2.82	2.45	2.11	2.21	1.81	2.28
		No	5.45	7.65	5.32	4.34	0.65	4.68	3.42	2.62	2.35	2.33	2.00	2.54
	3	Yes	2.78	4.66	3.56	2.44	-0.25	2.64	2.80	2.52	2.21	2.31	2.02	2.37
		No	4.70	6.73	4.63	3.35	0.08	3.90	3.21	2.61	2.27	2.39	2.19	2.53
	4	Yes	2.13	4.16	3.61	2.25	-0.38	2.35	2.67	2.50	2.30	2.39	2.22	2.41
		No	4.18	5.81	4.40	3.12	0.09	3.52	3.09	2.53	2.37	2.44	2.40	2.56

Table 5: Average results for $P = U, R = N$.

We also observe that the effect of CPU time available on the average RPD is less significant and there are small differences between the results obtained with $w = 4$ and $w = 5$. As in the previous table, the best results are given when there is local search in the algorithm.

			Small						Medium					
w	RCL	LS	10	20	30	40	50	Avg.	60	80	100	120	140	Avg.
1	2	Yes	4.04	5.07	3.89	3.27	-0.01	3.25	2.75	2.55	2.38	2.32	1.86	2.37
		No	5.54	7.65	5.32	4.38	0.74	4.73	3.45	2.78	2.59	2.46	2.10	2.68
	3	Yes	3.83	4.97	3.31	2.78	-0.25	2.93	2.85	2.49	2.30	2.25	2.01	2.38
		No	5.21	6.82	4.63	3.49	0.27	4.08	3.16	2.69	2.50	2.45	2.12	2.58
	4	Yes	3.13	5.04	3.24	2.53	-0.31	2.72	2.78	2.58	2.37	2.40	2.25	2.48
		No	4.71	6.15	4.42	3.12	0.15	3.71	3.09	2.72	2.57	2.45	2.33	2.63
2	2	Yes	3.91	5.56	3.89	3.20	-0.06	3.30	2.76	2.48	2.30	2.28	1.79	2.32
		No	5.45	7.65	5.32	4.38	0.73	4.71	3.32	2.70	2.55	2.31	2.04	2.58
	3	Yes	2.98	5.00	3.29	2.55	-0.27	2.71	2.86	2.47	2.25	2.17	2.00	2.35
		No	4.70	6.89	4.63	3.46	0.25	3.98	3.21	2.69	2.46	2.40	2.10	2.57
	4	Yes	2.78	4.85	3.05	2.56	-0.33	2.58	2.73	2.47	2.22	2.30	2.01	2.35
		No	4.30	5.79	4.40	3.02	0.15	3.53	2.99	2.63	2.41	2.34	2.17	2.51
3	2	Yes	4.01	5.07	3.87	3.13	-0.16	3.19	2.61	2.46	2.33	2.25	1.82	2.29
		No	5.54	7.64	5.32	4.35	0.69	4.71	3.49	2.57	2.45	2.27	1.94	2.54
	3	Yes	2.94	4.85	3.31	2.58	-0.21	2.69	2.86	2.46	2.19	2.13	1.81	2.29
		No	4.70	6.73	4.63	3.38	0.25	3.94	3.17	2.63	2.35	2.35	1.94	2.49
	4	Yes	2.28	4.50	3.08	2.47	-0.35	2.40	2.69	2.41	2.14	2.18	2.12	2.31
		No	4.00	5.79	4.40	3.02	0.05	3.45	3.07	2.64	2.44	2.30	2.16	2.52
4	2	Yes	3.91	5.07	3.87	3.05	-0.14	3.15	2.76	2.36	2.25	2.09	1.77	2.26
		No	5.45	7.64	5.32	4.35	0.57	4.67	3.36	2.51	2.54	2.33	1.94	2.54
	3	Yes	2.82	4.90	3.26	2.54	-0.27	2.65	2.84	2.37	2.08	2.09	1.76	2.23
		No	4.72	6.73	4.63	3.37	0.14	3.92	3.21	2.49	2.30	2.33	1.97	2.46
	4	Yes	2.05	4.46	3.05	2.21	-0.39	2.28	2.72	2.41	2.08	2.16	1.99	2.27
		No	4.02	5.81	4.40	3.10	0.11	3.49	3.06	2.64	2.36	2.21	2.07	2.47
5	2	Yes	3.91	5.07	3.83	3.09	-0.14	3.15	2.74	2.38	2.20	2.18	1.79	2.26
		No	5.54	7.65	5.32	4.44	0.68	4.73	3.27	2.55	2.35	2.31	1.87	2.47
	3	Yes	2.79	4.85	3.29	2.51	-0.27	2.63	2.83	2.47	2.07	2.19	1.78	2.27
		No	4.72	6.73	4.63	3.45	0.22	3.95	3.14	2.54	2.36	2.27	1.96	2.46
	4	Yes	2.00	4.46	3.05	2.35	-0.40	2.29	2.68	2.43	2.02	2.14	2.01	2.25
		No	3.95	5.79	4.40	3.12	0.08	3.47	3.09	2.49	2.34	2.23	2.05	2.44

Table 6: Average results for $P = D, R = N$.

