Bi-objective parallel machine scheduling with additional resources during setups

Juan C. Yepes-Borrero[a,*], Federico Perea[b], Rubén Ruiz[a], Fulgencia Villa[a]

[a] *Grupo de Sistemas de Optimización Aplicada, Instituto Tecnológico de Informática, Ciudad Politécnica de la Innovación, Edifico 8G, Acc. B. Universitat Politècnica de València, Camino de Vera s/n, 46021, València, Spain.*
[b] *Dpto. Matemática Aplicada II, Universidad de Sevilla. Escuela Politécnica Superior, Calle Virgen de África 7, 41011 Sevilla, Spain*

Abstract

We present a bi-objective parallel machine scheduling problem with machine and job sequence dependent setup times, with the additional consideration of resources needed during setups. The availability of such resources is limited. This models many practical situations where setup times imply, for example, cleaning and/or the reconfiguration of productive equipment. These setups are performed by personnel, who are of course limited in number. The objectives considered are the minimization of the makespan and the minimization of the number of resources. Fewer available resources reduce production costs but inevitably increase the makespan. On the contrary, a greater number of resources increase costs but allow for more setups to be done in parallel and a reduced makespan. An algorithm based on iterated greedy approaches is proposed to search for the Pareto front of the problem. This algorithm is compared with state-of-the art methods adapted to the problem. Computational experiments, supported by statistical analyses, indicate that the proposed approach outperforms all other tested procedures.

Keywords: Scheduling, Parallel machine, Sequence dependent setup times, Multicriteria optimization, Resources

[*]Corresponding author. Tel: +34 96 387 70 00. Ext: 74914. Fax: +34 96 387 74 99
Email addresses: juancamiloyepes@gmail.com (Juan C. Yepes-Borrero), perea@us.es (Federico Perea), rruiz@eio.upv.es (Rubén Ruiz), mfuvilju@eio.upv.es (Fulgencia Villa)

1. Introduction

Operations research methods are being used more and more in the management of a variety of production planning and scheduling problems. Roughly speaking, scheduling problems consist of deciding which machines should process which jobs, and when such processing should be done. Out of the many scheduling problems proposed in the literature (flowshop, job shop, etc.), in this paper we deal with a generalized version of the unrelated parallel machine scheduling problem.

Many practical manufacturing settings can be modelled as a parallel machine scheduling problem. More specifically, multi-stage production systems frequently have a bottleneck stage that can be adequately solved with parallel scheduling models. Most of the literature, however, is centered around simplified models that lack many of the features needed for successful practical application. That is why more realistic models, and suitable algorithms to solve them, are needed. The basic unrelated parallel machine scheduling problem (referred to as UPM) deals with the scheduling of a set of jobs that need to be processed by exactly one of the available machines. Note that in the UPM, the processing time of a job depends on the machine to which it is assigned. This is already a generalization of the simpler identical parallel machine scheduling problem. The interested reader is referred to Fanjul-Peyro and Ruiz (2010) for a more detailed explanation of this UPM problem.

Machines are not the only resources employed in production shops. Many manufacturing environments require additional resources, typically machine operators, but also molds, jigs, fixtures, etc. For example, when workers are needed to be present during the processing of jobs, a new and more challenging problem than the UPM appears, called the Unrelated Parallel Machine scheduling problem with additional Resources in the Processes, denoted as UPMR-P.

Although Allahverdi (2015) shows that the literature including setup times is quite extensive, there still is a large portion of the scheduling literature that assumes that setup times are not sequence dependent, and are included in the processing times or just ignored. Nevertheless, setup times are ubiquitous in real settings. Both the UPM and UPMR-P assume that machines are ready to process a job right after they have finished processing the job before. This is not always realistic since machines may need reconfiguration, adjustments, cleaning, etc. between the processing of any two consecutive jobs. Furthermore, setup times often depend on the machine and on the job sequence. When setups are explicitly considered we have the UPMS (Unrelated Parallel Machine scheduling problem with Setups). A recent work is Fanjul-Peyro et al. (2019) which contains a detailed description of the UPMS and some algorithms to solve it.

When we jointly consider setup times and additional resources for carrying out the setups on machines we get a much more realistic problem. The number of the available additional resources is, in general, limited. This problem is called the Unrelated Parallel Machine scheduling problem with Setup times and additional Resources in the Setups (UPMSR-S). The UPMSR-S is an even more realistic extension of the parallel machine scheduling problem. This problem was first formulated and approached by Yepes-Borrero et al. (2020) and extended by Fanjul-Peyro (2020), who also consider the need of additional resources during job processing. The work we present now is an extension of these two papers, in the sense that we now consider the simultaneous minimization of both the makespan and the number of additional resources.

The vast majority of studies in the scheduling literature consider one single optimization objective. However, real world problems are, in general, multi-objective. In the particular case of the UPMSR-S, two clear and conflicting objectives arise: the minimization of the makespan (completion time of the schedule) and the minimization of the number of additional available resources. In the presence of setup times, increasing the number of available additional resources allows for a great number of setups to be carried out at the same time on different machines (as happens in real production shops). The immediate result is the reduction of the makespan. The contrary is also true, as fewer additional resources imply that some setup times might get delayed and machines may be put into idle mode due to a lack of resources (personnel). Therefore, a bi-objective approach is essential in order to study the best trade-off between the number of additional of resources and makespan. In this paper, we formally state a Bi-Objective Unrelated Parallel Machine scheduling problem with Setups and additional Resources in the Setups (BO-UPMSR-S) with two objectives: minimization of maximum completion time (makespan) and the minimization of the number of available additional resources. We also propose an algorithm to find feasible Pareto fronts for the BO-UPMSR-S. The Pareto front of a multi-objective optimization problem is the set of all feasible solutions which are not dominated by other feasible solutions.

Definition 1.1. *Given a multi-objective problem, a feasible solution S_a is said to be dominated by feasible solution S_b, if S_b is not worse than S_a in* *any objective, and $S_a \neq S_b$. A feasible solution that is not dominated by any other feasible solution is called* non-dominated. *The set consisting of all the non-dominated solutions is called the Pareto front.*

Considering that even the simplest specific case of the problem considered in this paper, i.e., the identical parallel machine scheduling problem without setups and only makespan minimization (problem denoted as $P//C_{\max}$ in the literature) is already $\mathcal{NP}$-Hard even for just two machines (Lenstra et al., 1977), it follows that the BO-UPMSR-S is also $\mathcal{NP}$-Hard. We therefore propose algorithms to efficiently find an approximation of the optimal Pareto front for the BO-UPMSR-S.

In short, the contribution of this paper is twofold:

- We propose a bi-objective unrelated parallel machine scheduling problem with setups and additional resources in the setups, in which we consider both the minimization of the makespan and the minimization of the number of available additional resources.
- We propose an algorithm to find feasible Pareto fronts for this problem, which is compared with adaptations of the best algorithms found in the multi-objective literature for related problems.

The rest of the paper is organized as follows. Section 2 reviews related literature. Section 3 formally states the problem addressed in this paper. This problem is solved using the algorithm proposed in Section 4. Afterwards, Section 5 briefly introduces other algorithms adapted from the literature. All of them are computationally compared in Section 6. Finally, conclusions and future research directions are given in Section 7.

2. Literature review

For many years, machine scheduling problems with setup times have been widely studied. Allahverdi (2015) surveyed problems with setup times. However, machine scheduling problems with additional resources have been the focus of far fewer studies. More specifically, as far as the authors know, multi-objective scheduling problems with setup times and additional resources assigned to the setups have yet to be studied. In this section, we summarize the most recent works on machine scheduling with setup times and some consideration of additional resources as well as other works on multi-objective machine scheduling.

Ruiz and Andrés-Romano (2011) propose a problem of unrelated parallel machine with setups whose duration depends on the assignment of resources. They do not consider limits on the available additional resources. Ruiz-Torres et al. (2007) study a uniform parallel machine problem without setup times where the processing times on the machines depend on the resources assigned (also without limits). Edis and Oguz (2012) use integer programming models to solve a parallel machine flexible resources problem where the resources may also speed up the processing times on machines. Edis and Ozkarahan (2012) propose solutions for a real-life resource-constrained parallel machine scheduling problem with integer programming and constraint programming models. Edis et al. (2013) present a complete review of parallel machine scheduling problems with additional resources. More recently, Bitar et al. (2016) propose a metaheuristic to solve an original unrelated parallel machine scheduling problem with auxiliary resources. Fanjul-Peyro et al. (2017) propose mathematical models and metaheuristics for the unrelated parallel machine problem without setup times, and with additional limited resources during the processing of jobs as a constraint. For the same problem, Villa et al. (2018) propose different heuristics, Vallada et al. (2019) propose metaheuristics methods and Fleszar and Hindi (2018) and Arbaoui and Yalaoui (2018) use constraint programming. Fanjul-Peyro (2020) study mathematical models and exact algorithms for unrelated parallel machine scheduling problems where resources might be needed for processing times, setup times or both and in all cases the objective is the minimization of the makespan. Finally, Yepes-Borrero et al. (2020) propose a GRASP algorithm to solve the problem with setup times and additional limited resources assigned to the setups, but only considering makespan as the optimization objective.

