Constructive Heuristics for the Unrelated Parallel Machine Scheduling Problem with Machine Eligibility and Setup Times*

Paz Perez-Gonzalez[1]†, Victor Fernandez-Viagas[1],
Miguel Zamora[2], Jose M. Framinan[1]

[1] Industrial Management, School of Engineering, University of Seville,

Ave. Descubrimientos s/n, E41092 Seville, Spain, {pazperez,framinan}@us.es

[2] CIAT Air Thermodynamics R & D Department, Cordoba, Spain

Abstract

This work considers a scheduling problem identified in a factory producing customised Heating, Ventilation and Air Conditioning (HVAC) equipment. More specifically, the metal folding section is modelled as unrelated parallel machines with machine eligibility and sequence-dependent setup times. The objective under consideration is the minimisation of the total tardiness. The problem is known to be NP-hard so approximate methods are needed to solve real-size instances. In order to embed the scheduling procedure into a decision support system providing high-quality solutions in nearly real time, our goal is to develop fast, efficient constructive heuristics for the problem. To this end, we select existing heuristics for related problems and adapt them to the problem under consideration. In addition, we propose a set of heuristics with novel repair and improvement phases. The performance of the methods adapted and our proposals are compared with the optimal/approximate solutions obtained by a solver for an MILP in two sets of instances with small and medium sizes. Additionally, big-size instances (representing more realistic cases for our company) have been solved using the proposed constructive heuristics, providing efficient solutions in negligible computational times.

Keywords: Scheduling, Constructive Heuristics, Unrelated Parallel Machines, Setup times

*This is the Submmited Manuscript of an article published by Elsevier in Computers and Industrial Engineering 131, May 2019, pp. 131-145, available online: http://dx.doi.org/10.1016/j.cie.2019.03.034

†Corresponding author. Tel.: +34-954487214.

1 Introduction

Manufacturing highly customised products requires machines capable to efficiently perform a large number of different operations. Although, in many cases, these machines exhibit a varying degree of flexibility (usually at the expense of requiring relatively high setup times among operations), there may be operations that must be carried out using specific machinery. At the shop-floor scheduling level, this translates into a manufacturing scenario where a job can be only processed on a subset of all machines available –machine eligibility–, and where, in these eligible machines, there may be substantial setup times depending on the specific characteristics of the job to be processed.

In this paper, we address the problem of scheduling jobs in unrelated parallel machines with sequence-dependent setup times where not all jobs can be processed on all machines, with the objective of minimising the total tardiness of the jobs. Our work is motivated by a scheduling problem identified in the metal folding section of a factory producing customised Heating, Ventilation and Air Conditioning (HVAC) equipment for large facilities (i.e. airports, shopping malls, etc.). We focus on a particularly complex problem within this factory (see Section 2 for details), which is the manufacturing of the different metal pieces that constitute the boxes and ducts of the air handling units. Given their specific characteristics –size, shape, thickness, etc.– (in some cases they are one-of-a-kind product) together with the large number of these pieces per box, the folding process is carried out by a mix of automated/manual machines, each one with different characteristics and performance, arranged in parallel.

A due date for each metal piece can be established by offsetting the equipment's due date according to the estimated assembly –last step in the process– time, which operates on a FIFO basis. However, it is not advantageous to fold the pieces using FIFO, as the time required to prepare the tools in the folding machines –usually much higher than the processing time itself– is clearly highly sequence-dependent. It is then convenient to process consecutively products with similar shapes so tool changes are minimised. At the same time, not all folding machines can process the dimensions and thickness of the different pieces. All these characteristics conform a scheduling problem on unrelated parallel machines with setup times and machine eligibility. In the company, this problem was being solved according to the experience of the section's foreman, although, in view of the high number of pieces that have to be scheduled in a shift –around 200–, rather simple rules were used to assign the jobs to the machines, and the schedule of the jobs within each machine was later carried out by each machine supervisor on each shift, who most of the times tried to minimise setup times without a real visibility of the due date of the jobs. As a consequence, delays and expediting 'hot jobs' were not uncommon. The initial idea of our research was to explore whether a more formal, model-based, approach for the problem could be used to obtain better schedules. A primary objective of the company is that the jobs are completed not later than their due date, however, the tightness of some of the due dates may render the problem infeasible, so the problem has been modelled with total tardiness objective minimization. The scheduling problem in parallel machines with total tardiness objective is already NP-hard, so it is the problem under

consideration and, as a consequence, we focus on developing procedures to provide high-quality (but not necessarily optimal) solutions for a large number of jobs in very short time intervals. More specifically, we propose very fast constructive heuristics that outperform the available heuristics for related problems, which were adapted to the problem under consideration.

The structure of the paper is as follows. The motivation is described in Section 2, while the description of the problem and the related literature is presented in Section 3. The problem is modelled as a Mixed Integer Linear Programming (MILP) model in Section 4, so small instances can be solved to assess the performance of the heuristics proposed in Section 5. More specifically, we propose adapted versions of two heuristics from the related literature and new heuristics. All methods are tested on different sets of instances with small-, medium- and big-size instances in Section 6. The computational experience is shown in Section 7. Finally, conclusions and future research lines are presented in Section 8.

2 Background

As mentioned in the introduction, the problem under consideration takes place in a HVAC factory located in Southern Spain. The equipments are manufactured in the facility which is divided in two main parts: Metal Manufacturing and Assembly. The first part is devoted to manufacture the different metal pieces of the equipment, while in the second the equipment is build by assembling the metal parts together with the expansion valves, coils, compressors, and the electrical and control systems. The main competitive advantage of the company is to provide the customers with highly customized equipment in very short lead times. Therefore, Sales Managers promise very tight due dates and, once a customer order arrives, the Engineering Department produces a bill of material for each equipment. Since the assembly process is rather stable, due date fulfilment heavily depends on the performance of the Metal Manufacturing Area.

Figure 1 shows a scheme of the layout of the Metal Manufacturing Area. Metal sheets enter in the process as raw material using an Automatic Store and Retrieval System (AS/RS), where it is buffered until it is cut in one of the three automated cutting machines. After the cut, each piece obtained can be considered as a reference (job) and is placed in the AS/RS waiting to be folded. The folding section is formed by three stations: the first one is composed by two Automatic Folding machines fed by Robots (AFR), supervised by one or two workers who introduce the technical drawing in the machines to carry out the corresponding folding; the second one is composed of two Automatic Folding machines fed Manually (AFM), with one or two workers feeding and supervising them; and finally, the third station is formed by five manual folding machines fed manually (MFM). Each MFM is fed, operated and supervised by one worker. Depending on the characteristic of the reference (thickness, size, complexity of the folding, ...), it can be folded only in a specific subset of all/some AFR, AFM or MFM. For example, AFR are not able to fold long pieces of sheets, and very small pieces should be processed by MFM. Furthermore, machines of the same type are

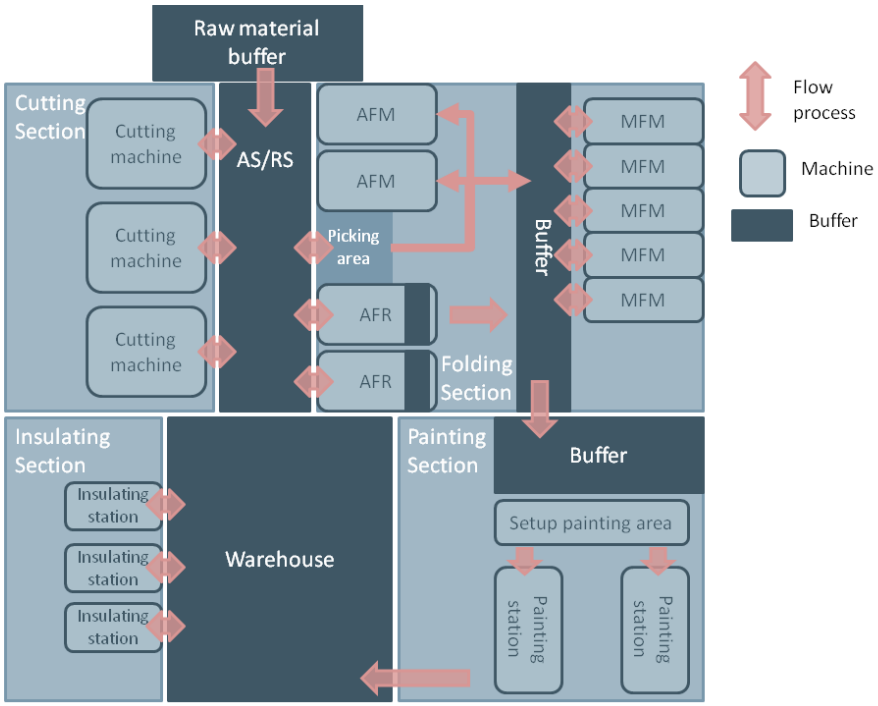


Figure 1: Scheme of the metal manufacturing process/layout

not equal, since they are from different suppliers and age, therefore the processing times of each machine within a type is not the same. Most of the time consumed in the folding process is devoted to setup (i.e. changing tools, feeding, and loading the folding sequence in the case of AFR/AFM). These setup times are not only dependent of the processing sequence –references with similar design require the same tools and consequently less setup time is needed if processed consecutively– but also on the specific machine: For example, the setup activities required for MFM (essentially tool changing) are different to those for AFR (loading the folding programme).

If a reference is to be folded in one of the AFR, then it is moved from the AS/RS to the corresponding station, where a robot feeds it to the AFR. Once folded, the robot removes the piece and places it in a buffer. In contrast, the references to be folded in AFM or MFM are moved to the picking area by the AS/RS, where one operator places them in the corresponding AFM or in the buffer for MFM. Once folded, all references are transported in batches to the Painting Section. The Painting Section consists of two painting lines operating under a FCFS policy, as the treatment is usually the same for all references. Painted references are stored in the warehouse, where the references wait to the assembly. If the reference must be insulated, it is moved to the corresponding insulation workstation prior to be transported to the assembly area.