Table 7 presents the results for the metaheuristic in which the elements of the RCL are chosen following a uniform distribution ($P = U$) and the resources are considered in the constructive phase ($R = Y$). We observe that in small instances, the best results are given for the largest RCL size, while in

medium instances the best results are given for RCL=3. As in the previous table, we observe that the effect of CPU time available on the average RPD is less significant and there are no big differences between the results obtained with $w = 4$ and $w = 5$. Once again, the algorithm with local search yields better results.

			Small						Medium					
w	RCL	LS	10	20	30	40	50	Avg.	60	80	100	120	140	Avg.
1	2	Yes	3.78	5.82	3.76	2.42	0.01	3.16	3.13	2.71	2.22	2.37	2.10	2.51
		No	4.72	7.72	5.01	3.34	0.43	4.24	3.40	2.84	2.35	2.56	2.12	2.66
	3	Yes	3.40	5.40	3.56	2.37	-0.42	2.86	2.98	2.55	2.19	2.51	2.27	2.50
		No	5.42	6.94	4.42	3.15	-0.10	3.97	3.21	2.79	2.36	2.58	2.37	2.66
	4	Yes	2.62	5.09	3.45	2.37	-0.44	2.62	2.92	2.61	2.30	2.73	2.52	2.61
		No	5.01	6.41	4.34	2.85	-0.08	3.70	3.03	2.71	2.58	2.79	2.64	2.75
2	2	Yes	3.28	5.57	3.84	2.41	-0.08	3.00	3.05	2.65	2.16	2.32	2.05	2.45
		No	4.72	7.72	5.01	3.34	0.42	4.24	3.38	2.67	2.42	2.44	2.05	2.59
	3	Yes	3.47	5.37	3.36	2.22	-0.42	2.80	2.82	2.50	2.13	2.48	2.22	2.43
		No	4.17	6.94	4.42	3.12	-0.07	3.72	3.22	2.65	2.34	2.42	2.30	2.59
	4	Yes	2.33	5.09	3.33	2.35	-0.47	2.52	2.90	2.48	2.31	2.53	2.52	2.55
		No	3.92	6.36	4.34	2.81	-0.08	3.47	3.03	2.69	2.38	2.67	2.42	2.64
3	2	Yes	3.38	5.68	3.82	2.36	-0.12	3.02	3.11	2.51	2.07	2.36	1.93	2.40
		No	4.72	7.72	5.01	3.34	0.43	4.24	3.34	2.65	2.30	2.29	2.14	2.54
	3	Yes	2.89	5.26	3.32	2.30	-0.42	2.67	2.95	2.44	2.06	2.47	2.16	2.42
		No	4.34	6.94	4.42	2.90	-0.11	3.70	3.04	2.62	2.31	2.42	2.23	2.53
	4	Yes	2.14	4.96	3.21	2.36	-0.48	2.44	2.84	2.45	2.28	2.43	2.24	2.45
		No	4.33	6.36	4.34	2.81	-0.08	3.55	2.99	2.60	2.45	2.53	2.38	2.59
4	2	Yes	3.32	5.57	3.79	2.25	-0.10	2.96	3.05	2.46	2.03	2.28	1.94	2.36
		No	4.72	7.72	5.01	3.33	0.41	4.24	3.33	2.84	2.24	2.32	1.99	2.55
	3	Yes	3.10	5.20	3.32	2.22	-0.53	2.66	2.90	2.39	2.14	2.33	2.21	2.39
		No	4.29	6.94	4.42	3.03	-0.14	3.71	3.07	2.58	2.20	2.43	2.17	2.49
	4	Yes	2.20	4.79	3.21	2.36	-0.48	2.41	2.82	2.43	2.21	2.53	2.31	2.46
		No	4.11	6.36	4.28	2.81	-0.08	3.49	3.03	2.56	2.41	2.54	2.38	2.58
5	2	Yes	3.28	5.57	3.79	2.33	-0.10	2.97	3.08	2.46	2.07	2.28	1.94	2.36
		No	4.72	7.72	5.01	3.32	0.30	4.21	3.33	2.75	2.27	2.35	1.97	2.53
	3	Yes	3.45	5.20	3.32	2.31	-0.42	2.77	2.92	2.37	2.04	2.29	2.11	2.35
		No	4.17	6.94	4.42	3.03	-0.06	3.70	3.12	2.59	2.26	2.19	2.14	2.46
	4	Yes	2.03	4.89	3.21	2.36	-0.50	2.40	2.79	2.40	2.01	2.51	2.24	2.39
		No	3.92	6.36	4.28	2.81	-0.13	3.44	2.99	2.48	2.40	2.45	2.33	2.53

Table 7: Average results for $P = U, R = Y$.

