Annals of Operations Research
https://doi.org/10.1007/s10479-026-07065-5

ORIGINAL RESEARCH

Unrelated parallel machine scheduling problem with setup times and additional resources: an enhanced metaheuristic to address resource-related infeasibilities

Juan C. Yepes-Borrero[1] · **Javier Alcaraz**[2] · **Marta López-García**[1] · **Mario Villaizán-Vallelado**[1,3]

Received: 17 May 2025 / Accepted: 14 January 2026

Abstract
Efficient scheduling tools are essential for managing production environments where both machine availability and additional resource constraints play a significant role. This paper addresses the Unrelated Parallel Machine scheduling problem with setup times and additional resources in the Setups (UPMSR-S), an NP-hard problem that models real-world production settings where setups require limited resources, such as personnel or specialized equipment. We propose an enhanced algorithm designed to better handle resource-related infeasibilities and consistently outperform state-of-the-art methods. This is demonstrated through an extensive computational campaign on 1,000 benchmark instances, with improvements in Relative Percentage Deviation (RPD) exceeding 70% for several instance sizes. The proposed approach is well suited to large production environments involving setup and resource constraints, showing strong performance in challenging scheduling settings. Statistical analysis confirms that the method is highly effective across a wide range of instance sizes and scenarios, with particularly strong performance as the number of jobs and machines increases.

Keywords Scheduling · Unrelated Parallel Machine · Additional resources · Metaheuristics · GRASP

1 Introduction

In today's globalized economy, companies must optimize their processes to remain competitive (Mahmoodi et al., 2024). This requires careful management not only of time, but also of the resources used, which are often limited. In many manufacturing environments, these resources (such as operators or tools required for machine setup) are not infinite, and their scarcity can become a bottleneck in production (Luo et al., 2023). For instance, in ceramic tile manufacturing, machines process large batches automatically, but setup adjustments (e.g.,

✉ Juan C. Yepes-Borrero
juancamilo.yepes@uva.es

1 Departamento de Estadística e Investigación Operativa, Universidad de Valladolid, Paseo de Belén 7, 47011 Valladolid, Spain

2 Centro de Investigación Operativa, Universidad Miguel Hernández, Elche 03202, Spain

3 Telefonica Scientific Research, Madrid, Spain

Published online: 26 January 2026

mould or glaze changes) must be performed manually by a limited number of specialized operators. These operators constitute a renewable setup resource that can only assist one machine at a time, meaning that parallel setups are constrained by their availability. This situation often leads to waiting times between consecutive setups and exemplifies the practical relevance of modelling resource limitations in scheduling problems. When several machines require setup operations simultaneously but only one operator is available, the pending setups must wait in sequence, delaying production and increasing idle time. Efficient allocation and management of these finite resources is critical to maintaining smooth operations and reducing delays (Mahmoodi et al., 2024). Moreover, the increasing operational complexity of production systems, driven by Industry 4.0 advancements, product customization, and shorter delivery times, puts additional pressure on companies to find efficient solutions that optimize not only the use of machines but also of human and material resources (Florescu & Barabas, 2022; Luo et al., 2023).

Problems involving the sequencing of jobs on machines, known as scheduling problems, are central to production optimization (Avgerinos et al., 2023; Yepes-Borrero et al., 2021). These problems vary in the level of challenge they pose depending on the structure of the production environment. Among the different types of scheduling problems, one of the most common is the Parallel Machine scheduling problem (PM), where each job must be processed by exactly one machine from a set of machines that can operate simultaneously (Avgerinos et al., 2023; Elidrissi et al., 2023). The challenge lies in determining how to assign jobs to machines in a way that optimizes an objective, such as minimizing tardiness or the maximum completion time among all machines, also known as makespan (Avgerinos et al., 2023; Elidrissi et al., 2023; Lee & Jang, 2019).

An extension of this is the Unrelated Parallel Machine scheduling problem (UPM) (Lei et al., 2021; Mor et al., 2025; Yan et al., 2025), where jobs have different processing times on different machines, which significantly complicates the assignment decisions. This difficulty is further amplified when setup times are considered, a common complication in modern production environments. Machines often require reconfiguration or preparation between jobs, a process known as setup, which can vary in duration depending on the specific job being processed next. These setups must be efficiently managed to minimize downtime. This introduces the Unrelated Parallel Machine scheduling problem with sequence-dependent setup times (UPMS) (Fang et al., 2022), where the setup time between jobs depends on the sequence in which jobs are processed. However, beyond the challenge of scheduling setups, many production environments also face the constraint of limited additional resources, such as personnel or tools required to perform the setups. When these resources are scarce, the resulting problem is harder to solve in practice due to the increased coupling between scheduling and resource allocation decisions leading to the Unrelated Parallel Machine scheduling problem with setup times and additional Resources in the Setups (UPMSR-S) (Yepes-Borrero et al., 2020). The inclusion of these additional resource constraints not only enhances the realism of the model but also fundamentally alters its combinatorial structure and feasibility conditions, needing novel algorithmic approaches. Even in its simplest form, with just two identical machines, the PM is known to be $\mathcal{NP}$-hard (Lenstra et al., 1977). The problem becomes increasingly difficult to solve in practice as the instance size grows, making exact methods impractical for large-scale instances.