A preliminary analysis of the data reveals that the Folding Section constitutes the bottleneck of the Metal Manufacturing Area given the higher capacity of the Cutting and Painting sections. The main problems in the Folding Section stem from two causes: First, cutting is carried out using a machine-proprietary cutting algorithm that optimises the number of pieces that can be obtained

from a metal sheet without taken into account further constraints in the Folding Section. Secondly, the scheduling of the references in the Folding section is carried out according to the experience of the section's foreman, using simple rules to assign jobs to machines and leaving the schedule of jobs within each machine to the machine supervisor on each shift. The idea is then to redesign the process so the scheduling in the Folding Section drives the cutting process and not vice versa.

By offsetting the estimated time of the assembly process and that of painting/insulation –which operate under a FIFO policy– *internal* due dates for completing the folding of each reference can be obtained. The fulfilment of these due dates is critical to ensure that the assembly takes place on time. Therefore, the scheduling problem to be considered contains the following characteristics:

- Unrelated parallel machines, implying different processing times for each job (reference) on each machine (folding machine);
- Machine eligibility, as each job can be processed in a subset of all available machines;
- Machine- and sequence-dependent setup times;
- Total tardiness minimisation objective. Note that, although the goal is to finish the jobs before their due date, job tardiness is allowed in order to avoid infeasibility issues.

Once the scheduling problem has been identified and the corresponding data gathered/estimated, the decision support system should be implemented by devising solution procedures adapted for the problem, also taking into account the required decision interval (Dios and Framinan, 2016). In our case, decision makers require fast methods to be able to iterate using different sets of jobs. Therefore, we focus on fast, constructive heuristic methods that can be implemented in the decision support system.

3 Problem Statement and Related literature

Using the notation by Graham et al. (1979), the problem under consideration can be denoted as $Rm|M_j, s_{ijk}|\sum T_j$. In this scheduling problem, n jobs have to be assigned to (and subsequently sequenced on) a set M of m parallel machines. Each job j has a different processing time on each machine i, denoted as p_{ij}, and a due date d_j. Not all jobs can be processed on all machines, so for each job j we have a subset of machines, denoted as M_j, where it can be processed. Additionally, there are setup times for each machine and each job which are sequence-dependent. Therefore, if job j is processed before job k in a machine i eligible for both jobs, the setup time is s_{ijk}. The objective of the problem is to minimize the total tardiness, i.e. $\sum T_j$ where $T_j = \max\{C_j - d_j, 0\}$, with C_j the completion time of job j. This problem is NP-hard unless there is a solution where no more than one job is assigned to each machine. Clearly, this case should verify that $n \leq m$, which does not constitute a case of interest. If this rather special case does not happen, then there is at least one machine for which more than one job is assigned. Scheduling jobs in this machine with

$\sum T_j$ objective is well-known to be NP-hard (see e.g. Framinan et al., 2014), so our problem is also NP-hard.

A number of papers related to scheduling on parallel machines based on real-world problems have been published in the last years. Gravel et al. (2000) solve a parallel machine scheduling problem inspired in an aluminium foundry considering objectives related to the metal flow; Dolgui et al. (2010) consider a problem related to multi-product metal foundries; or more recently Fu et al. (2017) tackle, in a metal packaging industry, an unrelated parallel machines scheduling problem with job splitting and sequence-dependent setup times.

To the best of our knowledge, the problem has not been addressed in the literature, with the exception of a very preliminary work by Carrasco et al. (2017). Some papers address the unrelated parallel machines problem with machine eligibility and setup times with additional constraints. Rambod and Rezaeian (2014) consider the $Rm|M_j, s_{ijk}|C_{max}$ problem with an additional constraint related to the production of imperfect items and rework processes to achieve an acceptable quality level for these imperfect items. They present an integer (non linear) programming model, and apply Genetic and Bees Algorithms to provide approximate solutions for the problem. Afzalirad and Rezaeian (2015) propose an integer (non linear) programming model to optimally solve the $Rm|M_j, s_{ijk}, r_j, prec|\sum L_j$ problem –for small instances–, as well as a Genetic Algorithm and an Ant Colony Optimization algorithm to solve approximately large instances. Later, Afzalirad and Rezaeian (2016) tackle the same problem but considering makespan objective, proposing a Genetic Algorithm and an Artificial Immune System algorithm. Chen (2006) considers machine eligibility and setup times dependent on the job family, i.e. when a job is scheduled after a job of a different type. A Tabu Search based heuristic is proposed to solve the $Rm|M_j, \text{family-setup}|\max T_j$ problem. Shahvari and Logendran (2017) consider the $Rm|M_j, \text{family-setup}, r_j, a_i, LB_f|F_l(\sum w_j C_j, \sum w_j T_j)$ problem with sequence and family dependent setup times, where a_i indicates initial unavailability periods for machines, and LB_f implies a minimum size (lower bound) for batches. They present an MILP model to solve the problem optimally. Additionally, they develop a metaheuristic based on Tabu Search which solves problems with up to three families, 5 machines and 24 jobs.

Regarding unrelated parallel machine scheduling literature considering machine eligibility constraint (without setup times), Al-Salem and Armacost (2002) compare the optimal solutions for the $Rm|M_j|C_{max}$ problem to those provided by a constructive heuristic for small instances. Afzalirad and Shafipour (2015) propose a mathematical programming model and two Genetic Algorithms for $Rm|M_j|C_{max}$ including resource constraints, i.e. additional resources must be available to process jobs. Small-size instances of the $Rm|M_j|C_{max}$ problem with controllable processing times are solved optimally by Low and Wu (2016) with an integer linear programming model, and big size instances are solved approximately by two Ant Colony Optimization metaheuristics.

Finally, regarding unrelated parallel machine problems with setup times constraints (without machine eligibility), the $Rm|s_{ijk}|C_{max}$ problem has been tackled using metaheuristics in the following references: Rabadi et al. (2006) propose a metaheuristic, denoted as Meta-RaPS, Arnaout

et al. (2010) provide an Ant Colony Optimization algorithm; Vallada and Ruiz (2011) present two Genetic Algorithms; Ying et al. (2012) a Simulated Annealing method; and finally, Diana et al. (2015) present an Immune-inspired algorithm, using instances by Al-Salem and Armacost, 2002 to compare their proposal to previous methods from the literature. A different objective and approach is considered by Rocha et al. (2008) with an integer linear programming model to solve optimally the $Rm|s_{ijk}|C_{max} + \sum w_j T_j$ problem for small-size instances.

Summarizing, there are some contributions that address scheduling in parallel machines with machine eligibility and setup times, but consider additional constraints that make the problem very different, so adapting these solution procedures does not produce efficient methods for the problem under consideration. Furthermore, most of them do not propose fast, constructive heuristics, which is the focus of our research. On the other hand, from the analysis of the literature with machine eligibility without setup times, Al-Salem and Armacost (2002) provide a constructive heuristic for the $Rm|M_j|C_{max}$ problem which can be adapted to our case. We are not aware of constructive heuristics in the literature on parallel machines considering setup times and, among the procedures used to generate the initial schedules for the metaheuristics, the one proposed by Rabadi et al. (2006) for Meta-RaPS metaheuristics is developed for $Rm|M_j|C_{max}$ and it seems the heuristic amenable to be adapted for our problem, since Ying et al. (2012) generates the initial solutions randomly, and Arnaout et al. (2010), Diana et al. (2015) and Vallada and Ruiz (2011) use initial solutions specifically designed for other scheduling problems. Therefore, the methods that will be be adapted to our problem are the construction method by Rabadi et al. (2006), and the heuristic by Al-Salem and Armacost (2002). Both methods will be presented in Section 5 but, in order to check the performance of these adaptations, we first present a MILP model in the next section.

4 Mixed Integer Linear Programming Model

In this section we present a Mixed Integer Linear Programming (MILP) model adapting the MILP proposed by Vallada and Ruiz (2011) for $Rm|s_{ijk}|C_{max}$, which in turn is based on Guinet (1993). The data of the model is presented in the following:

i	Index for the machines
j, k	Index for the jobs
N	Set of jobs
N^*	Set of jobs union a dummy job 0
M	Set of machines
p_{ij}	Processing time of job j in machine i
s_{ijk}	Sequence-dependent setup time on machine i if job j is scheduled before job k
d_j	Due date of job j
V	Big number

Decision variables are:

$$X_{ijk} \quad \begin{cases} 1 & \text{if job } j \text{ precedes job } k \text{ on machine } i \\ 0 & \text{otherwise} \end{cases}$$

C_{ij} Completion time of job j on machine i

T_j Tardiness of job j

$$\min \sum_{j \in N} T_j \tag{1}$$

s.t.

$$T_j \geq C_{ij} - dj \qquad \forall i \in M,\ \forall j \in N \tag{2}$$

$$\sum_{i \in M} \sum_{\substack{j \in N^* \\ j \neq k}} X_{ijk} = 1 \qquad \forall k \in N \tag{3}$$

$$\sum_{i \in M} \sum_{\substack{k \in N \\ j \neq k}} X_{ijk} \leq 1 \qquad \forall j \in N \tag{4}$$

$$\sum_{k \in N} X_{i0k} \leq 1 \qquad \forall i \in M \tag{5}$$

$$\sum_{\substack{h \in N^* \\ h \neq j}} X_{ihj} \geq X_{ijk} \qquad \forall j,\ k \in N,\ j \neq k,\ \forall i \in M \tag{6}$$

$$C_{ik} + V(1 - X_{ijk}) \geq C_{ij} + s_{ijk} + p_{ik} \qquad \forall j \in N^*,\ \forall k \in N,\ j \neq k,\ \forall i \in M \tag{7}$$

$$C_{i0} = 0 \qquad \forall i \in M \tag{8}$$

$$X_{ijk} \in \{0, 1\} \qquad \forall j \neq k \in N^*,\ \forall i \in M \tag{9}$$

$$T_j \geq 0 \qquad \forall j \in N \tag{10}$$

$$C_{ij} \geq 0 \qquad \forall j \in N,\ \forall i \in M \tag{11}$$

Equation (1) computes the total tardiness to be minimized. Constraint set (2) computes the tardiness of each job j. Constraint set (3) ensures that each job has only one predecessor and it is assigned to only one machine. Constraint set (4) imposes that the maximum number of successors of job j is one. Constraint set (5) ensures that at least one job is assigned for each machine i. Constraint set (6) checks, for each machine i, if job j is scheduled before job k on that machine, j has at least one predecessor (i.e. at least the dummy job). Constraint set (7) imposes that a job must not start to be processed until its predecessor has been completed. Constraint set (8) guarantees that the dummy job is the first one to be assigned on each machine i. Constraint set (9) defines the binary variables representing the precedences between jobs. Remaining sets (10 and 11) define the continuous variables of the model.