Table 8 presents the results for the metaheuristic in which elements of the RCL are chosen following a decreasing distribution ($P = D$) and the resources are considered in the constructive phase ($R = Y$). Like in the other

three combinations, the best average RPD is obtained with the algorithm with the local search. In this combination we observe that in both groups of instances the best average RPD is obtained with RCL=4. As in the previous tables, we do not observe big differences among average RPD with $w = 4$ and $w = 5$.

			Small						Medium					
w	RCL	LS	10	20	30	40	50	Avg.	60	80	100	120	140	Avg.
1	2	Yes	2.92	5.72	3.47	2.75	-0.14	2.94	2.91	2.64	2.23	2.32	1.98	2.42
		No	4.94	7.79	5.01	3.34	0.43	4.30	3.55	2.88	2.30	2.41	2.14	2.66
	3	Yes	3.19	5.17	3.23	2.43	-0.40	2.72	3.01	2.64	2.29	2.46	2.09	2.50
		No	4.40	6.94	4.48	3.01	-0.06	3.75	3.14	2.59	2.32	2.48	2.12	2.53
	4	Yes	2.71	5.25	3.19	2.33	-0.43	2.61	2.68	2.48	2.26	2.41	2.29	2.42
		No	4.36	6.46	4.28	2.80	-0.06	3.57	3.09	2.66	2.29	2.44	2.24	2.55
2	2	Yes	2.92	5.59	3.47	2.52	-0.25	2.85	2.85	2.55	2.08	2.34	1.85	2.34
		No	4.72	7.72	5.01	3.34	0.41	4.24	3.52	2.79	2.22	2.32	1.94	2.56
	3	Yes	2.87	5.19	3.36	2.42	-0.42	2.68	2.83	2.61	2.12	2.37	2.02	2.39
		No	4.17	6.96	4.48	3.12	-0.06	3.73	3.16	2.57	2.24	2.24	1.98	2.44
	4	Yes	2.89	5.02	3.27	2.26	-0.44	2.60	2.62	2.48	2.18	2.20	2.15	2.32
		No	4.46	6.36	4.34	2.81	-0.16	3.56	3.02	2.58	2.15	2.37	2.16	2.46
3	2	Yes	3.19	5.55	3.47	2.51	-0.25	2.89	2.84	2.56	2.14	2.32	1.90	2.35
		No	4.72	7.79	5.01	3.30	0.40	4.24	3.48	2.56	2.19	2.27	1.98	2.50
	3	Yes	2.58	5.33	3.13	2.42	-0.52	2.59	2.85	2.48	2.08	2.25	1.96	2.32
		No	4.17	6.66	4.42	3.03	-0.06	3.64	3.12	2.62	2.21	2.29	2.09	2.46
	4	Yes	2.73	4.73	3.27	2.29	-0.43	2.52	2.57	2.38	2.11	2.29	2.05	2.28
		No	3.92	6.36	4.28	2.80	-0.09	3.45	3.02	2.55	2.27	2.36	2.12	2.46
4	2	Yes	2.92	5.43	3.47	2.50	-0.33	2.80	2.81	2.46	2.10	2.33	1.77	2.29
		No	4.72	7.72	5.01	3.34	0.43	4.24	3.11	2.77	2.31	2.24	1.97	2.48
	3	Yes	2.58	5.28	3.11	2.36	-0.52	2.56	2.82	2.47	2.09	2.26	1.94	2.32
		No	4.17	6.94	4.42	3.03	-0.06	3.70	3.21	2.46	2.24	2.30	1.88	2.42
	4	Yes	2.20	4.75	3.25	2.31	-0.43	2.42	2.58	2.36	2.11	2.20	2.03	2.26
		No	3.92	6.36	4.34	2.81	-0.12	3.46	2.88	2.57	2.24	2.21	2.11	2.40
5	2	Yes	2.92	5.43	3.47	2.49	-0.24	2.82	2.76	2.55	2.05	2.22	1.88	2.29
		No	4.72	7.72	5.01	3.16	0.33	4.19	3.31	2.62	2.31	2.20	1.84	2.46
	3	Yes	2.30	5.14	3.12	2.40	-0.53	2.49	2.74	2.50	2.19	2.20	1.93	2.31
		No	4.17	6.94	4.40	3.03	-0.10	3.69	3.14	2.47	2.15	2.22	1.89	2.37
	4	Yes	2.55	4.71	3.21	2.11	-0.44	2.43	2.59	2.39	2.09	2.20	2.01	2.26
		No	3.92	6.36	4.28	2.79	-0.17	3.43	2.91	2.51	2.17	2.27	2.12	2.39

Table 8: Average results for $P = D, R = Y$.