Multi-objective parallel machine scheduling problems are much less studied than the single objective variants. Hoogeveen (2005) presents a comprehensive review of multi-objective scheduling in which the most important results of different variations of scheduling problems are studied. Cochran et al. (2003) propose a genetic algorithm to solve a parallel machine scheduling problem with makespan and total weighted tardiness objectives. Bandyopadhyay and Bhattacharya (2013) modify the well-known NSGA-II algorithm of Deb et al. (2002) to solve a parallel machine scheduling problem with three objectives: minimization of total tardiness, minimization of the deterioration cost and minimization of makespan. Torabi et al. (2013) studied a multi-objective parallel machine problem with setup times and additional resources assigned to the processing times, seeking to minimize the total weighted flowtime, the total weighted tardiness and total machine load variation. Wang and Liu (2015) adapt the NSGA-II algorithm to solve a parallel machine scheduling problem with preventive maintenance planning over two resources (machines and molds) in order to minimize the unavailability of those resources. Rostami et al. (2015) use a mathematical model based on fuzzy chance-constrained programming to solve a multi-objective parallel machine scheduling problem with job deterioration and learning effect to minimize the makespan and the total earliness/tardiness. More recently, Zhou et al. (2018) propose a mathematical model and a differential evolution algorithm for a multi-objective parallel batch processing machine scheduling problem considering electricity consumption costs with the objective of minimizing the makespan and the total electricity cost.

There are other related studies considering different scheduling problems such as, the flowshop problem. Daniels and Chambers (1990) present heuristic procedures for a multi-objective flowshop problem in order to minimize the makespan and the total tardiness. Armentano and Arroyo (2004) propose a tabu search algorithm in order to minimize the makespan and the maximum tardiness in a flowshop scheduling problem. Framinan and Leisten (2008) propose a multi-objective iterated greedy search (MOIGS) for flowshop scheduling with minimization of makespan and flowtime. Minella et al. (2008) present a review of multi-objective algorithms for the flowshop scheduling problem. Minella et al. (2011) present a new algorithm called the Restarted Iterated Pareto Greedy (RIPG) to solve two multi-objective flowshop problems. The first problem consists of minimizing the makespan and total tardiness, and the second problem consists of minimizing the makespan and total flowtime. That research shows that the RIPG algorithm yields better results than the existing algorithms previously proposed for the flowshop scheduling problem. Dalfard and Mohammadi (2012) use two metaheuristics to solve a multi-objective flexible job shop problem with parallel machines and maintenance constraints. In addition, the authors propose a new mathematical model for the problem.

Although parallel machine scheduling problems have been extensively studied over many years, there are few works that deal with multi-objective parallel machine scheduling problems. Actually, to the best of our knowledge, there are no studies that consider setups and additional resources in multi-objective parallel machine scheduling problems. Considering that most real world scheduling problems are multi-objective and many industrial environments need both machines setups and additional resources, we believe that this paper studies a problem that could model real life scheduling problems.

3. Problem description

We now formally introduce the bi-objective unrelated parallel machine scheduling problem with resources in the setups (BO-UPMSR-S). In this problem we have the following sets:

- $N = \{1, ..., n\}$ is the set of jobs that must be processed. Jobs are indexed by j, k and l.
- $M = \{1, ..., m\}$ is the set of unrelated parallel machines available. Machines are indexed by i.
- $T = \{1, \ldots, t_{\max}\}$ is the set of time units. Time units are indexed by t. $t_{\max}$ is an upper bound for the makespan.

  Each machine can process one job at a time and if the processing of a job on a machine begins, that job must be finished without interruptions (i.e., no preemption exists). Jobs must be processed by exactly one out of the m available machines.

Additionally, we also have the following input data:

- $p_{ij} \geq 0$ is the processing time of job j on machine i. Machines are unrelated, meaning that the processing time of a job j may differ depending on the machine on which this job is processed.
- $s_{ijk} \geq 0$ is the setup time needed to prepare machine i between the processing of jobs j and k.
- $r_{ijk} \geq 0$ is the number of resources needed to do the setups between jobs j and k on machine i. Note that setup times and resource needs depend both on the machine and on the job processing sequence.

We note that the processing times, setup times, and the number of resources needed are deterministic. As mentioned in Section 1, in the BO-UPMSR-S there are two objectives: the minimization of the makespan (called $C_{\max}$) and the minimization of the maximum number of resources needed at any time during the processing sequence. Because in this problem minimizing one of the objectives might increase the other one, a multi-objective approach is necessary and a major question arises: which type of solution should we look for? Out of the several approaches that are valid in multi-objective optimization, in this paper we look for the Pareto front, consisting of non-dominated solutions defined in Section 1.

In order to present a mixed integer linear (MILP) formulation for the BO-UPMSR-S, we propose a bi-objective adaptation to the model proposed in Yepes-Borrero et al. (2020). Before defining the model, the following variables must be defined:

- Y_{ij}: Binary variable that takes value 1 if job j is processed on machine i, 0 otherwise.
- X_{ijk}: Binary variable that takes value 1 if job k is the successor of job j on machine i, 0 otherwise.
- H_{ijkt}: Binary variable that takes value 1 if the setup between the successive jobs j and k, on machine i, ends at instant t, 0 otherwise.
- $C_{\max}$: Maximum completion time of the schedule (makespan).
- $R_{\max}$: Maximum number of resources needed at any time during the processing sequence.

Additionally, the set $N_0 = N \cup \{0\}$ must be defined. In this set 0 is a dummy job and all machines start and end in job 0. We set $p_{i0} = s_{i0k} = s_{ik0} = r_{i0k} = r_{ik0} = 0$, $\forall\, i \in M$; $\forall\, k \in N_0$.

A model for the BO-UPMSR-S is:

$$\min\ (C_{\max}, R_{\max}) \tag{1}$$

$$\text{s.t.} \sum_{k \in N} X_{i0k} \leq 1,\ i \in M \tag{2}$$

$$\sum_{i \in M} Y_{ij} = 1,\ j \in N \tag{3}$$

$$Y_{ij} = \sum_{k \in N_0, j \neq k} X_{ijk},\ i \in M, j \in N \tag{4}$$

$$Y_{ik} = \sum_{j \in N_0, j \neq k} X_{ijk},\ i \in M, k \in N \tag{5}$$

$$\sum_{t \leq t_{\max}} H_{ijkt} = X_{ijk}, \forall i \in M, j \in N_0, k \in N, k \neq j \tag{6}$$

$$\sum_{t} tH_{ijkt} \geq \sum_{l \in N_0} \sum_{t \leq t_{\max}} H_{iljt}(t + s_{ijk} + p_{ij}) - \bar{M}(1 - X_{ijk}),$$

$$\forall\ i \in M, j \in N_0, k \in N, k \neq j \tag{7}$$

$$\sum_{i \in M, j \in N_0, k \in N, k \neq j, t' \in \{t, \ldots, t+s_{ijk}-1\}} r_{ijk} H_{ijkt'} \leq R_{\max}, \forall\ t \leq t_{\max} \tag{8}$$

$$\sum_{t \leq t_{\max}} tH_{ijkt} \leq C_{\max}, \forall i \in M, j \in N_0, k \in N_0, k \neq j \tag{9}$$

$$X_{ijk} \geq 0,\ Y_{ij} \geq 0, H_{ijkt} \in \{0, 1\}.$$

Objective (1) minimizes the makespan and the maximum number of resources needed in the solution. Constraints (2) ensure that on each machine i, at most one job is assigned to the first position of the sequence. Constraints (3) establish that each job j is assigned to one and only one machine. Constraints (4) enforce that each job j that is processed on machine i has one and only one successor k. Constraints (5) establish that each job k that is processed on machine i, has one and only one predecessor j. Constraints (6) set that for each machine i and for successive jobs j and k on machine i, the setup time between these jobs has to end in only one moment before $t_{\max}$. Constraints (7) enforce that the setup between successive jobs j and k on machine i, must end at the earliest. Constraints (8) impose that the number of resources used is lower than $R_{\max}$ at any instant of time. Constraints (9) ensure that the $C_{\max}$ must be greater or equal than instant of time when all the setups (including the final fictitious setup) have finished.

Since the simplified version of this model that only considers one objective is solvable only for instances of small size (see Yepes-Borrero et al., 2020), we do not test it in the computational results. However, we show this model in this section because it helps understanding the proposed BO-UPMSR-S.