Note that the model presented is valid for the $Rm|s_{ijk}|\sum T_j$ problem, and the machine eligibility constraint is not imposed in the model, but in the data by assigning a very large processing times to those jobs on the machines where they cannot be processed.

5 Constructive heuristics

Although the MILP model presented previously may provide optimal solutions for small-size instances in reasonable computational times, this is not possible for realistic sizes due to the NP-hard nature of the problem. In this section, we focus on the development of fast heuristics to obtain good (but not necessarily optimal) solutions.

As the proposed problem has not been tackled previously (see Section 3), we adapt the existing constructive heuristics by Al-Salem and Armacost (2002) and Rabadi et al. (2006) in Section 5.1. Al-Salem and Armacost (2002) address the $Rm|M_j|C_{max}$ problem, so their constructive heuristic focuses on the assignment to the machines, since the scheduling of jobs within each machine is not relevant for makespan. Therefore, we adapt their proposal by considering $\sum T_j$ as objective and schedule the jobs in the machines according to the machine- and sequence-dependent setup times considerations. In contrast, the heuristic by Rabadi et al. (2006) addresses the $Rm|s_{ijk}|C_{max}$ problem and does not consider machine eligibility, so it has been adapted to our problem avoiding infeasible solutions, and the objective has been modified accordingly. Additionally, we also develop specific constructive heuristics for the problem under consideration, which are described in Section 5.2.

Before describing these methods in detail, let us present the notation used in all the heuristics: $M_j := \{i \in M : j \text{ can be processed on machine } i\}$ is the set of machines eligible for job j, whereas $N_i = \{j \in N : i \in M_j\}$ is the set of jobs that can be processed on machine i. Therefore, $|M_j|$ is the number of machines eligible for j and $|N_i|$ is the number of jobs that can be processed on machine i. Furthermore, we can define $J_h = \{j \in PJ : |M_j| = h\}$ as the set of jobs with h eligible machines, with $PJ \subseteq N$ the set of pending jobs to be scheduled. These sets allow us to classify the jobs according to their flexibility to be processed on the machines.

For each machine i and tuple of jobs k and j, the following two indicators SP_{ikj} and sp_{ij} can be defined, where B is a big number:

$$SP_{ikj} = \begin{cases} s_{ikj} + p_{ij} & \text{if } i \in M_j \cup M_k \\ B & \text{in the opposite case} \end{cases} \tag{12}$$

$$sp_{ij} = \begin{cases} p_{ij} + \frac{1}{|N_i - 1|} \sum_{k \neq j / i \in M_k} s_{ikj} & \text{if } i \in M_j \\ B & \text{in the opposite case} \end{cases} \tag{13}$$

Note that SP_{ikj} gives an indicator of the total processing times (i.e. including setup times) for job j if this job is processed on machine i after job k. sp_{ij} thus represents an aggregated estimate of the total processing times of job j on machine i.

Furthermore, the following two indices for the machines can be defined:

$$i^1 = arg \min_{i \in M_j} sp_{ij} \tag{14}$$

$$i^2 = arg \min_{i \in M_j - \{i^1\}} sp_{ij} \tag{15}$$

where i^1 indicates the preferred machine assignment for job j, and i^2 indicates the second preferred assignment. The ratio ρ_j can be computed for each job j to indicate the criticality of the first preferred assignment with respect to the second one:

$$\rho_j = \frac{sp_{i^1 j}}{sp_{i^2 j}} \tag{16}$$

Finally, we denote C_i the load of machine i (i.e. the completion time of the last job) which is updated every time a new job is scheduled in this machine. The last job scheduled on machine i is denoted by $l(i)$.

The methods described in this section, included in Table 1, are: AA_(α), AA_RL(α) and AA_RI(α) adapted from Al-Salem and Armacost (2002), and RMA and RMA2 adapted from Rabadi et al. (2006), described in Section 5.1; and the new heuristics H1 to H5 described in Section 5.2. All of them consider the following three phases:

Phase 1 Construction: Construct a solution using one of the following methods: Algorithm 1, Algorithm 2 (Section 5.1), Algorithm 3 or Algorithm 4 (Section 5.2).

Phase 2 Repair (optional): If the solution provided by Phase 1 results in an empty (i.e. no jobs assigned) machine, then we apply the following procedure: Identify the job eligible for this machine that it is causing the highest tardiness. If this job is scheduled in a machine with more than one job assigned, then then job is removed and inserted in the empty machine.

Phase 3 Improvement (optional): Apply one of the following algorithms:

- ILee97: This algorithm is proposed by Lee (1997) for the $1|s_{ijk}|\sum w_j T_j$ problem, so it is applied to each machine (see Lee, 1997 for details).
- IP: If the solution provided by Phase 2 verifies that $\sum T_j > 0$, for each machine with total tardiness greater than 0 and more than one job assigned, the job generating the highest tardiness is assigned to the best position selected by the procedure BestPosition_Total (see Section 5.2 for details of this procedure).

5.1 Adapted constructive heuristics

Al-Salem and Armacost (2002) present a constructive heuristic depending on a given parameter $\alpha > 0$. As this heuristic is proposed for the $Rm|M_j|C_{max}$ problem, a straightforward adaptation does not provide good solutions for our problem. Therefore, the original algorithm is adapted by embedding some intelligence with respect to the constraints and the objective of our problem, i.e.

Table 1: Proposed Methods

Method	Phase 1	Phase 2	Phase 3	Parameter
AA(α)	Algorithm 1			$\alpha \in \{0.5, 0.7, 0.9\}$
AA_RL(α)	Algorithm 1	Repair	ILee97	$\alpha \in \{0.5, 0.7, 0.9\}$
AA_RI(α)	Algorithm 1	Repair	IP	$\alpha \in \{0.5, 0.7, 0.9\}$
RMA	Algorithm 2	–	–	–
RMA2	Algorithm 2 modified	–	–	–
H1	Algorithm 3	–	–	–
H2	Algorithm 3	Repair	IP	–
H3	Algorithm 3 + EDD	Repair	IP	–
H4	Algorithm 4	–	–	–
H5	Algorithm 4	Repair	IP	–

the machine- and sequence-dependent setup times and total tardiness. The resulting algorithm is presented as Algorithm 1, and it has the same steps as in the original version regarding the construction of the sequence depending on the eligibility of each job. In the first step, the jobs that have only one eligible machine are assigned to that machine. Jobs with more than one eligible machine are assigned to machine i^1 (see Equation (14)) when $\rho_j \leq \alpha$, with ρ_j given in Equation (16). Note that the definition of i^1, i^2 and ρ_j have been modified with respect to Al-Salem and Armacost (2002) in order to include the setup times. If the condition $\rho_j \leq \alpha$ is not verified, the remaining jobs in each set J_h are sorted according to the Earliest Due Date (EDD) rule to avoid tardiness, and are assigned to machines in order to balance the load across machines. Whenever job j is assigned to the last position of machine i, C_i is computed accordingly taking into account the setup times incurred between job j and the last job scheduled in this machine $l(i)$.

Rabadi et al. (2006) propose a constructive heuristic for the $Rm|s_{ijk}|C_{max}$ problem to generate an initial schedule for its metaheuristic Meta-RaPS. In Algorithm 2 we present the adaptation to our problem. The procedure is similar to the original one, where in Step 1 the set of less loaded machines is constructed, and in Step 2 the job with minimal value of $SP_{il(i)j}$ is selected, checking in our adaptation the eligibility condition. Additionally, if there are no pending jobs that can be assigned to any of the less-loaded machines, the job and machine are selected according to the minimum value of $C_i - SP_{il(i)j} - d_j$. Steps 4 and 5 are similar to the original heuristic, changing the computation of the objective function.

RMA2 is similar to RMA, but in Algorithm 2 Step 1 obtains the set of machines with minimal total tardiness instead of the set of machines with minimal load. More specifically, it obtains $T_{min} = \{i \in M : T_i = min_{k \in M} T_k\}$, with T_k the total tardiness of jobs scheduled in machine k, and T_i the set of machines with minimum total tardiness. Step 2 selects j^* and i^* so they obtain the minimal total tardiness (instead of the minimal value of SP). Ties are broken in the same way than in Algorithm 2.