Note that the lowest average RPD in the small instances is achieved for $P = D, R = N, w = 4, RCL = 4$, $LS = Y$, and is equal to 2.28%, whereas for the medium set of instances the best average is found for $P = D, R = Y, w =$

Factor	$p-value$ Small	$p-value$ Medium
P	0.392	0
R	9.17e-11	0.0734
w	5.56e-05	0
RCL	0	0.0632
LS	0	0

Table 9: $p-values$ for the ANOVA analysis. 0 stands for a value less than 2×10^{-16}.

$5, RCL = 4$, $LS = Y$, and is equal to 2.26%.

5.3.1 ANOVA

In order to analyze if the effects of the factors analyzed over the RPD are significant, or on the contrary, the differences observed in the previous tables are due to chance, we perform an Analysis of Variance (ANOVA), considering the RPD as the response variable, and the five factors analyzed (taking into account that the instances are fixed for all combinations of factors). We consider a standard level of significance 0.05. We perform one different analysis for the small set and another for the medium set. The p-values obtained for each individual factor are summarized in Table 9. In small instances, factors R, w, RCL and LS are significant, while in the group of medium instances the factors P, w, and LS have a significant impact on the RPD.

In order to find which factor levels would be optimal for our algorithms, we also compute the averages per level, and the Tukey Honest Significant Difference (HSD) intervals for those factors with more than two levels. Tukey HSD intervals are confidence intervals on the differences between the means of the factor levels. Let μ_1 and μ_2 be the averages of a given variable, for factor level 1 and factor level 2, respectively. If the Tukey HSD interval for $\mu_1 - \mu_2$ is completely positive, then level 1 yields significantly larger results than level 2. Analogously, if the interval is completely negative then level 2 yields significantly larger results than level 1. If the interval contains number zero, then no significant differences are observed between the two averages.

- Factor P. For the small instances, the average RPD for $P = D$ is 3.33, and for $P = U$ is 3.35. For the medium instances, the average

RPD for $P = D$ is 2.41, and for $P = U$ is 2.52. Since this factor is significant in medium instances, we conclude that assigning probabilities in a decreasing way ($P = D$) yields significantly better results than assigning probabilities uniformly ($P = U$), when selecting the elements in the Restricted Candidate List in medium instances. In small instances, this factor is not significant, but the average RPD is also better with $P = D$.

- Factor R. For the small instances, the average RPD for $R = N$ is 3.42, and for $R = Y$ is 3.26. For the medium instances, the average RPD for $R = N$ is 2.47, and for $R = Y$ is 2.45. We observe that in both sets of instances, the average RPD is better if we take into account the resources during the constructive phase. However, the best average RPD obtained in the small set of instances is with the factor $R = N$ (Table 6). These contradictory results motivate that this factor will be further analyzed over the large set of instances in Section 5.4.

- Factor w. The average RPD for the different values of this factor are given in Table 10.

  We observe that the differences between one level of w and the next are smaller and smaller when w increases, both for the small and the medium instances. In small instances, the Tukey HSD interval for the difference in average RPD between levels $w = 2$ and $w = 5$ is (-0.187, 0.0220). Since 0 is contained in that interval, we conclude that there are no significant differences between these two levels of this factor. In the medium set, the Tukey HSD interval for the difference in average RPD between levels $w = 3$ and $w = 5$ is (-0.058, 0.006). Since 0 is contained in that interval, we conclude that there are no significant differences between these two levels of this factor. Therefore, considering levels $w = 3$, $w = 4$ or $w = 5$ seem to yield the same results, statistically speaking, and better than the results for $w \leq 2$.

- Factor RCL. The average RPD for the different values of this factor are given in Table 11.

  For this factor, in small instances, none of the Tukey HSD intervals contains 0, and therefore we conclude that setting a Restricted Candidate List with four elements yields significantly better results than the other two sizes analyzed (since this level provides the lowest sample average).

In medium instances, we observe slightly better results with $RCL = 3$, although the ANOVA showed no significant differences (p-value = 0.0632).

- Factor LS. The average RPD for the two levels of this factor are given in Table 12.

  This factor is significant in both sets of instances, where the best average RPD is obtained with local search.

From the experiments, it seems clear that the algorithm must have local search, and the size of the RCL in small instances should be 4. In the medium instances set the size of the RCL is not so clear as the factor is not significant, however, we will take RCL = 4 since it is the value in which the best average RPD was found for this group of instances (with $P = D$, $R = Y$ and $w = 5$). Since for factor w there are no significant differences among levels $3, 4, 5$, we will take $w = 3$ because CPU times are lower. Finally, factor P is set to D (decreasing probabilities), since with this configuration the algorithm yields better results.