The reader may note that in the BO-UMPSR-S we address three subproblems simultaneously:

1. Assignment problem: assignment of jobs to machines.
2. Sequencing problem: order of jobs on each machine.
3. Timing problem: the time at which setups between jobs start.

In the next section we propose an efficient algorithm for the BO-UPMSR-S.

4. The Truncated Restarted Iterated Pareto Greedy algorithm

As stated in Section 2, several algorithms have already been proposed to solve multi-objective scheduling problems. To solve the BO-UPMSR-S problem studied in this paper, we propose an algorithm based on the Restarted Iterated Pareto Greedy (RIPG) by Minella et al. (2011). The motivation behind this choice is the following: The original RIPG is a multi-objective adaptation of the Iterated Greedy (IG) originally proposed by Ruiz and Stützle (2007), that consists of iteratively destroying a solution partially, and reconstructing it using a greedy procedure. Afterwards, the solution obtained in the reconstruction phase may be improved by a local search procedure. The original IG has been recognized as a high performing method for flowshop scheduling and different variants in the literature. Minella et al. (2011) compared the RIPG against some of the best methods identified in the review of Minella et al. (2008), where 23 state-of-the-art multi-objective algorithms were comparatively evaluated. Later, Ciavotta et al. (2013) applied the RIPG to the multi-objective flowshop scheduling problem with setup times and compared the RIPG against 17 high performing optimizers. In all these studies RIPG produced the best observed results and this motivates us to select this methodology as a basis for the proposed procedure.

In the rest of the section we explain the different phases of the algorithm we propose, which is referred to as truncated RIPG (T-RIPG), and we also highlight its differences with respect to the original RIPG. It is important to note that the original RIPG was proposed for a multi-objective flowshop problem, seeking to minimize the makespan, flowtime and total tardiness. However, the problem we address in this paper is not a flowshop but a parallel machine problem, and the objectives treated are the makespan and the maximum number of additional resources needed. Therefore, we need to adapt T-RIPG significantly with respect to RIPG.

T-RIPG has four main phases: Initialization, greedy, timing/repairing and local search. Furthermore, there is a selection operator to choose which solution will be processed by the last three aforementioned phases. Finally, in order to avoid premature convergence, we repeat all phases with a restarted phase in which a new set of initial solutions is generated. Therefore, the T-RIPG consists of six phases in total. It has to be noted that throughout the process the algorithm keeps an archive of non-dominated solutions that is updated as new non-dominated solutions are found. Hereinafter this archive will be called the *working set*. Figure 1 shows the general outline of the proposed Truncated-RIPG (T-RIPG) procedure. The main parts of the algorithm are detailed in the following sections.

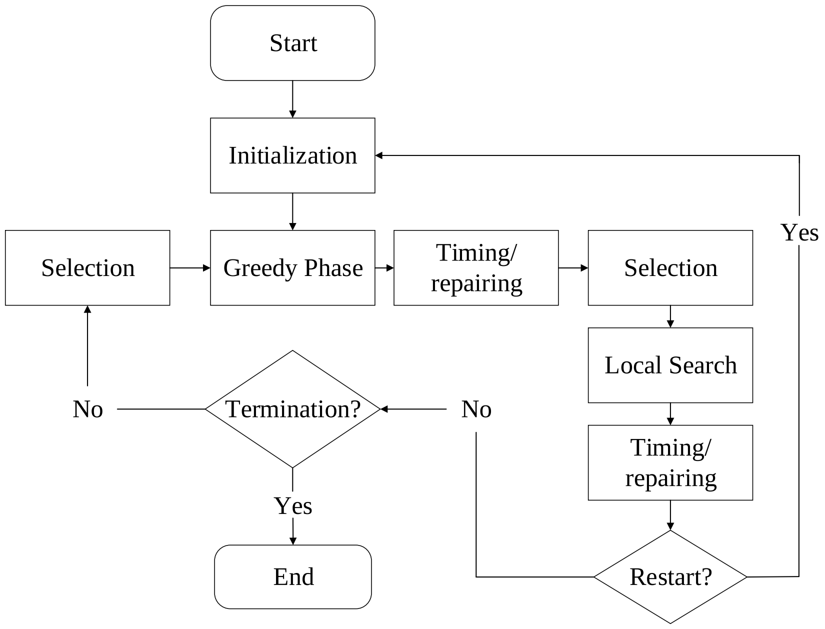


Figure 1: Flowchart of the proposed Truncated-RIPG (T-RIPG).

4.1. Initialization

In order to generate good initial solutions, we use the GRASP (*Greedy Randomized Adaptive Search Procedure*) algorithm proposed in Yepes-Borrero et al. (2020). This algorithm considers balanced solutions for both objectives (makespan and resources) and gives good results for the problem with makespan minimization considering the number of additional resources as a constraint. The main idea in this algorithm is to avoid sequences with large setup times or with a large consumption of resources, by calculating a value referred to as $\lambda_{i,j,k}$ defined as:

$$\lambda_{i,j,k} = C'_i + p_{ij} + (\theta_{s(i,k-1,k)} * \theta_{r(i,k-1,k)}) + (\theta_{s(i,k,k+1)} * \theta_{r(i,k,k+1)}) - (\gamma_{s(i,k)} * \gamma_{r(i,k)})$$

where C'_i is the partial completion time on machine i before the insertion of job j; p_{ij} is the processing time of job j on machine i; $\theta_{s(i,k-1,k)}$ (s is for setup) is the setup time between the job in position $k-1$ and job j inserted in position k on machine i. $\theta_{r(i,k-1,k)}$ (r is for resources) are the resources needed to do the setup $\theta_{s(i,k-1,k)}$. Similarly, $\theta_{s(i,k,k+1)}$ is the setup between job j (inserted in position k) and the job in position $k+1$ on machine i. $\theta_{r(i,k,k+1)}$ are the resources needed to do the setup $\theta_{s(i,k,k+1)}$. $\gamma_{s(i,k)}$ is the setup that is no longer done after the insertion of job j in position k on machine i. $\gamma_{r(i,k)})$ are the resources needed in the setup $\gamma_{s(i,k)}$.

The GRASP algorithm consists of inserting each non-assigned job into each position of the partial solution. At each insertion, $\lambda_{i,j,k}$ is calculated. When all non-assigned jobs are inserted into all possible positions, a job is chosen at random from a list with the best candidates (those with lower $\lambda_{i,j,k}$ values) and is assigned to the solution. That list's size varies depending on a value $\alpha \in [0, 1]$. The larger α, the larger the size of the list. In other words, larger α values add more randomness to the algorithm.

Additionally, two solutions are built by allowing for lower makespan values regardless of the consumption of resources. That is, by replacing $\theta_{r(i,k-1,k)} = \theta_{r(i,k,k+1)} = \gamma_{r(i,k)} = 1$ in the calculation of $\lambda_{i,j,k}$. In brief, two solutions are generated looking for a lower consumption of resources, and two solutions are generated which look for lower makespan. In order to have a variety of solutions, every time the algorithm is restarted (see Section 4.6), a new set of four solutions is generated as explained before, but more randomness is added to the GRASP algorithm (by increasing α).

4.2. Greedy phase

In this phase, a solution is destroyed and reconstructed generating new solutions. The first step of this phase consists of removing a group of d randomly selected jobs, and putting them in a set of pending jobs to be assigned called D. As with other parameters, the value of d affects the performance of the algorithm and will be calibrated later. Once the d jobs are removed, the second step consists of reconstructing the solution. In this step, the proposed T-RIPG substantially differs from the original RIPG (or general IG procedure). The original RIPG reinserts each job of the set D of pending jobs into all possible positions in the partial solution (the solution without the removed jobs), and generates a new set of partial solutions where each insertion of the selected job is a partial solution. Once the selected job is inserted into all possible positions, the new set of partial solutions is evaluated to eliminate the dominated partial solutions. This process is repeated until all removed jobs are reinserted and a set of non-dominated final solutions is obtained (when the last job is inserted into each position, each solution is a final solution). However, due to the nature of the problem studied in this paper, the partial solutions obtained during the reinsertion of the removed jobs may not provide enough information (this is further explained in the later Section 4.3). Furthermore, the high number of calculations needed to re-evaluate the consumption of resources at each insertion at each period of time from the point where the job is inserted up to when the last setup is finished, leads us to propose a new greedy process that works as follows: After removing the d jobs in the solution, instead of inserting only the first job of the d jobs, we insert each job into all possible positions of all machines in the partial solution. In order to find the best insertion without doing a complete (and costly) calculation of makespan and resources, we calculate at each insertion the same $\lambda_{i,j,k}$ explained in Section 4.1. As explained before, this value seeks to measure, in a single expression, the consumption of resources, the setup times and the makespan. After the insertion of all jobs into each possible position, we assign the best job to the best insertion found, that is, the insertion with lowest value of $\lambda_{i,j,k}$. Then, the assigned job is removed from the list of pending jobs to be assigned and the process is repeated until $d-1$ jobs are assigned.