Algorithm 1 Constructive heuristic: Adapted from Al-Salem and Armacost (2002)

procedure AA(α)
  Step 0
  Set $PJ = \emptyset$
  $T_{sum} = 0$
  for $i = 1, \ldots, m$ **do**
    $l(i) = 0$; $C_i = 0$; $assign = false$
  end for
  Step 1
  Let $\Pi = (\pi_1, \ldots, \pi_n)$ be the sequence of jobs in non-increasing order of d_j
  for $j = 1, \ldots, n$ **do**
    if $|M_{\pi_j}| = 1$ **then**
      i^* is the eligible machine of π_j; $assign = true$
    else
      if $\rho_{\pi_j} \leq \alpha$ **then**
        $i^* = i^1$; $assign = true$
      else
        $PJ = PJ \cup \{\pi_j\}$
      end if
    end if
  end for
  if $assign = true$ **then**
    $l(i^*) = \pi_j$
    if $C_{i^*} = 0$ **then**
      $C_{i^*} = p_{i^*\pi_j}$
    else
      $C_{i^*} = C_{i^*} + SP_{i^*l(i)\pi_j}$
    end if
    $T_{\pi_j} = \max\{C_{i^*} - d_{\pi_j}, 0\}$; $T_{sum} += T_{\pi_j}$
  end if
  Step 2
  for $h = 2, \ldots, m$ **do**
    Let Π^h be the sequence of jobs in J_h in non-increasing order of d_j
    for each $j \in \Pi^h$ **do**
      $i^* = arg\min_{i \in M_j}\{C_i + sp_{ij}\}$
      $l(i^*) = j$
      if $C_{i^*} = 0$ **then**
        $C_{i^*} = p_{i^*j}$
      else
        $C_{i^*} = C_{i^*} + SP_{i^*l(i^*)j}$
      end if
      $T_j = \max\{C_{i^*} - d_j, 0\}$; $T_{sum} += T_j$
    end for
  end for
end procedure

Algorithm 2 Constructive heuristic: Adapted from Rabadi et al. (2006)

procedure RMA
  Step 0
  Set $PJ = \{1, \ldots, n\}$
  $T_{sum} = 0$
  for $i = 1, \ldots, m$ **do**
    $l(i) = 0$; $C_i = 0$
  end for
  Step 1
  $L_{min} = \{i \in M : C_i = min_{k \in M} C_k\}$; (set of machines with minimum load)
  Step 2
  Select $j^* \in PJ$ and $i^* \in M_{j^*} \cap L_{min}$ such as they yield the minimal value for $SP_{il(i)j}$ (In case of ties with respect to j, select j^* so $|M_{j^*}|$ is minimal. In case of ties with respect to i, select i^* so $|N_{i^*}|$ is minimal).
  if $M_j \cap L_{min} = \varnothing \; \forall j \in PJ$ (there are not pending jobs eligible for any machine in L_{min}) **then**
    Select $j^* \in PJ$ and $i^* \in M_{j^*}$ such as $C_i - SP_{il(i)j} - d_j$ is minimal (In case of ties, select j^* sp $|M_{j^*}|$ is minimal. In case of ties, select i^* so $|N_{i^*}|$ is minimal).
  end if
  Step 3
  $l(i^*) = j^*$
  Step 4
  if $C_{i^*} = 0$ **then**
    $C_{i^*} = p_{i^* j^*}$
  else
    $C_{i^*} = C_{i^*} + SP_{i^* l(i^*) j^*}$
  end if
  $T_{j^*} = \max\{C_{i^*} - d_{j^*}, 0\}$
  $PJ = PJ - \{j^*\}$; $N_{i^*} = N_{i^*} - \{j^*\}$
  Step 5
  if $PJ = \varnothing$ **then**
    $T_{sum} = \sum_{j \in N} T_j$ and STOP
  else
    Go to Step 1
  end if
end procedure

5.2 New constructive heuristics

In this section we propose the new heuristics H1-H5. These heuristics use two different algorithms for the Construction Phase (see Table 1) described in Algorithm 3 and Algorithm 4. Both algorithms use the following methods:

- BestPosition(j, i): Job j is scheduled in the best position (i.e the position which provides the minimal value of $\sum T_j$) among all positions in its eligible machine i. However, if the tardiness of the sequence on machine i prior to scheduling job j was zero and the tardiness on the machine after inserting job j in a given position is also zero, then the position to insert job j is the one with minimal value of $\sum(C_j - d_j)$. This method returns the so-obtained position k^*.

- BestPosition_Total(j): Job j is assigned and scheduled in the best position (i.e. that with minimal value of total tardiness) among all positions in all its eligible machines. This method returns the machine i^* and the position k^*.

Algorithm 3 generates the sets J_h of jobs with h eligible machines. For each value of h, each job $j \in J_h$ is assigned to its eligible machine (when $h = 1$), or to the less-loaded machine (in case of ties, it is asssigned to the less eligible machine, i.e. the machine with lowest value of $|N_i|$) if $h > 1$. If the selected machine i^* is empty, then $C_i = p_{i^*j}$. Otherwise, the position to insert j in i^* is obtained by BestPosition(j, i^*). H1 and H2 use Algorithm 3 in Phase 1 (Construction). H2 includes the phases 2 (Repair) and 3 (Improvement) explained previously. Heuristic H3 is similar to H2, but the jobs in each set J_h are sorted in non-increasing order of due dates (EDD).

Algorithm 4 is similar to Algorithm 3, using the EDD order as in H3, but uses BestPosition_Total(j) as criterion to select the machine if $h > 1$. Heuristics H4 and H5 use Algorithm 4 in Phase 1 (Construction). H5 includes the phases 2 (Repair) and 3 (Improvement) explained previously.

6 Sets of instances

In order to test the methods presented in the previous section, instances representing different cases have to be constructed. The following data are required for each instance: number of jobs n and machines m; processing times, p_{ij}; setup times s_{ijk}; due dates d_j; and the set of eligible machines by a job j, M_j.

In this paper we have generated different sets of instances based on Vallada and Ruiz (2011) for the $Rm|s_{ijk}|C_{max}$ problem, including due dates and machine eligibility to adapt them to our problem. Therefore, processing times have been generated uniformly between 1 and 99, i.e. $p_{ij} \sim U[1, 99]$, and we have considered two of the four levels for setup times: $s_{ijk} \sim U[1, 49]$ and $U[1, 124]$. Regarding due dates, we have generated them using the uniform distribution $U[\widetilde{C}_{max}(1 - \tau -$

Algorithm 3 New Construction 1

procedure H1
  Step 0
  Set $J_h = \{j \in N / |M_j| = h\}$
  $T_{sum} = 0$
  for $i = 1, \ldots, m$ **do**
    $C_i = 0$
  end for
  Step 1
  for $h = 1, \ldots, m$ **do**
    for each $j \in J_h$ **do**
      if $h = 1$ **then**
        Let i^* be the eligible machine of j
      else
        $i^* = arg\min_{i \in M_j} C_i$. In case of ties select i^* such that $|N_i|$ is lower
      end if
      if $C_{i^*} = 0$ **then**
        Assign j to i^*
        $C_{i^*} = p_{i^* j}$
      else
        k^*=BestPosition(j, i^*)
        Update C_{i^*}
      end if
      Update T_{sum}
      $N_{i^*} = N_{i^*} - \{j\}$
    end for
  end for
end procedure

Algorithm 4 New Construction 2

```
1: procedure H4
2:     Step 0
3:     Set J_h = {j ∈ N/|M_j| = h}
4:     T_sum = 0
5:     for i = 1, ..., m do
6:         C_i = 0
7:     end for
8:     Step 1
9:     for h = 1, ..., m do
10:        Let Π^h be the sequence of jobs in J_h in non-increasing order of d_j
11:        for each j ∈ Π^h do
12:            if h = 1 then
13:                Let i* be the eligible machine of j
14:                if C_i* = 0 then
15:                    Assign j to i*
16:                    C_i* = p_i*j
17:                else
18:                    k*=BestPosition(j, i*)
19:                    Update C_i*
20:                end if
21:            else
22:                (k*, i*)=BestPosition_Total(j)
23:                Update C_i*
24:            end if
25:            Update T_sum
26:        end for
27:    end for
28: end procedure
```

$R/2), \widetilde{C}_{max}(1-\tau+R/2)]$ (see for example Chen (2006), Lin et al. (2010) and Lee et al. (2013)) considering the parameters $\tau \in \{0.2, 0.4\}$ and $R \in \{0.2, 0.4\}$ where

$$\widetilde{C}_{max} = \max_{j \in N} \frac{\sum_{i \in M_j} p_{ij} + \frac{\sum_{k=1}^{n} s_{ikj}}{n}}{|M_j|} \cdot \frac{n}{m}$$

The procedure adopted for the generation of machine eligibility is described in Algorithm 5. This method is a combination of the proposals by Alagöz and Azizoğlu (2003) and Gokhale and Mathirajan (2012), so we control the probabilities by a constant, denoted p, and the number of eligible machines for each job. Values of p are taken from $\{50, 80, 100\}$.

Algorithm 5 Machine eligibility procedure

procedure MACHINE ELIGIBILITY PROCEDURE(p)
  for $j = 1, \ldots, n$ **do**
    Generate a random number $k \in [1, 100]$
    if $k \leq p$ **then**
      $|M_j| \in U[1, m/2]$
    else
      $|M_j| \in U[1, m]$
    end if
    Select randomly $|M_j|$ machines eligible for j
  end for
  if $\exists i \in M$ such as $i \notin M_j \ \forall j \in N$ **then**
    Assign randomly a job to i
  end if
end procedure

We have considered different values of n and m to generate three sets of instances depending on these parameters. 10 instances per combination have been constructed.

- Small-size instances: $n = \{6, 8, 10, 12\}$ and $m = \{2, 3, 4, 5\}$ in the same way than in Vallada and Ruiz (2011). In total, a set of 3,840 instances have been solved optimally using the MILP model to provide an exact comparison for the constructive heuristics proposed.

- Medium-size instances: $n = \{15, 20, 25, 30, 35, 40\}$ and $m = \{2, 4, 6, 8\}$. This set of 5,760 instances has been generated to test the limits of the MILP model, as it was able to solve the small instances given by Vallada and Ruiz (2011) in negligible computational time (see Section 7).