5.4 Calibration of metaheuristics over the large set

In this section we show the results over the 75 instances of the large set of our metaheursitic. For this set of instances we set $RCL = 4$, $P = D$, and $LS = Y$. As we mentioned before, factor R took different optimal levels for small and medium instances, and therefore we test both $R = Y$ and $R = N$ for the large instances. Finally, although the best average RPD (in both small and medium instances) is obtained with $w = 5$, there are no significant differences among $w = 3$, $w = 4$ and $w = 5$ in medium instances. Therefore, for this factor we set $w = 3$ (the lowest CPU time). Table 13 shows the average results per group of instances according to n, as well as the global average, for the two levels of factor R. In both cases, we compare with the best known solution (BKS), found by the TPhA in Fanjul-Peyro (2020), and with the lower bound (LB) obtained by the same algorithm. The comparison with the LB is needed since in some instances the solution returned by TPhA is not guaranteed to be optimal. We observe that in large instances the best results are obtained with $R = N$. We confirm that statement with a $t - test$, which shows statistically significant impact of factor R with $p - value = 0.00014$. In short, the GRASP yields feasible solutions which are very close to the

Level	Average RPD Small	Average RPD Medium
$w = 1$	3.45	2.56
$w = 2$	3.37	2.49
$w = 3$	3.32	2.45
$w = 4$	3.29	2.42
$w = 5$	3.28	2.39

Table 10: Average RPD for different maximum CPU time allowed.

Level	Average RPD Small	Average RPD Medium
$RCL = 2$	3.76	2.46
$RCL = 3$	3.27	2.45
$RCL = 4$	3.00	2.48

Table 11: Average RPD for different sizes of RCL.

Level	Average RPD Small	Average RPD Medium
$LS = Y$	2.73	2.38
$LS = N$	3.95	2.55

Table 12: Average RPD for different levels of LS.

best known solution (1% on average) and to the best lower bounds (3% on average), in short CPU times.

Reference	n	200	250	300	350	400	Avg.
BKS	$R = Y$	1.67	1.99	1.59	1.45	1.61	1.66
	$R = N$	1.68	2.00	1.58	1.36	1.50	1.62
LB	$R = Y$	4.34	4.33	3.79	3.43	3.34	3.85
	$R = N$	4.35	4.33	3.78	3.34	3.23	3.81

Table 13: Evolution of average RPD with number of jobs for large instances, fixing $P = D$, $w = 3$ (CPU time = $3n$ seconds), and $RCL = 4$.

6 Managerial insights

In a real setting, assessing and comparing algorithms is a multicriteria matter. Practitioners would like to have good performance in:

- quality of solution (RPD). TPhA is a brilliant approach that proven successful in Fanjul-Peyro (2020). Our aim is not to compete with that algorithm, but rather to use its solutions as a basis for assessing the quality of our metaheuristic. The metaheuristic we propose has slightly larger GAPs than TPhA in instances where the solver in TPhA can actually perform well, see Table 13.

- computational effort (CPU time). As we have seen, our algorithm finds solutions which are very close to those provided by the TPhA, using a few seconds (our algorithm) instead of 30 minutes (TPhA). Figure 2 shows that, in the set of large instances, after only 10 seconds our algorithm finds solutions which are around 2.1% far from the TPhA, getting as close as 1.85% after only 60 seconds, and 1.80% after 120 seconds. In production settings where time to react is essential, the speed for having a new production plan compensates the (low) reduction in solution quality.

- how it escalates (maximum size that an algorithm can solve). Even the best commercial solvers explode when the size of the model increases up to some extent. On the other hand, metaheuristics do escalate better, and CPU times do not increase as much. We have solved instances

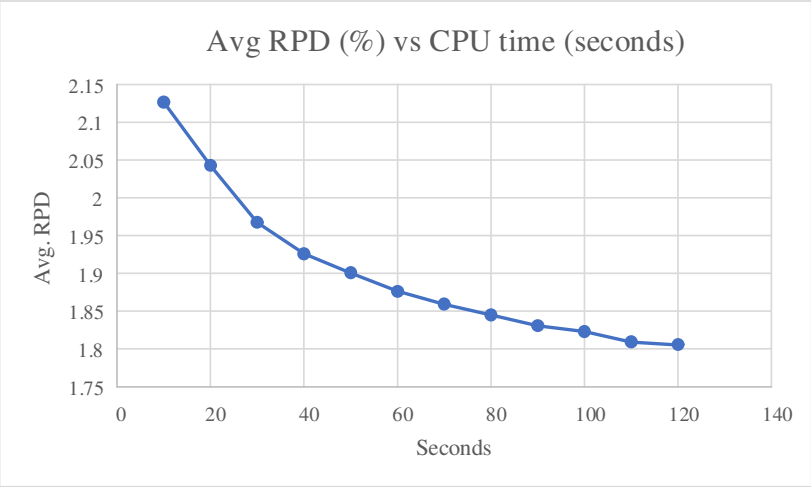

Figure 2: Evolution of RPD with time, for the set of larger instances.
Figure 2 Alt Text: For every ten seconds, the average RPD of the best solution found so far over all instances is computed, which drops from around 2.1% after 10 seconds, to around 1.8% after two minutes.

with 700 jobs and 8 machines, 1000 jobs and 8 machines, 1300 jobs and 5 machines, and 2000 jobs and 5 machines. In all of them our algorithm finds feasible solutions after seconds. However, even the best commercial solver would run out of memory if trying to solve such instances.