When there is only one pending job to be assigned, this last job is reinserted into each possible position of the partial solution. Each job insertion generates a complete final solution that is added to the working set. All solutions obtained in this last step are processed in the timing/repairing phase. The differences between the original greedy phase and the proposed greedy phase are summarized in the following points:

1. At the end of first iteration (when the first of the d removed jobs is

reinserted), the greedy phase of RIPG algorithm generates $n - d + m$ new partial solutions, while the new greedy phase only generates one new partial solution.

2. In the greedy phase of the RIPG, the new partial solutions are evaluated in order to remove all dominated solutions. In the new greedy phase, as there is only one partial solution, there is no evaluation to do.
3. At any iteration i, the greedy phase of the RIPG generates $(n - (d - i) + m) \times ps$ new partial solutions, where ps is the number of partial solutions generated in the previous iteration. In the other hand, in the new greedy phase we only work with one partial solution during the first $d - 1$ iterations (until there is only one job in the list of pending jobs D). Finally, the new greedy generates $n - 1 + m$ new solutions by inserting the last job at each position of the (unique) partial solution.

As we mentioned before, the idea with this new greedy phase is to save time by not handling many partial solutions at each iteration. Due to space considerations, detailed results of the comparison between our greedy phase and the original greedy phase of RIPG algorithm are available as online materials.

It is important to note that in the first iteration all the solutions generated in the initialization phase go through this phase as shown in Figure 1. In the other iterations of the algorithm, only the solution selected by the selection operator (explained later in Section 4.4) is processed by this phase. Algorithm 1 shows a pseudocode of the greedy phase.

Algorithm 1: GreedyPhase

```
1 Remove at random $d$ jobs from the solution and put them in set $D$
2 while $size(D) > 1$ do
3     foreach $j \in D$ do
4         Insert the job into each possible position on all machines and
            calculate $\lambda_j = \min_{ik} \lambda_{ijk}$;
5     end
6     Assign the job with the lowest $\lambda_j$ and remove it from $D$;
7 end
8 Insert the remaining job from $D$ into each possible position on all
    machines and save all complete solutions in the working set;
9 Delete Dominated Solutions from the working set;
```

4.3. Timing/repairing phase

Another big difference between the original RIPG and the proposed T-RIPG is the inclusion of a repairing method. Here the beginning of the setups may be postponed by adding idle times on the machines, with the objective of reducing the consumption of resources (even if this increases the makespan). In order to illustrate, consider an example with $m = 2$ machines (indexed by letter i) and $n = 4$ jobs (indexed by letter j), with the processing times (p_{ij}), setup times (s_{ijk}), and resource needs (r_{ijk}) as given in Tables 1, 2, 3 respectively.

	j_1	j_2	j_3	j_4
i_1	6	2	3	5
i_2	3	4	8	5

Table 1: p_{ij} for an example with 4 jobs and 2 machines.

Machine i_1					*Machine* i_2				
	j_1	j_2	j_3	j_4		j_1	j_2	j_3	j_4
j_1	5	2	3	4	j_1	4	3	2	4
j_2	5	5	4	7	j_2	5	3	5	3
j_3	4	4	2	3	j_3	3	5	5	4
j_4	4	2	4	5	j_4	3	4	4	5

Table 2: Setup times (s_{ijk}) for an example with 4 jobs and 2 machines.

Machine i_1					*Machine* i_2				
	j_1	j_2	j_3	j_4		j_1	j_2	j_3	j_4
j_1	3	2	3	4	j_1	4	4	2	5
j_2	5	2	5	4	j_2	4	2	5	4
j_3	4	4	4	5	j_3	4	4	5	3
j_4	4	4	4	2	j_4	5	5	4	5

Table 3: Consumption of resources (r_{ijk}) for an example with 4 jobs and 2 machines.

Figure 2 shows a solution to this example before and after the repairing process. We observe in Figure 2(a) that 10 resources are needed in this solution, and the makespan is 11. However, as we observe in Figure 2(b), if we postpone the beginning of the setup on machine 2, the new makespan is 12, but this solution only needs 5 resources. It is important to note that postponing the beginning of a setup may not increase the makespan if the postponed setup is not on the makespan machine and the new completion time on the machine is lower than that of the makespan machine.

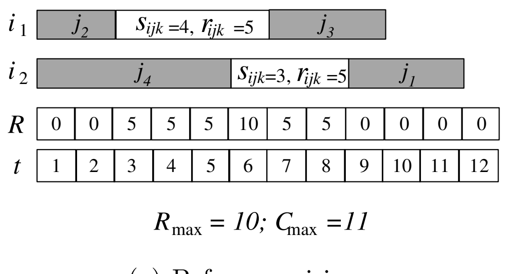
The first step of this process consists of deciding if the solution will be repaired or not. Since this is a costly process, not all solutions obtained in the greedy phase undergo this procedure. We define a probability p for a solution to be processed by the repairing algorithm. Of course, p is another parameter of the algorithm and will be calibrated. The repairing algorithm is a modification of the one proposed in Yepes-Borrero et al. (2020). The adaptation proposed consists of assigning the maximum number of resources $R_{\max}$ that the solution may need, and repairing the sequence in order to make it feasible with this number of resources. The maximum number of resources $R_{\max}$ is chosen at random from a uniform distribution between r_l and r_u, where r_l is the maximum r_{ijk} in the original solution (the solution before entering this phase), and r_u is the number of resources needed to execute the original solution without idle times. Once the maximum number of resources is chosen, the repairing process is the same as the original in Yepes-Borrero et al. (2020). We evaluate the consumption of resources during all time periods and if the consumption of resources is greater than the maximum allowed, we postpone the beginning of the setup on the machine that has the latest setup start, among the machines that are processing setups during that time period. The postponement is carried out until another machine ends a setup and its resources are available. That process is repeated until the resource consumption does not exceed $R_{\max}$ during any time period. Finally, all solutions obtained in this last step are added to the working set and the dominated ones are removed. Algorithm 2 shows a pseudocode of the repairing phase.

4.4. Selection phase

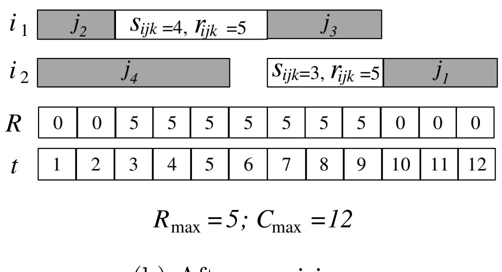
In this phase, the solution that will be processed by the remaining phases of the algorithm is selected from within the current working set. To select such a solution, we use the selection operator proposed in the original RIPG of Minella et al. (2011), named Modified Crowding Distance Assignment (MCDA). This method is based on the Crowding Distance operator proposed in Deb et al. (2002). We choose this operator because it has been successfully tested in different studies on related problems (see Minella et al., 2011, Ciavotta et al., 2013, Zhang et al., 2020). In short, the idea in the MCDA

Algorithm 2: RepairingPhase

```
1 Choose the maximum number of resources $R_{\max}$
2 for $t < C_{\max}$ do
3     Evaluate the consumption of resources at time period $t$;
4     if consumption of resources $> R_{\max}$ then
5         Postpone the beginning of the setup on the machine with the
            latest starting setup;
6         Update $C_{\max}$;
7     end
8 end
9 Delete Dominated Solutions from the working set
```



(a) Before repairing.



(b) After repairing.

Figure 2: Example of the proposed repairing mechanism.

and in the original Crowding Distance operator is to favor the selection of the most isolated solutions in the same level of dominance, i.e., solutions that have fewer nearby solutions in the objective space. We select the most isolated solution and later explore its neighborhood in order to avoid having large gaps in the resulting Pareto fronts. To avoid the repeated selection of the same solution at successive iterations, MCDA assigns a fitness value affected by a selection counter to each solution in the working set. Every time a solution is selected, the selection counter increases and the probability of this solution being selected in the next iteration diminishes.

4.5. Local search

Once the repairing phase is finished, the selection operator chooses the solution that will be processed by the local search among the solutions in the working set. The local search proposed consists of a very simple and fast method. We remove, at random, one job from the machine that determines makespan and we reinsert it into all possible positions on the other machines. As in the greedy phase, each reinsertion generates a new solution which will be repaired with the procedure explained in Section 4.3. As in the previous phases, all solutions obtained in this last step are added to the working set and the dominated ones are removed. It is important to notice that the solutions that are processed in the local search must be justified to the left if the solution has idle times. The local search process is repeated until ℓ iterations pass without improvements in the working set, ℓ being another parameter that will be calibrated. Algorithm 3 shows the general procedure of the local search.