- Big-size instances: $n = \{50, 100, 150, 200, 250\}$ and $m = \{10, 15, 20, 25, 30\}$, in the same way than Vallada and Ruiz (2011). This set contains 6,000 instances with sizes similar to the ones described in Section 3.

7 Computational Experiments

In this section, the MILP (Section 4) and the constructive heuristics (Section 5) are evaluated using the sets of instances described in Section 6. Heuristics have been implemented in C#. Additionally, the MILP model has been implemented in C# using the solver Gurobi (Gurobi Optimization Inc., 2017) with a time limit of 100 seconds.

In order to determine the best method among those proposed, we compare FO_{IM} the objective value for each method M for each instance I using the relative deviation index (Fernandez-Viagas and Framinan, 2015) as follows:

$$RDI_{IM} = \frac{FO_{IM} - Best_I}{Worst_I - Best_I}$$

Where $Best_I$ and $Worst_I$ are the best and worst total tardiness value provided by all methods respectively. Additionally, for each instance I and for each method M we have registered the computational time. The average RDI for a given method for a given set of instances is denoted ARDI.

We have carried out a Design of Experiments (DoE) for each set of instances with the following factors: Method, n, m, *setup*, p, τ and R. Therefore, the experiments consist of a full factorial design with 10 observations per combination and response variable RDI. The levels for each factor are the following:

- Method (different levels depending on the set of instances):
  - Small and Medium sizes instances (17 levels): MILP, AA(0.5), AA(0.7), AA(0.9), AA_RL(0.5), AA_RL(0.7), AA_RL(0.9), AA_RI(0.5), AA_RI(0.7), AA_RI(0.9), RMA, RMA2, H1, H2, H3, H4, H5.
  - Big sizes instances (16 levels): AA(0.5), AA(0.7), AA(0.9), AA_RL(0.5), AA_RL(0.7), AA_RL(0.9), AA_RI(0.5), AA_RI(0.7), AA_RI(0.9), RMA, RMA2, H1, H2, H3, H4, H5.
- n (different levels depending on the set of instances):
  - Small sizes instances (4 levels): 6, 8, 10, 12
  - Medium sizes instances (6 levels): 15, 20, 25, 30, 35, 40
  - Big sizes instances (5 levels): 50, 100, 150, 200, 250
- m (different levels depending on the set of instances):
  - Small sizes instances (4 levels): 2, 3, 4, 5
  - Medium sizes instances (4 levels): 2, 4, 6, 8
  - Big sizes instances (5 levels): 10, 15, 20, 25, 30

- *setup* (2 levels): $U[1, 49]$, $U[1, 124]$
- p (3 levels): 50, 80, 100
- τ (2 levels): 0.2, 0.4
- R (2 levels): 0.2, 0.4

7.1 Small-size instances

All 3,840 instances have been solved in 0.15 seconds on average. In order to test the performance of the constructive heuristics against the optimal solutions, we have solved the instances using all heuristic methods. The CPU times have been less than one millisecond per instance for all methods.

Table 2 shows the ARDI values provided for each level of the factors. It can be observed that the best method for this set of instances is H5, with a total average of 5.863% respect to the optimum. This is the best method for all levels of all factors as it can be observed in Table 2. AA_RI(0.5), AA_RI(0.7) and H3 have similar performance but they provide an ARDI over 72% worse than H5. RMA and RMA2 have the worst performance, so seems better to adapt methods designed for machine eligibility constraint. Note that AA(α) is improved including the algorithm by Lee (1997), AA_RL(α), but the performance is better with the Repair and Improvement phases described in Section 5, i.e. AA_RI(α). In fact, all the best methods include the proposed Improvement and Repair phases.

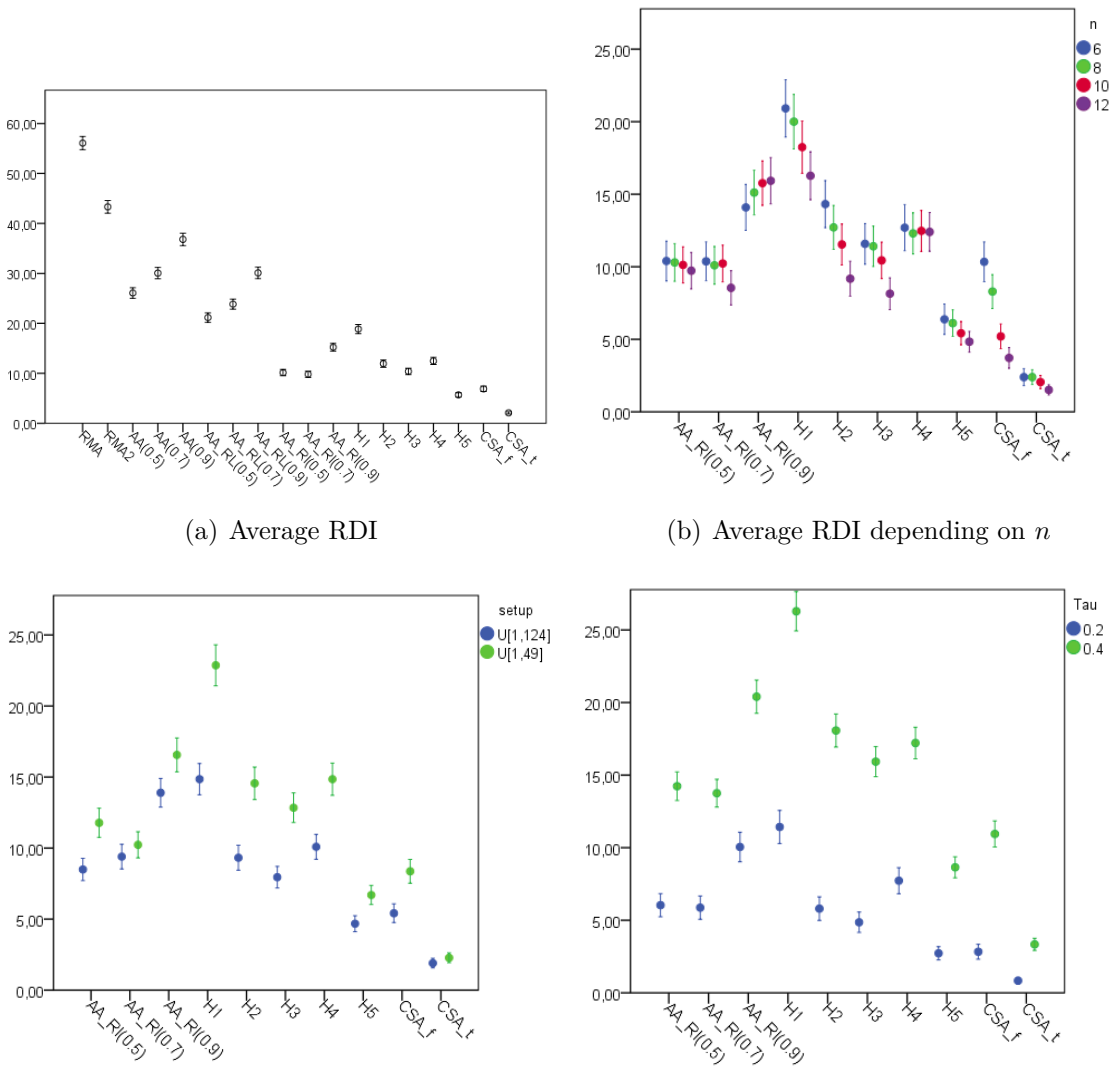
Figure 2 contains four graphics with Tukey 95% confidence intervals to analyse the influence of the levels of the factor Method on the variable RDI. Figure 2(a) shows the global performance for all methods. It can be seen that H5 is clearly the best method with statistically significant differences. Additionally, Figures 2(b), 2(c) and 2(c) include the results for the factors presenting the most relevant influence (n, setup and τ) and considering only the best methods in order to have a more detailed view of the statistical differences among them. Hence, in Figure 2(b) it can be observed that H5 improves its performance when n increases, with statistical differences for all levels of this factor to the rest of methods. From Figure 2(c) we conclude that all methods provide better results when setup times are generated in $U[1, 124]$, being again H5 the best method statistically different to the others for both levels. In the same way, in Figure 2(d) we can observe that all methods are more efficient when τ is 0.2. H5 is the best for both levels with statistic differences.

7.2 Medium-size instances

This set of instances has been solved using the MILP solver with a time limit of 100 seconds, and also using the constructive heuristics to compare the results. In this case the RDI values are computed with respect to the minimum obtained for all methods since not all instances (5,699 out of 5,760, 98.96%) have been solved optimally within the time limit. For this reason, in Table 3 the ARDI of the MILP is different to zero. Taking into account that for some instances the solver has