- monetary cost (price of the final decision tool to be installed in the company). Not all companies can afford a commercial solver, and for these companies, which typically are small and medium-sized enterprises (SMEs), other alternative decision tools should be provided.

Our metaheuristic is an improvement over the existing literature in three of the four criteria aforementioned: it rapidly provides feasible good-quality solutions, it can find solutions to larger instances, it is cheaper (no need to buy a commercial solver) than algorithms relying on a commercial solver. With respect to the quality of solution, it goes without saying that the lower the gap with respect to the optimal solution, the better. However, it often happens that a planner would prefer a planning sequence in a few seconds, which is expected to be close to the optimal, than having the actual optimal if for this they need to wait for much longer. Can a production plant be

paralyzed for so long? In some contexts that is not possible, and in case of a disruption, a new production plan needs to be designed as fast as possible.

7 Conclusions and future work

In this paper we have addressed an unrelated parallel machines scheduling problem, in which both the processing of the jobs and the setups of the machines require additional resources. We have proposed different heuristic strategies, based on constructing a sequence of jobs for each machine, and then repairing it in case more resources than available are required. Such heuristic strategies have been randomized following a GRASP strategy. The resulting metaheuristic has been tested over a number of randomly generated instances, and compared with the solutions obtained by an exact algorithm proposed in an earlier research. Different factors that might affect the performance of the metaheuristic have been tested statistically.

In short, the main conclusions we obtain from the experiments we have carried out are:

- Randomizing the heuristic greatly improves the quality of the solution returned. Roughly speaking, the improvement in average RPD is from 7% of the heuristic to 2% in the metaheuristic in small instances.

- Allowing more CPU time also improves the solution returned, as expected. However, it seems that after some time the improvements are not significant for the medium and large instances. Therefore, we would recommend running the GRASP for $3n$ seconds, as it gives the same quality of solution (statistically speaking) as $5n$ seconds. As expected, the maximum CPU time allowed should depend on the size of the instance, whenever this is possible.

- Giving larger probability of being chosen to the assignments in the RCL causing lower increases in the makespan (factor $P = D$), also improves the performance of the algorithm.

- Considering the resources during the constructive phase (factor $R = Y$) makes the algorithm slower. However, for small instances this reduction in speed is compensated as the quality of the solutions returned is better when considering the resources. On the contrary, for the medium and

large instances this improvement is not observed. An explanation for this is that for the medium and large instances, including the resources in the constructive phase allows for a too low variety of solutions (as the computational effort is too large).

- Adding a local search to the algorithm significantly improves the quality of the solutions.

The metaheuristic proposed, with appropriate tuning of the aforementioned parameters, returns solutions which are on average around 1.6% far from the solutions returned by the exact three phase algorithm in Fanjul-Peyro (2020).

Further research on this topic may focus on different directions: improving metaheuristics, analyzing multiobjective and/or stochastic versions of this problem, etc.

Acknowledgements

The authors would like to acknowledge the support from "Ministerio de Ciencia, Innovación y Universidades" under grant "PID2020-114594G Optimization on data science and network design problems: large scale network mdoels meeto optimization and data science tools", from the "Junta de Andalucía" under grant "AT21_00032 Optimización Aplicada al Tejido Productivo Andaluz", and from the "Generalitat Valenciana" under grant "NUevos REtos en SEcuenciación" (No. AICO/2020/049). Juan C. Yepes-Borrero acknowledges financial support by "Colfuturo" under program "Crédito-Beca" grant number 201503877 and from "El Instituto Colombiano de Crédito Educativo y Estudios Técnicos en el Exterior – ICETEX" under program "Pasaporte a la ciencia – Doctorado, Foco-reto país 4.2.3", grant number 3568118.

Finally, special thanks are due to Luis Fanjul-Peyro, who kindly provided us with the solutions to the benchmark, which have been used for assessing the quality of our algorithm.

Data availability statement

The authors confirm that the data supporting the findings of this study are available within the article [and/or] its supplementary materials.

References

Abdeljaoued, M., Saadani, N., and Bahroun, Z. (2020). Heuristic and metaheuristic approaches for parallel machine scheduling under resource constraints. *Operational Research*, 20(4):2109–2132.