4.6. Restart

As in the original RIPG, the restart phase consists of restarting the algorithm from the initialization phase and creating a new set of initial solutions in order to increase the variability of solutions. Before restarting the algorithm, the working set is saved and a new set of solutions is generated with more randomness than the solutions generated in the previous iteration (by increasing the α value explained in Section 4.1). All new solutions are processed by the greedy phase and the timing/repairing phase as in the first iteration of the algorithm. The solutions obtained are added to the working set and all solutions are evaluated in order to remove the dominated ones. The process continues with the selection procedure. In order to determine when the restart must be done, we propose a simple counter of successive

Algorithm 3: Local search general procedure.

```
1  $LS\_iterations\_without\_improvement = 0$;
2  $Partial\_Working\_Set \leftarrow Working\_Set$;
3  while $LS\_iterations\_without\_improvement < \ell$ do
4      $M^* \leftarrow$ Makespan machine;
5      Remove at random one job from $M^*$;
6      Insert the removed job into each possible position on all machines
         and save all complete solutions in $Partial\_Working\_Set$;
7      Delete Dominated Solutions from $Partial\_Working\_Set$;
8      if $Partial\_Working\_Set \neq Working\_Set$ then
9          $LS\_iterations\_without\_improvement = 0$;
10         $Working\_Set \leftarrow Partial\_Working\_Set$;
11     else
12         $LS\_iterations\_without\_improvement + +$;
13     end
14 end
```

iterations without improvements in the working set. When the counter is reached, the algorithm is restarted. This parameter is called q and will be calibrated in Section 6.1.

The overall Truncated-RIPG (T-RIPG) procedure is shown in Algorithm 4, where *ExecutionTime* is the running time of the algorithm and *TimeLimit* is the time limit in which the algorithm stops.

5. Other multi-objective algorithms

In order to evaluate the performance of the proposed T-RIPG algorithm, we compare it with other multicriteria algorithms in the literature, adapted to the BO-UPMSR-S studied in this paper. The algorithms adapted and re-implemented are the well known NSGA-II proposed by Deb et al. (2002), used in many studies as a baseline in multi-objective optimization, the MOIGS algorithm proposed by Framinan and Leisten (2008) and the original RIPG proposed by Minella et al. (2011). These last two methods are known to produce state-of-the-art results for some hard scheduling problems. We add to all re-implemented algorithms the timing/repairing phase explained in 4.3 as it greatly improves results and in order to have a fair comparison. We now briefly describe all these methods. The interested reader is referred to

Algorithm 4: Truncated-RIPG (T-RIPG) general procedure.

```
while $ExecutionTime < TimeLimit$ do
    $InitialSet := InitilizationPhase$;
    foreach $Solution \in InitialSet$ do
        GreedyPhase($Solution$);
        Timing/RepairingPhase($Solution$);
    end
    Delete Dominated Solutions from the working set;
    $SelectedSolution := SelectionPhase$;
    while $LS\_iterations\_without\_improvement < \ell$ do
        LocalSearch($SelectedSolution$);
    end
    while $Iterations\_without\_improvement < q$ do
        $SelectedSolution := SelectionPhase$;
        GreedyPhase($SelectedSolution$);
        Timing/RepairingPhase($SelectedSolution$);
        Delete Dominated Solutions from the working set;
        $SelectedSolution := SelectionPhase$;
        while $LS\_iterations\_without\_improvement < \ell$ do
            LocalSearch($SelectedSolution$);
        end
    end
end
```

the corresponding references in order to obtain complete details on these algorithms.

5.1. NSGA-II algorithm

The proposed adaptation of the well known NSGA-II follows the same idea proposed in Deb et al. (2002). An initial population of s solutions is generated. The crossover operator for parallel machines proposed in Vallada and Ruiz (2011) is used to generate new solutions (called offspring). Once all offspring are generated, the solutions are evaluated and sorted into different non-domination levels. The first level has all non-dominated solutions, the second level has all solutions dominated by the solutions in the first level but not dominated by other solutions. That sorting is repeated until all solutions are assigned into a level. To generate the next generation of initial solutions, the best s solutions are taken from the previous step. The first solutions assigned to the new generation are those that belong to the first level. This process is repeated with the other non-dominated levels until s solutions are assigned. If the number of solutions in the level is larger than s minus the solutions already assigned, a selection operator is used to choose a solution until the next generation is completed. With this new population, the process is repeated until the stopping criteria is satisfied.

5.2. MOIGS algorithm

The implemented adaptation of the MOIGS algorithm has some differences in respect to the original algorithm proposed in Framinan and Leisten (2008). Similarly to the original method, two initial solutions are generated. For each solution, d elements are removed and reinserted into each position in the solution. Each insertion generates a new partial solution. When a job is reinserted into each position in the solution, all partial solutions are evaluated and the dominated ones are removed. The next job is assigned to each position in all new partial solutions. This process is repeated until all jobs are assigned, the final solutions are evaluated and the dominated solutions are removed, finishing the first iteration of the algorithm. The original MOIGS algorithm repeats the same process with the new group of non-dominated solutions. Since the resource evaluation requires the calculation of the consumption of resources at each time period, when we have many solutions this process is very slow. The proposed adaptation consists of, once the first iteration is finished, using the greedy phase proposed for the T-RIPG in Section 4.2. At each iteration the set of non-dominated solutions is updated and the process is repeated with this new set of solutions.

5.3. RIPG algorithm

In order to evaluate the relevance of the new Greedy phase proposed for the T-RIPG, we also adapt the original RIPG algorithm presented in Minella et al. (2011). The difference between the Greedy phase of the RIPG and the Greedy phase of the T-RIPG is explained in Section 4.2. The other phases in the RIPG (Initialization, Selection, Local search and Restart) were implemented as explained in Section 4. Note that this is not the original RIPG, but an adaptation to the problem at hand. Recall that RIPG was proposed for the permutation flowshop with multiple objectives, not for parallel machine scheduling, let alone the addition of resources.

6. Experimental analysis

The comparison of results from different algorithms in a multi-objective optimization setting is far from trivial. Solutions obtained by an algorithm for a multi-objective problem are actually a set of non-dominated solutions and the comparison between two different methods becomes difficult. In this paper, we adopt the same two performance indicators studied in Minella et al. (2011) and in Zhang et al. (2020).

The first indicator is the so-called Unary Epsilon Indicator (I^1_ϵ) presented in Knowles et al. (2005). This indicator measures the distance between the Pareto front obtained by a given algorithm and the optimal or reference Pareto front. The reference Pareto front is the front built with all non-dominated solutions from all tested methods, that is, the best known Pareto front for a given instance.

Before calculating I^1_ϵ, the objective values are normalized into values in the interval (1,2). Lower values of I^1_ϵ indicate that the obtained Pareto front is close to the optimal or reference Pareto front. The Unary Epsilon indicator is calculated as follows: $I^1_\epsilon = I_\epsilon(A,P) = \max_{p\in P} \min_{a\in A} \max_{1\leq g\leq 2}\{f'_g(a)/f'_g(p)\}$ where:

- A is the Pareto front obtained by a given algorithm.
- P is the optimal or reference Pareto front.
- $f'_g(a)$ is the normalized value in objective g for the solution $a \in A$.

- $f'_g(p)$ is the normalized value in objective g for the solution $p \in P$.

The second indicator is the unary version of the Hypervolume indicator (I_H) proposed in Zitzler et al. (2003). This indicator measures the hypervolume (the area in the case of two objectives) of the space dominated by a Pareto front. To calculate this indicator, the objective values are normalized into values in the interval (0,1), and the reference point necessary to close the area is set to 1.2. Therefore, the maximum I_H value that can be obtained by $1.2 \times 1.2 = 1.44$. As suggested by Minella et al. (2011), we employ both indicators as when one indicator contradicts the other when comparing two algorithms, it means that there is no strong dominance of either algorithm over the other.

To evaluate the algorithms, we carry out an extensive computational and statistical study and we compare the algorithms proposed in previous sections. First, all algorithms are calibrated on a calibration set of instances. Afterwards, all algorithms are compared on an evaluation set of instances. The benchmark consists of a set instances with different sizes. We use as a base, the large instances (without resources) of Vallada and Ruiz (2011). This set of instances was generated by varying the number of jobs (n), number of machines (m), the setup times (s_{ijk}) and the consumption of resources (r_{ijk}), as follows:

- n varies among $\{50, 100, 150, 200, 250\}$.
- m varies among $\{10, 15, 20, 25, 30\}$.
- The setup times s_{ijk} are generated by four different uniform random distributions across the ranges $\{1-9\}$, $\{1-49\}$, $\{1-99\}$ and $\{1-124\}$. We have chosen these intervals since they model different realistic settings and have been previously used in studies of parallel machine scheduling with setup times (see Vallada and Ruiz, 2011, Diana et al., 2015, Yepes-Borrero et al., 2020).
- The consumption of resources r_{ijk} is generated by two different uniform random distributions in the ranges $\{1 - m\}$ and $\{1 - 5m\}$.
- The processing times p_{ij} are generated by a uniform random distribution between 1 and 99. Despite that this interval of variation implies a great difference between processing times, we use it as they have been used in other scheduling research articles before.