		AA(0.5)	AA(0.7)	AA(0.9)	AA_RL(0.5)	AA_RL(0.7)	AA_RL(0.9)	AA_RI(0.5)	AA_RI(0.7)	AA_RI(0.9)	H1	H2	H3	H4	H5	RMA	RMA2
n	6	30.307	34.023	39.690	21.375	21.863	25.870	10.885	11.095	14.535	21.930	15.144	12.460	13.182	6.883	54.368	42.391
	8	26.829	31.654	39.055	21.571	25.098	32.324	10.464	10.314	15.343	20.546	13.178	11.676	12.455	6.225	58.421	43.971
	10	25.409	29.418	36.470	21.965	25.432	32.107	10.245	10.353	15.875	18.431	11.616	10.545	12.551	5.465	57.184	43.613
	12	23.406	26.728	33.824	20.940	24.127	31.385	9.795	8.617	16.001	16.619	9.263	8.219	12.569	4.877	56.804	45.547
m	2	23.181	23.847	25.855	22.226	22.944	25.299	11.781	11.328	14.367	14.275	8.405	8.042	20.852	7.356	52.872	52.806
	3	25.339	30.047	36.746	21.767	25.631	32.759	9.786	8.511	14.992	18.316	12.642	9.348	11.372	5.159	53.142	42.662
	4	29.437	33.579	42.252	22.479	25.331	32.968	10.365	10.067	16.329	21.315	14.096	12.200	10.955	5.911	58.113	42.302
	5	27.993	34.351	44.185	19.379	22.615	30.660	9.457	10.473	16.066	23.619	14.058	13.310	7.578	5.025	62.649	37.751
p	50	23.617	28.112	37.589	18.326	21.800	30.392	8.279	7.892	14.391	18.775	11.735	10.285	9.437	4.341	55.667	41.147
	80	25.783	29.611	36.031	20.948	23.567	29.618	10.076	9.513	15.197	19.628	11.560	10.244	12.270	5.455	56.401	43.853
	100	30.063	33.644	38.159	25.114	27.024	31.255	12.686	12.879	16.729	19.740	13.605	11.645	16.361	7.791	58.014	46.642
setup	$U[1, 49]$	24.256	28.682	35.307	20.057	23.183	28.659	12.039	10.599	16.831	23.635	15.097	13.305	15.136	6.946	54.051	39.323
	$U[1, 124]$	28.719	32.230	39.212	22.869	25.077	32.184	8.655	9.591	14.046	15.128	9.503	8.145	10.242	4.779	59.337	48.438
R	0.2	28.391	32.608	41.100	23.366	25.776	33.889	11.089	10.901	16.368	18.391	12.485	10.602	10.789	5.555	56.622	40.425
	0.4	24.584	28.304	33.419	19.560	22.485	26.954	9.606	9.289	14.509	20.372	12.115	10.848	14.590	6.170	56.766	47.336
τ	0.2	18.013	21.941	28.432	13.862	16.377	22.255	6.089	6.020	10.137	11.634	5.918	4.977	7.752	2.775	48.395	33.545
	0.4	34.963	38.970	46.087	29.064	31.884	38.588	14.605	14.170	20.740	27.129	18.682	16.473	17.626	8.950	64.993	54.216
Total		26.488	30.456	37.260	21.463	24.130	30.422	10.347	10.095	15.439	19.381	12.300	10.725	12.689	5.863	56.694	43.880

Table 2: Average RDI: Small Sizes Instances



(a) Average RDI

(b) Average RDI depending on n

(c) Average RDI depending on *setup*

(d) Average RDI depending on τ

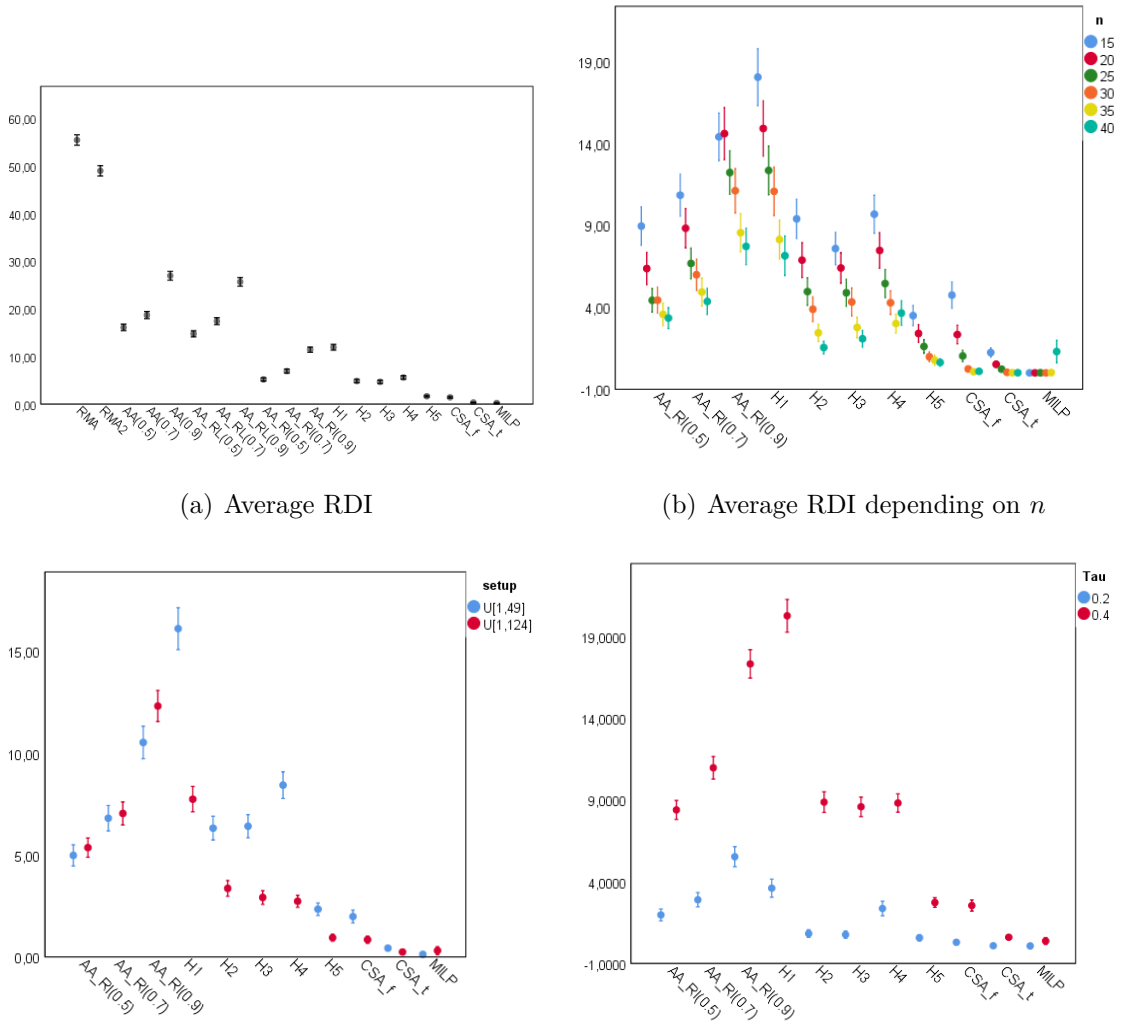
Figure 2: Tukey confidence intervals 95% Small Sizes Instances

reached the time limit of 100 seconds, its average computational time is 7.98 seconds (see the last row in Table 3), with an average of 48.19 seconds for the biggest size ($n = 40$ and $m = 8$).

Regarding constructive heuristics, in Table 3 it can be observed that H5 is the best heuristic for this set of instances, with 1.654% of Average RDI. The second best performance is provided by H3 with 4.701% of ARDI. In the same way that the results for small-size instances, RMA and RMA2 do not provide good results. Average CPU times needed for all constructive heuristics are almost negligible as it can be observed in the last row of Table 3.

Figure 3 shows the Tukey 95% confidence intervals. Figure 3(a) shows that H5 is the best constructive heuristic. In Figure 3(b) it can be seen that H5 presents the best performance for all values of n significantly different to the rest of constructive heuristics, and for the biggest value

$n = 40$ there are not statistical differences between MILP and H2, H3 and H5. Regarding setup times, Figure 3(c) shows that H5 is the best method and significantly different to the rest of the methods for the two levels. However, Figure 3(d) shows a very good performance of H5 when the parameter τ is 0.4, with statistical differences with the rest of the methods, but there are not differences between H2 and H3 when $\tau = 0.2$.



(a) Average RDI

(b) Average RDI depending on n

(c) Average RDI depending on *setup*

(d) Average RDI depending on τ

Figure 3: Tukey confidence intervals 95% Medium Sizes Instances

7.3 Big-size instances

This set of instances has not been solved by the MILP model due to the computational requirements, so the results regarding RDI are computed with respect to the minimal value obtained for all methods. It can be observed in Table 4 that the best method for this set of instances is AA_RI(0.5)

		AA(0.5)	AA(0.7)	AA(0.9)	AA_RL(0.5)	AA_RL(0.7)	AA_RL(0.9)	AA_RI(0.5)	AA_RI(0.7)	AA_RI(0.9)	H1	H2	H3	H4	H5	RMA	RMA2	MILP
n	15	22.954	26.304	34.938	19.853	22.903	30.582	9.025	10.938	14.492	18.152	9.474	7.692	9.758	3.521	61.053	47.013	0.000
	20	18.684	23.060	32.320	16.882	21.163	30.952	6.376	8.865	14.666	15.090	6.898	6.485	7.543	2.444	60.787	46.261	0.000
	25	15.137	17.709	27.703	14.093	16.854	26.806	4.421	6.678	12.222	12.375	4.982	4.896	5.445	1.609	57.205	48.553	0.004
	30	15.329	17.141	26.300	14.368	16.351	25.785	4.428	5.976	11.094	11.057	3.875	4.319	4.272	0.979	53.452	48.608	0.000
	35	12.616	14.519	21.143	12.156	14.072	20.777	3.556	4.915	8.530	8.109	2.426	2.752	2.992	0.753	52.389	50.531	0.004
	40	12.228	13.823	19.688	11.465	13.494	19.419	3.330	4.339	7.696	7.122	1.528	2.063	3.632	0.617	48.424	53.432	1.283
m	2	12.386	11.504	11.843	12.315	11.451	11.774	5.052	5.033	5.903	3.257	1.135	1.190	7.282	1.012	47.145	55.879	0.010
	4	18.606	20.242	25.948	17.930	19.585	25.615	5.737	7.259	11.229	9.001	3.335	3.042	6.741	1.856	50.936	49.779	0.001
	6	17.678	21.787	33.147	15.822	20.019	31.601	5.198	7.936	14.171	15.023	6.236	6.182	5.310	2.139	59.966	46.665	0.002
	8	15.961	21.504	37.124	13.145	18.836	33.890	4.771	7.579	14.497	20.655	8.749	8.391	3.096	1.610	64.158	43.942	0.847
p	50	13.633	15.186	24.372	12.361	13.994	23.055	3.329	4.815	9.273	12.932	4.962	5.173	2.999	0.873	51.484	47.450	0.068
	80	17.320	19.963	27.845	15.920	18.704	26.655	5.358	7.087	11.741	11.426	4.670	4.346	5.706	1.593	56.811	47.925	0.253
	100	17.521	21.129	28.830	16.128	19.721	27.450	6.881	8.953	13.336	11.594	4.960	4.586	8.116	2.496	58.360	51.824	0.324
setup	$U[1,49]$	12.750	15.503	22.068	11.885	14.643	21.028	5.011	6.857	10.580	16.202	6.357	6.478	8.471	2.360	49.121	43.117	0.119
	$U[1,124]$	19.566	22.016	31.963	17.720	20.303	30.412	5.368	7.047	12.321	7.766	3.371	2.924	2.743	0.948	61.982	55.015	0.311
R	0.2	16.434	20.818	32.633	15.183	19.538	31.140	5.942	8.160	13.795	9.252	4.703	4.275	4.000	1.355	52.966	45.642	0.323
	0.4	15.881	16.700	21.398	14.422	15.408	20.301	4.437	5.743	9.105	14.716	5.025	5.128	7.214	1.953	58.138	52.491	0.107
τ	0.2	6.914	9.735	17.212	6.099	8.815	16.173	1.970	2.895	5.528	3.607	0.828	0.763	2.360	0.569	43.485	34.378	0.069
	0.4	25.402	27.784	36.819	23.507	26.131	35.267	8.409	11.008	17.372	20.360	8.899	8.639	8.854	2.739	67.618	63.754	0.361
Total		16.158	18.759	27.016	14.803	17.473	25.720	5.189	6.952	11.450	11.984	4.864	4.701	5.607	1.654	55.552	49.066	0.215
CPU time		0.0004	0.0004	0.0004	0.0015	0.0015	0.0014	0.0023	0.0022	0.0021	0.0020	0.0031	0.0040	0.0035	0.0042	0.0000	0.0000	2.5131