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

Avalos-Rosales, O., Angel-Bello, F., and Alvarez, A. (2015). Efficient metaheuristic algorithm and re-formulations for the unrelated parallel machine scheduling problem with sequence and machine-dependent setup times. *International Journal of Advanced Manufacturing Technology*, 76(9-12):1705–1718.

Çanakoglu, E. and Muter, I. (2020). Identical parallel machine scheduling with discrete additional resource and an application in audit scheduling. *International Journal of Production Research*, 59(17):5321–5336.

Chen, L., Ye, D., and Zhang, G. (2018). Parallel machine scheduling with speed-up resources. *European Journal of Operational Research*, 268(1):101–112.

Diana, R. O. M., de Franca Filho, M. F., de Souza, S. R., and de Almeida Vitor, J. F. (2014). An immune-inspired algorithm for an unrelated parallel machines' scheduling problem with sequence and machine dependent setup-times for makespan minimisation. *Neurocomputing*, 163:94–105.

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

Fanjul-Peyro, L. (2020). Models and an exact method for the unrelated parallel machine scheduling problem with setups and resources. *Expert Systems with Applications: X*, 5:100022.

Fanjul-Peyro, L., Perea, F., and 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. and Ruiz, R. (2010). Iterated greedy local search methods for unrelated parallel machine scheduling. *European Journal of Operational Research*, 207(1):55–69.

Fanjul-Peyro, L. and Ruiz, R. (2011). Size-reduction heuristics for the unrelated parallel machines scheduling problem. *Computers and Operations Research*, 38(1):301–309.

Fanjul-Peyro, L. and Ruiz, R. (2012). Scheduling unrelated parallel machines with optional machines and jobs selection. *Computers and Operations Research*, 39(7):1745–1753.

Fanjul-Peyro, L., Ruiz, R., and Perea, F. (2019). Reformulations and an exact algorithm for unrelated parallel machine scheduling problems with setup times. *Computers and Operations Research*, 101:173–182.

Feo, T. A. and Resende, M. G. (1989). A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8(2):67 – 71.

Fleszar, K. and Hindi, K. (2018). Algorithms for the unrelated parallel machine scheduling problem with a resource constraint. *European Journal of Operational Research*, 271(3):839–848.

Fu, Y., Jiang, G., Tian, G., and Wang, Z. (2019). Job scheduling and resource allocation in parallel-machine system via a hybrid nested partition method. *IEEJ Transactions on Electrical and Electronic Engineering*, 14(4):597–604.

Garey, M. and Johnson, D. (1979). *Computers and Intractability: A Guide to the Theory of NP completeness*. W. H. Freeman and Company.

Lee, J.-H., Jang, H., and Kim, H.-J. (2020). Iterative job splitting algorithms for parallel machine scheduling with job splitting and setup resource constraints. *Journal of the Operational Research Society*, 72(4):780–799.

Li, K., Chen, J., Fu, H., Jia, Z., and Fu, W. (2019). Uniform parallel machine scheduling with fuzzy processing times under resource consumption constraint. *Applied Soft Computing Journal*, 82:105585.

Ozer, E. A. and Sarac, T. (2019). MIP models and a matheuristic algorithm for an identical parallel machine scheduling problem under multiple copies of shared resources constraints. *TOP*, 27:94–124.

Prata, B. A., Abreu, L. R., and Lima, J. Y. F. (2021). Heuristic methods for the single-machine scheduling problem with periodical resource constraints. *TOP*, 29:524–546.

Sekkal, N. and Belkaid, F. (2020). A multi-objective simulated annealing to solve an identical parallel machine scheduling problem with deterioration effect and resources consumption constraints. *Journal of Combinatorial Optimization*, 40(3):660–696.

Tran, T., Araujo, A., and Beck, J. (2016). Decomposition methods for the parallel machine scheduling problem with setups. *Informs Journal of Computing*, 28(1):83–95.

Vallada, E., Villa, F., and Fanjul-Peyro, L. (2019). Enriched metaheuristics for the resource constrained unrelated parallel machine scheduling problem. *Computers and Operations Research*, 111:415–424.

Villa, F., Vallada, E., and Fanjul-Peyro, L. (2018). Heuristic algorithms for the unrelated parallel machine scheduling problem with one scarce additional resource. *Expert Systems with Applications*, 93:28–38.

Yepes, J. C., Perea, F. Villa, F., and Ruiz, R. (2021). Bi-objective parallel machine scheduling with additional resources during setups. *European Journal of Operational Research*, 292(2):443–455.

Yepes, J. C., Villa, F., Perea, F., and Caballero, J. P. (2020). Grasp algorithm for the unrelated parallel machine scheduling problem with setup times and additional resources. *Expert Systems With Applications*, 141:1–12.