All combinations of the previous instance factors result in a total of $5 \times 5 \times 4 \times 2 = 200$ possibilities. We generate 5 instance replicates for each possible combination, yielding a total of 1,000 instances for the evaluation set, and one additional replicate of each possible combination for the calibration set (for a total of 200 calibration instances).

The methods are coded in Microsoft Visual Studio 2019 using C#. All algorithms share common codes and functions. The experiments are run on virtual machines with 2 virtual processors and 8GBytes of RAM memory each under Windows 10 Enterprise 64 Bits. Machines are virtualized in an OpenStack virtualization framework supported by 12 blades, each one with four 12-core AMD Opteron Abu Dhabi 6344 processors running at 2.6 gigahertzs and 256 gigabytes of RAM, for a total of 576 cores and 3 terabytes of RAM. Note that no parallel computing is carried out and virtual machines are used to distribute the testing load.

6.1. Calibration of the T-RIPG algorithm

The proposed T-RIPG algorithm depends on the following four parameters:

- Parameter d refers to the number of jobs to remove in the greedy phase (Section 4.2). We have tested this parameter at four levels: $d = 0.05 \times n$, $d = 0.1 \times n$, $d = 4$ and $d = 5$. The fixed levels $d = 4$ and $d = 5$ were chosen because of the results obtained in different works that use Iterated Greedy algorithms (Ruiz et al., 2019, Zhang et al., 2020, Minella et al., 2011).

- Parameter p denotes the probability with which a solution is repaired in the repairing/timing phase (Section 4.3). We have tested five levels: $p = 0$, $p = 0.25$, $p = 0.5$, $p = 0.75$ and $p = 1$. Note that $p = 0$ means that there is no repairing phase, which will be used as a witness level, to check the contribution of such a phase to the complete algorithm. Note also that $p = 1$ means that all solutions obtained in the greedy phase are repaired.

- Parameter ℓ is the number of iterations without improvements in the Pareto front before stopping the local search (Section 4.5). We have tested four levels: $\ell = 0$, $\ell = 50$, $\ell = n$ and $\ell = 2 \times n$. Again, $\ell = 0$ is a witness level in which there is no local search.

- Parameter q is the number iterations without improvements in the Pareto front before restarting the algorithm (Section 4.6). We have tested this parameter at four levels: $q = 0$, $q = 50$, $q = n$ and $q = 2 \times n$. The level $q = 0$ is again a witness indicating no restarting phase.

We propose a full factorial design of experiments (DOE), with all combinations of factors and levels in order to evaluate the different algorithm configurations. That gives a total of $4 \times 5 \times 4 \times 4 = 320$ possible configurations. Each configuration is tested in the 200-instance calibration set. A total of 64,000 approximations of the Pareto front are obtained as a result. Each configuration has the same stopping criterion that depends on the number of jobs in the instance: $t = n$ seconds. In total each one of the 320 configurations needs 30,000 seconds to run all instances, for a total of 9,600,000 seconds, or a bit more than 111 days of CPU time to complete the calibration. In order to statistically compare all configurations, an analysis of variance (ANOVA) is applied (Montgomery (2012)). Note that all necessary hypotheses are checked, with no problems in the residuals found. The response variables of the ANOVA are the *Unary Epsilon* (I^1_ϵ) and the *Hypervolume indicators* (I_H). The results indicate that all factors are statistically significant. Detailed ANOVA results are available as online materials.

Figures 3 and 4 show the means plot of I_H and I^1_ϵ with Tukey's Honest Significant Difference (HSD) 95% confidence intervals for the different calibrated parameters. When HSD intervals do not overlap there are significant differences between the groups (observed means in our case). We can easily see the relevance of the timing/repairing phase, the local search and the restart as $p = 0$, $\ell = 0$ and $q = 0$ are significantly worse than the other levels. Following the results of the experiment, the parameters were set to the combination yielding the best average and statistically significant results: $d = 4$, $p = 1$, $\ell = 50$ and $q = 50$.

In order to have a fair comparison, the other algorithms were also calibrated with similar comprehensive and CPU-intensive calibrations. Due to space considerations, detailed results are not shown here, although they are available as online materials.

6.2. Computational comparisons among algorithms

We now compare the proposed T-RIPG with the other algorithms in the 1,000-instance evaluation set. As in the calibration, all algorithms have the same stopping criteria with a CPU time limit of $t = n$ seconds. Table 4 shows the average values of the Hypervolume (I_H) and Unary epsilon indicators (I^1_ϵ). We observe that on average, the proposed T-RIPG yields better results

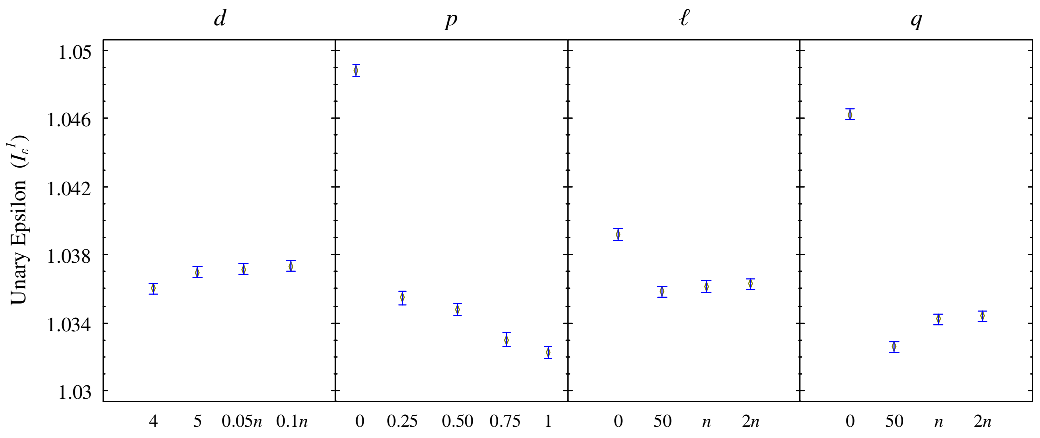


Figure 3: Unary epsilon (I_ϵ^1) Means plots with Tukey's Honest Significant Difference (HSD) 95% confidence intervals for all factors in T-RIPG calibration.

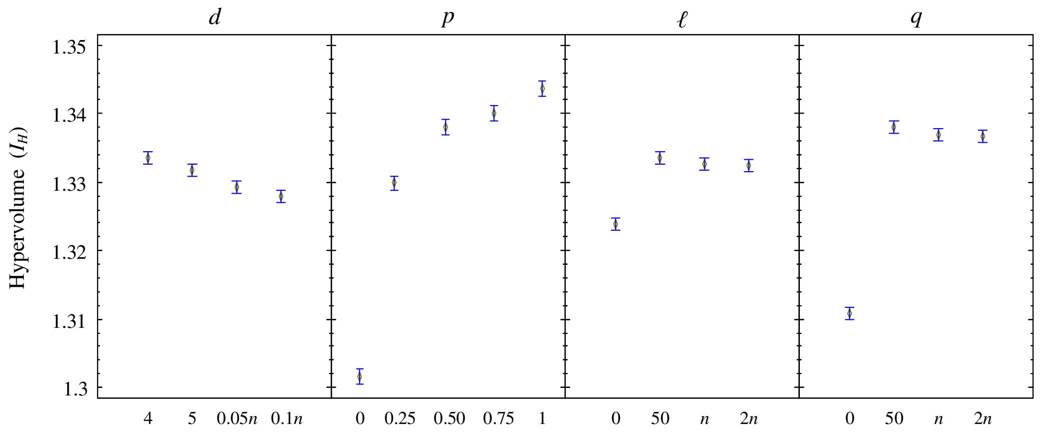


Figure 4: Hypervolume (I_H) Means plots with Tukey's Honest Significant Difference (HSD) 95% confidence intervals for all factors in T-RIPG calibration.

for both indicators. We also observe that the second best algorithm in both indicators is the original RIPG. It is interesting that for the MOIGS algorithm the results in instances with few jobs are worse than the other algorithms for both indicators. However, when the size of the instances increases, this algorithm yields better results than the RIPG and the NSGA-II algorithms for I_H. That may be due to the restarting phase, because in small instances, the algorithms converge prematurely, while in large instances, the space of solutions is larger.