Table 3: Average RDI and CPU times (seconds): Medium Sizes Instances

with 1.377% of ARDI, being H5 very close with 1.460%. It can be observed that, for example, for the factor τ, when it is 0.2, AA_RI(0.5) is better than H5, and the opposite occurs when $\tau = 0.4$. Average CPU times in seconds have been included for all the methods in the last row of the Table 4, being all of them very fast.

Figure 4 shows the different graphics with Tukey 95% confidence intervals for the RDI. It can be seen in Figure 4(a) that H5 and AA_RI(0.5) do not present significant differences. Figure 4(b) shows that AA_RI(0.5) has a better performance than H5 for high number of jobs, being statistically different when $n = 250$. However, there are not differences between these methods for the two levels of the factor setup, as it can be observed in Figure 4(c). The conclusions observed in the Table 4 can be seen clearly in Figure 4(d), with H5 significantly better than AA_RI(0.5) when $\tau = 0.4$ being the opposite for $\tau = 0.2$.

8 Conclusions

In this paper we consider a scheduling problem identified in a manufacturing company where more than 200 jobs must be scheduled in a set of unrelated parallel machines every day. Taking into account the different constraints of the manufacturing process, the machine layout and the company's objectives, the problem can be modelled as $Rm|M_j, s_{ijk}|\sum T_j$, i.e. unrelated parallel machines with machine eligibility and machine and sequence dependent setup times with objective the total tardiness. In addition, the company for which this problem is being studied required very fast methods to be embedded in a decision support system, so we focus on near real-time procedures to solve the scheduling problem, more specifically on constructive heuristics given the NP-hard nature of the problem.

First, the problem is modelled using an MILP model in order to be able to assess the performance of the heuristic solutions with respect to the optimal values, at least for those (small) instances that can be optimally solved in reasonable CPU times. Second, we adapt constructive heuristics found in the literature for similar problems trying to include the characteristics of our problem. More specifically, we have adapted the following methods:

- The constructive heuristic proposed by Al-Salem and Armacost (2002) for the $Rm|M_j|C_{max}$ problem, denoted AA(α) in this paper, with α a parameter of the original algorithm;
- A refinement of the AA(α) heuristic that includes an improvement phase based on the constructive heuristic developed by Lee (1997) for the $1|s_{ijk}|\sum w_j T_j$ problem, denoted AA_RL(α), and
- The constructive heuristic proposed by Rabadi et al. (2006) as initial schedule for their metaheuristic for the $Rm|s_{ijk}|C_{max}$ problem.

Third, we propose different constructive heuristics, labelled H1-H5, where H2, H3 and H5 include two phases named Improvement and Repair. In view of the good performance of AA(α), we have also

		AA(0.5)	AA(0.7)	AA(0.9)	AA_RL(0.5)	AA_RL(0.7)	AA_RL(0.9)	AA_RI(0.5)	AA_RI(0.7)	AA_RI(0.9)	H1	H2	H3	H4	H5	RMA	RMA2
n	50	8.782	13.642	31.135	7.522	12.165	27.265	2.294	4.111	10.807	28.808	6.502	6.637	9.405	0.699	69.973	38.660
	100	7.828	8.445	18.309	6.922	7.886	17.959	0.888	1.925	6.970	22.697	3.327	4.120	7.098	0.790	58.791	41.536
	150	8.916	6.787	11.089	8.412	6.412	10.955	1.028	1.208	4.537	19.830	2.149	2.595	5.168	1.576	46.404	46.466
	200	11.455	7.932	7.645	11.051	7.595	7.569	1.170	1.158	2.919	16.450	1.769	1.679	3.297	1.554	40.236	50.939
	250	13.437	8.835	6.442	13.232	8.648	6.394	1.505	1.236	2.155	15.089	1.112	1.697	3.069	2.683	36.780	44.047
m	10	15.514	11.250	11.112	15.064	11.005	10.989	2.447	2.083	3.989	9.959	1.597	1.968	7.338	0.368	39.655	49.761
	15	11.523	9.287	13.965	11.084	8.809	13.674	1.418	1.767	5.409	16.891	2.602	3.016	7.291	0.562	43.965	48.303
	20	9.519	9.138	16.256	8.774	8.500	15.549	1.244	2.035	6.048	20.743	3.335	3.561	5.905	1.126	50.093	46.858
	25	7.052	7.506	15.685	6.391	6.852	14.620	0.854	1.552	5.683	25.104	3.447	3.884	4.049	2.147	55.456	44.370
	30	6.808	8.459	17.601	5.824	7.541	15.311	0.922	2.200	6.258	30.176	3.879	4.299	3.453	3.098	63.016	32.355
p	50	10.612	9.163	13.972	9.882	8.539	13.019	1.346	1.716	5.174	23.219	2.912	3.399	3.075	1.838	48.004	41.765
	80	11.131	9.390	14.908	10.472	8.771	14.101	1.382	1.996	5.573	20.163	2.802	3.539	5.937	1.437	50.486	43.870
	100	8.507	8.832	15.891	7.929	8.314	14.965	1.403	2.070	5.685	18.342	3.201	3.099	7.810	1.105	52.820	47.354
setup	$U[1, 49]$	2.457	3.591	7.875	2.292	3.393	7.194	0.675	1.194	3.098	30.571	4.378	5.144	9.066	0.977	34.673	43.161
	$U[1, 124]$	17.710	14.665	21.973	16.564	13.690	20.863	2.079	2.661	7.857	10.579	1.566	1.547	2.148	1.944	66.201	45.498
R	0.2	6.977	7.782	18.429	6.571	7.339	17.400	1.088	2.084	7.192	13.712	2.531	2.871	4.516	1.626	46.794	45.031
	0.4	13.189	10.474	11.419	12.284	9.744	10.657	1.666	1.771	3.763	27.438	3.413	3.820	6.698	1.294	54.080	43.628
τ	0.2	1.470	2.583	9.372	1.349	2.434	8.532	0.294	0.616	2.687	4.375	0.297	0.288	3.016	2.448	38.498	37.480
	0.4	18.696	15.673	20.476	17.506	14.648	19.524	2.460	3.239	8.268	36.775	5.647	6.403	8.199	0.473	62.375	51.179
Total		10.083	9.128	14.924	9.428	8.541	14.028	1.377	1.927	5.477	20.575	2.972	3.346	5.607	1.460	50.437	44.330
CPU Time		0.0109	0.0108	0.0108	0.0121	0.0120	0.0120	0.0132	0.0132	0.0130	0.0024	0.0039	0.0044	0.0064	0.0073	0.0320	0.1473

Table 4: Average RDI and CPU times (seconds): Big Sizes Instances

implemented a version of this heuristic –denoted AA_RI(α)– including the Repair and Improvement phases.

All the aforementioned methods, including the MILP model using the solver Gurobi, have been tested on three sets of instances: small-, medium- and big-size. Gurobi provides optimal values in reasonable computational times for all small sizes and almost all medium sizes instances, but it is not able to provide good solutions for the big-size instances in less than 100 seconds. Among the heuristics, those including the repair and improvement phases provide the best performance, being H5 and AA_RI(0.5) the best ones with respect to the Average RDI. Particularly, H5 guarantees high-quality solutions H5 for instances with more than 200 jobs in negligible CPU time (less than 0.012 seconds on average when $n = 200$). Furthermore and in contrast to AA_RI(α), H5 does not have any parameter, thus avoiding the need of calibration for its implementation.

As a future research line, we would like to study this problem with the setup times dependent on the machine and on the family of jobs, since in the company there are references with reduced sequence-dependent setup times if jobs belong to the same family (i.e. a similar design of the parts). Additionally, it could be interesting to include weights in the objective function to capture the higher priority of jobs belonging to urgent orders. Finally, another option would be to apply metaheuristics to our problem, which could provide the scheduler with higher-quality solutions, although they would require much higher CPU times.