Yunusoglu, P. and Topaloglu Yildiz, S. (2021). Constraint programming approach for multi-resource-constrained unrelated parallel machine scheduling problem with sequence-dependent setup times. *International Journal of Production Research (In Press)*.

Appendix: Mathematical model

In this appendix we adapt the MILP in Fanjul-Peyro (2020) to the UPMSR-PS, which needs the following sets of variables:

- $C_{\max} \geq 0$ is the makespan.
- $C_j \geq 0$ is the completion time of job j. We set $C_0 = 0$.
- $X_{ijk} = 1$ if job k is processed right after job j on machine i, and takes value 0 otherwise.
- $Y_{ij} = 1$ if job j is processed on machine i, and takes value o otherwise.
- $B_j \geq 0$ is the completion time of the setup right before the processing of job j, on the machine where j is processed.
- $R_j^P \in [0, r_{\max}^P]$ defines the resources devoted in the processing of job j.
- $R_j^S \in [0, r_{\max}^S]$ defines the resources devoted in the setup before the processing of job j.
- $D_{jk}^P, E_{jk}^P, F_{jk}^S, G_{jk}^S$ are auxiliary binary variables.

$$\min C_{\max} \tag{4}$$

$$\text{s.t.} \sum_{j \in N_0, k \in N, j \neq k} s_{ijk} X_{ijk} + \sum_{k \in N} p_{ik} Y_{ik} \leq C_{\max},\ i \in M \tag{5}$$

$$\sum_{k \in N} X_{i0k} \leq 1,\ i \in M \tag{6}$$

$$\sum_{i \in M} Y_{ik} = 1,\ k \in M \tag{7}$$

$$Y_{ik} = \sum_{j \in N_0, j \neq k} X_{ijk},\ i \in M, k \in N \tag{8}$$

$$Y_{ij} = \sum_{k \in N_0, k \neq j} X_{ijk},\ i \in M, j \in N \tag{9}$$

$$C_k - C_j + A(1 - X_{ijk}) \geq s_{ijk} + p_{ik},\ i \in M, j \in N_0, k \in N, j \neq k \tag{10}$$

$$C_k \leq C_{\max},\ k \in N \tag{11}$$

$$B_k \leq C_k - \sum_{i \in M} p_{ik} Y_{ik},\ k \in N \tag{12}$$

$$B_k - C_j + A(1 - \sum_{i \in M} X_{ijk}) \geq \sum_{i \in M} s_{ijk} X_{ijk},\ j \in N_0, k \in N, j \neq k \quad (13)$$

$$C_k \geq \sum_{i \in M, j \in N_0} s_{ijk} X_{ijk} + \sum_{i \in M} r^P_{ik} Y_{ik},\ k \in M \quad (14)$$

$$r^P_{\max} \geq R^P_k \geq \sum_{i \in M} r^P_{ik} Y_{ik},\ k \in N \quad (15)$$

$$r^S_{\max} \geq R^S_k \geq \sum_{i \in M, j \in N} r^S_{ijk} X_{ijk},\ k \in N \quad (16)$$

$$C_k - C_j + A(1 - D^P_{jk}) \geq \sum_{i \in M} p_{ik} Y_{ik},\ j \in N_0, k \in N_0 \quad (17)$$

$$R^P_k - R^P_j + A(1 - E^P_{jk}) \geq \sum_{i \in M} r^P_{ik} Y_{ik},\ j \in N_0, k \in N_0 \quad (18)$$

$$D^P_{jk} + D^P_{kj} + E^P_{jk} + E^P_{kj} \geq 1,\ j \in N_0, k \in N_0, k > j \quad (19)$$

$$B_k - B_j + A(1 - F^S_{jk}) \geq \sum_{i \in M, \ell \in N_0} s_{i\ell k} X_{i\ell k},\ j \in N_0, k \in N_0 \quad (20)$$

$$R^S_k - R^S_j + A(1 - G^S_{jk}) \geq \sum_{i \in M, \ell \in N_0} r^S_{i\ell k} X_{i\ell k},\ j \in N_0, k \in N_0 \quad (21)$$

$$F^S_{jk} + F^S_{kj} + G^S_{jk} + G^S_{kj} \geq 1,\ j \in N_0, k \in N_0, k > j \quad (22)$$

Constraints (6)-(11) define the structure of the Unrelated Parallel Machines Scheduling problem with setups, without additional resources (see Avalos-Rosales et al. (2015) and Fanjul-Peyro et al. (2019)). Constraints (12)-(14) define the completion times for setups and jobs processing. Constraints (15) and (16) ensure that the limit of available resources is not exceeded, which is completed with the group of constraints for specific resources in processing (constraints (17)-(19)), and the group for specific resources in setups (constraints (20)-(22)). A is a large enough positive constant. More details about the variables and constraints involved in this model can be found in Fanjul-Peyro (2020).