It is important to remind that the values of each objective was normalized before calculating I_H and I_ϵ^1. Therefore, the differences among algorithms may not be appreciated by comparing final values of each indicator. In order to validate if the observed differences in I_H and I_ϵ^1 are statistically significant (significance level 0.05), an ANOVA is applied with both I_H and I_ϵ^1 as response variables and the algorithm as single factor. Figure 5 shows the means plot of I_H and I_ϵ^1 with Tukey's HSD 95% confidence intervals for the different algorithms. In terms of Hypervolume indicator, we can easily note that our T-RIPG is significantly better than the other algorithms. Moreover, for the Unary epsilon indicator, even if the difference between T-RIPG and RIPG is not as big as in the Hypervolume, the HSD intervals do not overlap (RIPG HSD interval = $[1.03218; 1.02339]$, T-RIPG HSD interval = $[1.02320; 1.01449]$), that is, the difference between these algorithms is statistically significant. The difference, considering that both indicators are normalized, is not only statistically significant but big as well.

Figures 6 and 7 show the average values of Hypervolume (I_H) and Unary epsilon indicators (I_ϵ^1) as a function of the instance size. From these graphs and Table 4, we draw the following conclusions:

- The proposed T-RIPG algorithm performs better than all the other algorithms for all groups of instances with the Hypervolume indicator.
- The proposed T-RIPG performs better than MOIGS and NSGA-II for all groups of instances with the Unary epsilon indicator. However, RIPG yields slightly better results in a few groups. In these cases, we often observe conflictive I_H and I_ϵ^1 values as per Table 4, indicating that for these cases no algorithm strongly dominates the other.

	T-RIPG		RIPG		MOIGS		NSGA-II	
	I_H	I_ϵ^1	I_H	I_ϵ^1	I_H	I_ϵ^1	I_H	I_ϵ^1
50 x 10	**1.2251**	**1.0291**	1.1056	1.0376	0.8479	1.2178	1.0678	1.1327
50 x 15	**1.2446**	**1.0230**	1.1050	1.0361	0.9096	1.1868	1.1176	1.0792
50 x 20	**1.2774**	**1.0226**	1.1818	1.0317	0.9678	1.1738	1.1711	1.0808
50 x 25	**1.2733**	**1.0271**	1.1539	1.0338	0.9149	1.1894	1.1618	1.0788
50 x 30	**1.2914**	**1.0325**	1.1693	1.0376	0.9444	1.2206	1.1993	1.0616
100 x 10	**1.2000**	**1.0405**	0.9812	1.0432	0.9765	1.1514	0.8938	1.1142
100 x 15	**1.2651**	**1.0266**	1.1095	1.0361	1.0319	1.1393	1.0727	1.0800
100 x 20	**1.2595**	**1.0206**	1.1388	1.0263	1.0820	1.0972	1.1019	1.0775
100 x 25	**1.2731**	**1.0192**	1.1355	1.0234	1.0667	1.0903	1.1316	1.0652
100 x 30	**1.2702**	**1.0151**	1.1113	1.0277	1.0467	1.1011	1.1200	1.0610
150 x 10	**1.2473**	**1.0251**	1.0163	1.0415	1.0703	1.1094	0.7789	1.1389
150 x 15	**1.2660**	**1.0139**	1.0352	1.0352	1.1133	1.1059	0.8600	1.1052
150 x 20	**1.2911**	1.0214	1.1360	**1.0119**	1.1312	1.0786	1.0175	1.0728
150 x 25	**1.3125**	**1.0105**	1.1578	1.0264	1.1864	1.0763	1.0915	1.0628
150 x 30	**1.2775**	**1.0121**	1.1074	1.0155	1.1285	1.0741	1.0270	1.0670
200 x 10	**1.3039**	**1.0121**	1.0429	1.0438	1.1305	1.1189	0.6771	1.2097
200 x 15	**1.3247**	**1.0167**	1.1410	1.0210	1.1816	1.0895	0.8482	1.1403
200 x 20	**1.3105**	**1.0103**	1.1070	1.0227	1.1728	1.0787	0.8531	1.1352
200 x 25	**1.3083**	**1.0133**	1.1501	1.0142	1.1842	1.0640	0.9275	1.0880
200 x 30	**1.3226**	**1.0091**	1.1338	1.0173	1.2035	1.0580	0.9521	1.0946
250 x 10	**1.3044**	**1.0162**	1.0216	1.0419	1.1819	1.1144	0.5984	1.2562
250 x 15	**1.3191**	**1.0157**	1.1191	1.0235	1.2097	1.0837	0.7590	1.1796
250 x 20	**1.2959**	**1.0167**	1.1250	1.0204	1.1842	1.0827	0.8059	1.1481
250 x 25	**1.3196**	1.0136	1.1707	**1.0122**	1.2546	1.0528	0.9227	1.1041
250 x 30	**1.3448**	**1.0103**	1.1722	1.0127	1.2249	1.0742	0.9404	1.1073
Average	**1.2851**	**1.0189**	1.1131	1.0277	1.0938	1.1132	0.9639	1.1096

Table 4: Computational results for all algorithms.

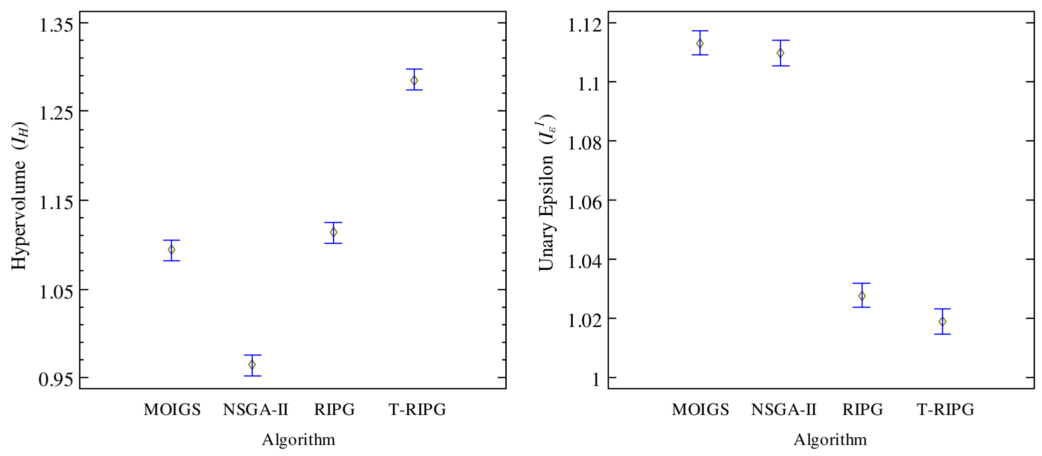


Figure 5: Hypervolume and Unary epsilon Means plots with Tukey's Honest Significant Difference (HSD) 95% confidence intervals for all tested algorithms.

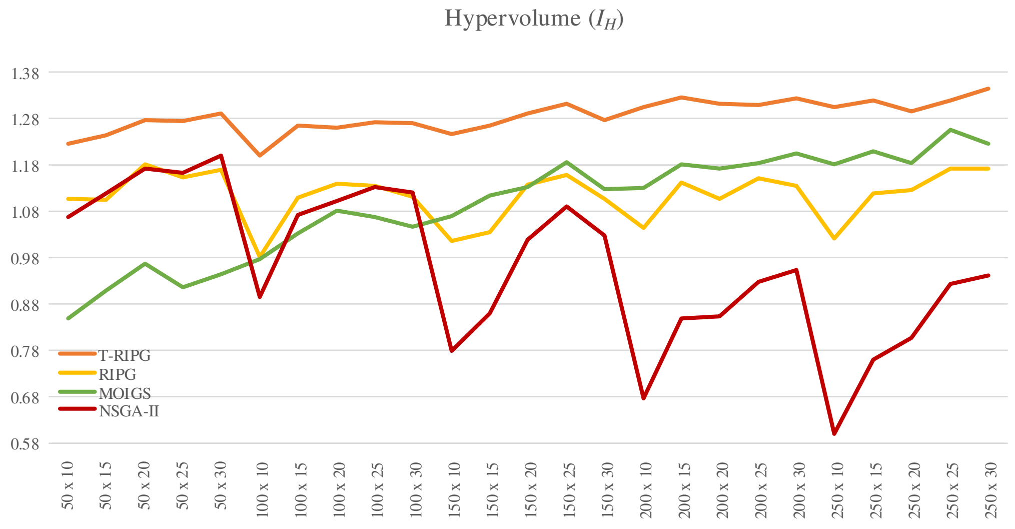


Figure 6: Hypervolume (I_H) for the tested algorithms as a function of the instance size.