References

Afzalirad, M. and Rezaeian, J. (2015). Design of high-performing hybrid meta-heuristics for unrelated parallel machine scheduling with machine eligibility and precedence constraints. *Engineering Optimization*, pages 1–21.

Afzalirad, M. and Rezaeian, J. (2016). Resource-constrained unrelated parallel machine scheduling problem with sequence dependent setup times, precedence constraints and machine eligibility restrictions. *Computers & Industrial Engineering*, 98:40–52.

Afzalirad, M. and Shafipour, M. (2015). Design of an efficient genetic algorithm for resource-constrained unrelated parallel machine scheduling problem with machine eligibility restrictions. *Journal of Intelligent Manufacturing*, 29(2):423–437.

Al-Salem, A. and Armacost, R. L. (2002). Unrelated machines scheduling with machine eligibility restrictions. *Engineering Journal of University of Qatar*, 15:193–210.

Alagöz, O. and Azizoğlu, M. (2003). Rescheduling of identical parallel machines under machine eligibility constraints. *European Journal of Operational Research*, 149(3):523–532.

Arnaout, J.-P., Rabadi, G., and Musa, R. (2010). A two-stage Ant Colony Optimization algorithm

to minimize the makespan on unrelated parallel machines with sequence-dependent setup times. *Journal of Intelligent Manufacturing*, 21(6):693–701.

Carrasco, F., Perez-Gonzalez, P., Fernandez-Viagas, V., and Framinan, J. M. (2017). Unrelated parallel machines with sequence dependent setup times and machine eligibility. In Bousonville, T., Melo, T., Rezg, N., and Vernadat, F., editors, *IESM 2017 7th International Conference on Industrial Engineering and Systems Management*, pages 220–225, Saarbrücken, Germany.

Chen, J.-F. (2006). Minimization of maximum tardiness on unrelated parallel machines with process restrictions and setups. *The International Journal of Advanced Manufacturing Technology*, 29(5):557–563.

Diana, R. O. M., de França Filho, M. F., de Souza, S. R., and de Almeida Vitor, J. F. (2015). 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.

Dios, M. and Framinan, J. M. (2016). A review and classification of computer-based manufacturing scheduling tools. *Computers & Industrial Engineering*, 99:229–249.

Dolgui, A., Eremeev, A. V., Kovalyov, M. Y., and Kuznetsov, P. M. (2010). Multi-product lot sizing and scheduling on unrelated parallel machines. *IIE Transactions (Institute of Industrial Engineers)*, 42(7):514–524.

Fernandez-Viagas, V. and Framinan, J. M. (2015). NEH-based heuristics for the permutation flowshop scheduling problem to minimise total tardiness. *Computers & Operations Research*, 60:27–36.

Framinan, J. M., Leisten, R., and Ruiz, R. (2014). *Manufacturing scheduling systems: An integrated view on models, methods and tools*, volume 9781447162.

Fu, L.-L., Aloulou, M. A., and Triki, C. (2017). Integrated production scheduling and vehicle routing problem with job splitting and delivery time windows. *International Journal of Production Research*, 55(20):5942–5957.

Gokhale, R. and Mathirajan, M. (2012). Scheduling identical parallel machines with machine eligibility restrictions to minimize total weighted flowtime in automobile gear manufacturing. *The International Journal of Advanced Manufacturing Technology*, 60(9-12):1099–1110.

Graham, R., Lawler, E. L., Lenstra, J. K., and Rinnooy Kan, A. (1979). Optimization and heuristic in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326.

Gravel, M., Price, W. L., and Gagné, C. (2000). Scheduling jobs in an Alcan aluminium foundry using a genetic algorithm. *International Journal of Production Research*, 38(13):3031–3041.

Guinet, A. (1993). Scheduling sequence-dependent jobs on identical parallel machines to minimize completion time criteria. *International Journal of Production Research*, 31(7):1579–1594.

Gurobi Optimization Inc. (2017). Gurobi Optimizer Reference Manual. http://www.gurobi.com.

Lee, J.-H., Yu, J.-M., and Lee, D.-H. (2013). A tabu search algorithm for unrelated parallel machine scheduling with sequence- and machine-dependent setups: minimizing total tardiness. *The International Journal of Advanced Manufacturing Technology*, 69(9-12):2081–2089.

Lee, Y. H. (1997). A heuristic to minimize the total weighted tardiness with sequence-dependent setups. *IIE Transactions (Institute of Industrial Engineers)*, 29(1):45–52.

Lin, S.-W., Lu, C.-C., and Ying, K.-C. (2010). Minimization of total tardiness on unrelated parallel machines with sequence- and machine-dependent setup times under due date constraints. *The International Journal of Advanced Manufacturing Technology*, 53(1-4):353–361.

Low, C. and Wu, G.-H. (2016). Unrelated parallel-machine scheduling with controllable processing times and eligibility constraints to minimize the makespan. *Journal of Industrial and Production Engineering*, 1015(February):1–8.

Rabadi, G., Moraga, R., and Al-Salem, A. (2006). Heuristics for the unrelated parallel machine scheduling problem with setup times. *Journal of Intelligent Manufacturing*, 17:85–97.

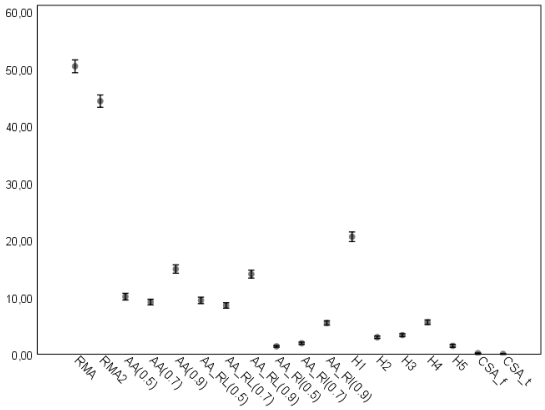
Rambod, M. and Rezaeian, J. (2014). Robust meta-heuristics implementation for unrelated parallel machines scheduling problem with rework processes and machine eligibility restrictions. *Computers & Industrial Engineering*, 77:15–28.

Rocha, P. L., Ravetti, M. G., Mateus, G. R., and Pardalos, P. M. (2008). Exact algorithms for a scheduling problem with unrelated parallel machines and sequence and machine-dependent setup times. *Computers & Operations Research*, 35(4):1250–1264.

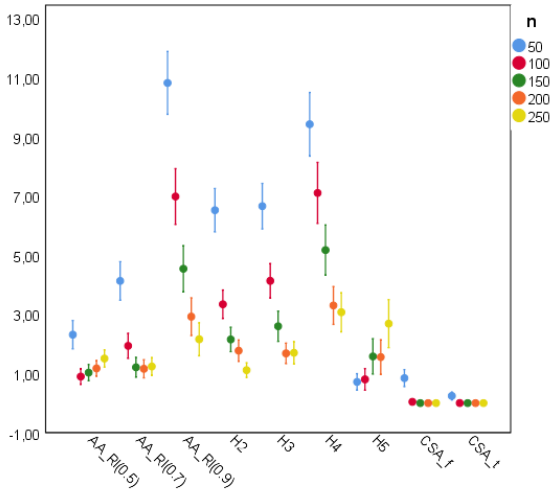
Shahvari, O. and Logendran, R. (2017). An Enhanced tabu search algorithm to minimize a bi-criteria objective in batching and scheduling problems on unrelated-parallel machines with desired lower bounds on batch sizes. *Computers & Operations Research*, 77:154–176.

Vallada, E. and Ruiz, R. (2011). A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. *European Journal of Operational Research*, 211(3):612–622.

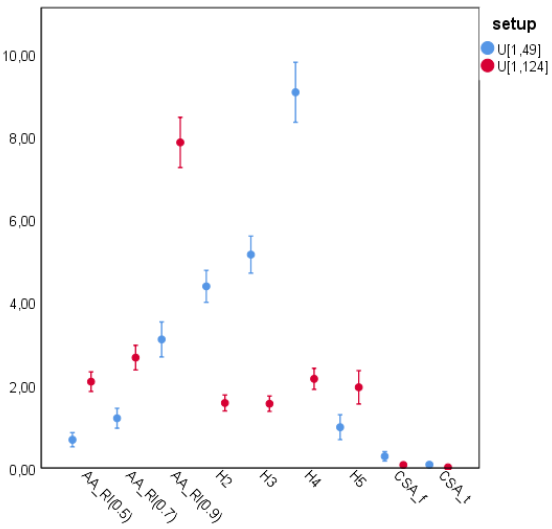
Ying, K.-C., Lee, Z.-J., and Lin, S.-W. (2012). Makespan minimization for scheduling unrelated parallel machines with setup times. *Journal of Intelligent Manufacturing*, 23(5):1795–1803.



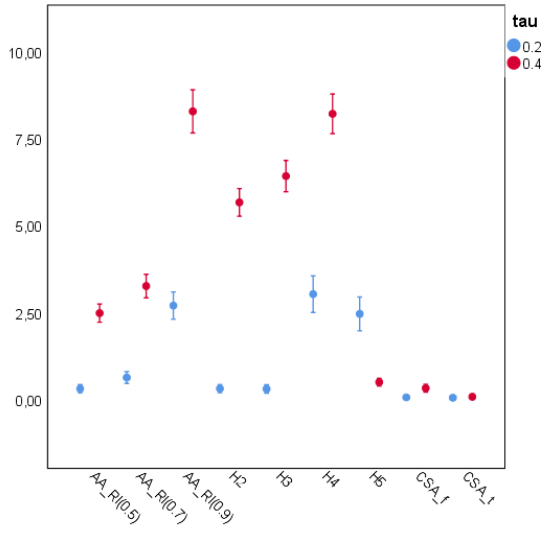
(a) Average RDI



(b) Average RDI depending on n



(c) Average RDI depending on *setup*



(d) Average RDI depending on τ

Figure 4: Tukey confidence intervals 95% Big Sizes Instances