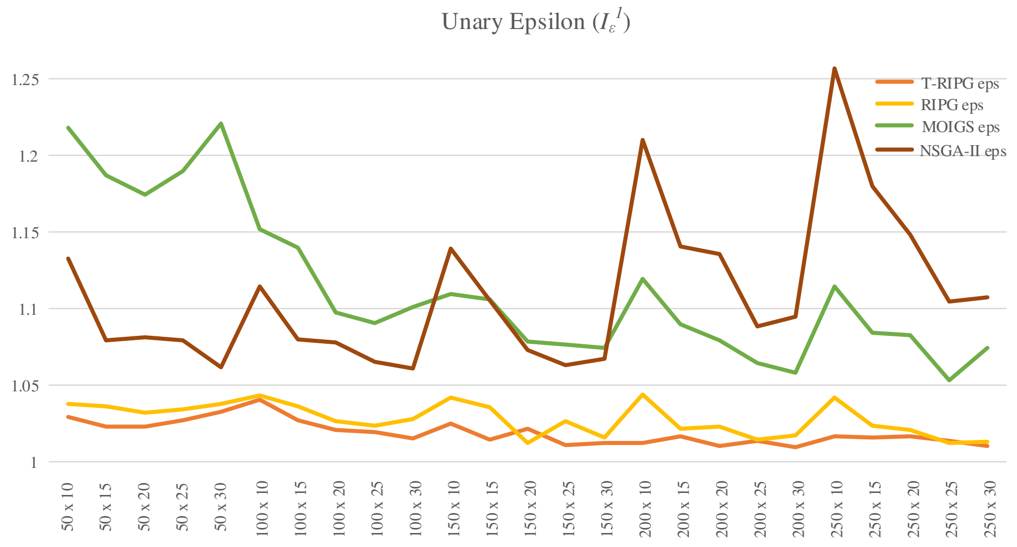


Figure 7: Unary epsilon for the tested algorithms as a function of the instance size.

7. Conclusions and future work

In this paper we have presented an efficient algorithm to solve the Bi-Objective Unrelated Parallel Machine scheduling problem with Setups and additional Resources in the Setups (BO-UPMSR-S). This problem is a generalized setting that incorporates many practical situations easily encountered in real production shops. A comprehensive computational campaign together with a detailed statistical analysis, based on the results obtained over a large benchmark, is presented to compare the proposed algorithm, called Truncate Restarted Iterated Pareto Greedy algorithm (T-RIPG), with other state-of-the-art adapted multi-objective algorithms. All algorithms were calibrated, and their parameters were set to those levels that produced the best results. That calibration shows the relevance of each phase of the algorithm. Of particular relevance is the repairing mechanism, a novel procedure aimed at greatly reducing resource consumption in constructed schedules by inserting idle times into machines before setups start, to avoid resource overload. This, together with fine tuned operators taken from the literature, results in a state-of-the-art approach. The proposed algorithm yields better results than the other algorithms for both performance indicators studied in this paper. Among the other algorithms adapted in this paper, in small instances, the RIPG algorithm outperforms the MOIGS algorithm. However, in large instances, the MOIGS algorithm performs better than RIPG. Future research on this topic will focus on adapting these algorithms to other multi-objective scheduling problems.

References

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

Arbaoui, T. and Yalaoui, F. (2018). Solving the unrelated parallel machine scheduling problem with additional resources using constraint programming. In Nguyen, N. T., Hoang, D. H., Hong, T.-P., Pham, H., and Trawiński, B., editors, *Intelligent Information and Database Systems*, pages 716–725, Cham. Springer International Publishing.

Armentano, V. and Arroyo, J. E. (2004). An application of a multi-objective tabu search algorithm to a bicriteria flowshop problem. *Journal of Heuristics*, 10:463–481.

Bandyopadhyay, S. and Bhattacharya, R. (2013). Solving multi-objective parallel machine scheduling problem by a modified NSGA-II. *Applied Mathematical Modelling*, 37(10):6718–6729.

Bitar, A., Dauzère-Pérès, S., Yugma, C., and Roussel, R. (2016). A memetic algorithm to solve an unrelated parallel machine scheduling problem with auxiliary resources in semiconductor manufacturing. *Journal of Scheduling*, 19(4):367–376.

Ciavotta, M., Minella, G., and Ruiz, R. (2013). Multi-objective sequence dependent setup times flowshop scheduling: a new algorithm and a comprehensive study. *European Journal of Operational Research*, 227(2):301–313.

Cochran, J. K., Horng, S.-M., and Fowler, J. W. (2003). A multi-population genetic algorithm to solve multi-objective scheduling problems for parallel machines. *Computers & Operations Research*, 30(7):1087–1102.

Dalfard, V. M. and Mohammadi, G. (2012). Two meta-heuristic algorithms for solving multi-objective flexible job-shop scheduling with parallel machine and maintenance constraints. *Computers & Mathematics with Applications*, 64(6):2111–2117.

Daniels, R. L. and Chambers, R. J. (1990). Multiobjective flow-shop scheduling. *Naval Research Logistics*, 37(6):981–995.

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197.

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. Recent Advancements in Hybrid Artificial Intelligence Systems and its Application to Real-World Problems Progress in Intelligent Systems Mining Humanistic Data.

Edis, E. and Ozkarahan, I. (2012). Solution approaches for a real-life resource-constrained parallel machine scheduling problem. *International Journal of Advanced Manufacturing Technology*, 58.

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

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

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

Framinan, J. M. and Leisten, R. (2008). A multi-objective iterated greedy search for flowshop scheduling with makespan and flowtime criteria. *OR Spectrum*, 30(11):787–804.

Hoogeveen, H. (2005). Multicriteria scheduling. *European Journal of Operational Research*, 167:592–623.

Knowles, J., Thiele, L., and Zitzler, E. (2005). A tutorial on the performance assessment of stochastic multiobjective optimizers. Third International Conference on Evolutionary Multi-Criterion Optimization (EMO 2005).

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

Minella, G., Ruiz, R., and Ciavotta, M. (2008). A review and evaluation of multiobjective algorithms for the flowshop scheduling problem. *Informs Journal on Computing*, 20:451–471.

Minella, G., Ruiz, R., and Ciavotta, M. (2011). Restarted iterated pareto greedy algorithm for multi-objective flowshop scheduling problems. *Computers & Operations Research*, 38(11):1521–1533.

Montgomery, D. (2012). *Design and Analysis of Experiments*. John Wiley & Sons, 8 edition.

Rostami, M., Pilerood, A. E., and Mazdeh, M. M. (2015). Multi-objective parallel machine scheduling problem with job deterioration and learning effect under fuzzy environment. *Computers & Industrial Engineering*, 85:206–215.

Ruiz, R. and Andrés-Romano, C. (2011). Scheduling unrelated parallel machines with resource-assignable sequence-dependent setup times. *The International Journal of Advanced Manufacturing Technology*, 57:777–794.

Ruiz, R., Pan, Q.-K., and Naderi, B. (2019). Iterated greedy methods for the distributed permutation flowshop scheduling problem. *Omega*, 83:213–222.

Ruiz, R. and Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177(3):2033–2049.

Ruiz-Torres, A. J., López, F. J., and Ho, J. C. (2007). Scheduling uniform parallel machines subject to a secondary resource to minimize the number of tardy jobs. *European Journal of Operational Research*, 179(2):302–315.

Torabi, S., Sahebjamnia, N., Mansouri, S., and Bajestani, M. A. (2013). A particle swarm optimization for a fuzzy multi-objective unrelated parallel machines scheduling problem. *Applied Soft Computing*, 13(12):4750–4762.

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*, 211:612–622.

Vallada, E., Villa, F., and Fanjul-Peyro, L. (2019). Enriched metaheuristics for the resource constrained unrelated parallel machine scheduling problem. *Computers & 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.

Wang, S. and Liu, M. (2015). Multi-objective optimization of parallel machine scheduling integrated with multi-resources preventive maintenance planning. *Journal of Manufacturing Systems*, 37:182–192.

Yepes-Borrero, J. C., Villa, F., Perea, F., and Caballero-Villalobos, J. P. (2020). Grasp algorithm for the unrelated parallel machine scheduling problem with setup times and additional resources. *Expert Systems with Applications*, 141:112959.

Zhang, Z., Tang, Q., Ruiz, R., and Zhang, L. (2020). Ergonomic risk and cycle time minimization for the u-shaped worker assignment assembly line balancing problem: A multi-objective approach. *Computers & Operations Research*, 118:104905.

Zhou, S., Li, X., Du, N., Pang, Y., and Chen, H. (2018). A multi-objective differential evolution algorithm for parallel batch processing machine scheduling considering electricity consumption cost. *Computers & Operations Research*, 96:55–68.

Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C. M., and da Fonseca, V. G. (2003). Performance assessment of multiobjective optimizers: an analysis and review. *IEEE Transactions on Evolutionary Computation*, 7(2):117–132.