Recent studies have emphasized the need to incorporate practical features such as energy efficiency, workforce allocation, maintenance coordination, and resource constraints into modern scheduling models (Avgerinos et al., 2023; Heinz et al., 2022; Mahmoodi et al., 2024). In parallel, recent reviews highlight that the diversification of parallel machine scheduling problems is still ongoing, with some areas, particularly sequence-dependent setups and

resource integration, still not fully explored (Geurtsen et al., 2023; Ying et al., 2024). According to Ying et al. (2024), sequence-dependent setups in parallel machine scheduling are at a growth stage, while Geurtsen et al. (2023) note that the number of studies simultaneously considering maintenance activities and additional resources is still limited. Despite this progress, most existing works continue to treat setups and resource availability as separate issues, and few models explicitly capture their combined effect. This gap motivates the study of problems such as the UPMSR-S, which address both sequence-dependent setups and renewable resource limitations in an integrated manner, thereby contributing to closing the gap between theoretical scheduling research and industrial practice. To illustrate this, real-world scenarios of the UPMSR-S can be found in several modern manufacturing environments. One representative example arises in the food-processing and packaging industry, particularly in multiproduct filling and sealing lines (Stefansdottir et al., 2017). These lines behave as unrelated parallel machines, since the processing time of each product varies substantially across machines due to differences in viscosity, temperature, or handling requirements. Between two consecutive batches, sequence-dependent setup operations, such as cleaning, sterilization, or format adjustments, must be performed. For instance, switching from an allergen-containing product to an allergen-free product requires a longer sanitation procedure than the reverse transition. These setups demand the intervention of certified technicians or specialized cleaning equipment, which constitute additional renewable resources available in limited quantity. When several lines require reconfiguration at the same time, the setups must wait for these resources to become available, leading to idle times and increased scheduling difficulty. This scenario fits naturally within the UPMSR-S framework, as both sequence-dependent setups and resource constraints must be jointly coordinated. Another relevant example appears in the machining of aerospace components using high-precision CNC (computer numerical control) equipment (Chen et al., 2024). Each part type can, in principle, be processed on multiple machines, but the processing times differ significantly across machines due to variations in stiffness, spindle capabilities, or fixture requirements. As a result, the environment is well modeled as one of unrelated parallel machines. Transitions between different families of parts involve sequence-dependent setup tasks, such as recalibrating tool heads, realigning fixtures, or installing dedicated tooling systems. These operations generally require the presence of highly skilled technicians or the use of scarce calibration devices, which act as additional constrained resources. When these resources are not immediately available, setups must be postponed, thereby altering machine schedules and creating additional bottlenecks. This type of manufacturing system exemplifies the UPMSR-S problem, where both the heterogeneity of machines and the limited availability of setup-related resources play a critical role in determining feasible and efficient schedules.

Given the NP-hard nature of the UPMSR-S problem, solving it using exact methods becomes impractical for large instances. Previous studies have demonstrated that exact approaches struggle with the computational burden as the number of jobs and machines increases. As a result, heuristic and metaheuristic methods have gained significant attention in recent years as viable alternatives for obtaining high-quality solutions in a reasonable amount of time. In particular, the addition of resource constraints transforms the search space, requiring customized metaheuristic components to effectively explore this more constrained solution space.

This paper focuses on the UPMSR-S problem and, given that it is computationally hard to solve using exact methods, we propose a novel hybrid metaheuristic algorithm to tackle it. Our approach aims to efficiently solve large and realistic instances of the problem, where resource limitations and machine setups pose significant challenges to production planning. In doing so, we aim to contribute to the ongoing development of advanced algorithms for

scheduling problems that incorporate both machine and resource constraints, thus improving their applicability in industrial environments.

The main **contributions** of our paper are summarized as follows:

- Development of a new algorithm for managing resource-related infeasibilities in scheduling problems with limited resources.
- Extensive computational testing, showing consistent and significant performance improvements over previous methods across a wide range of instance sizes.

The rest of the paper is organized as follows. Section 2 presents an overview of the related literature. Section 3 presents the formal definition of the problem and a mathematical model. Section 4 introduces the metaheuristic designed for solving the UPMSR-S. Section 5 shows the experimental campaign to assess the algorithms proposed. Finally, in Section 6 some conclusions and directions for future research are given.

2 Related work

Machine scheduling problems have been the focus of extensive research due to their relevance in industrial environments. As explored in Yazdani and Haghani (2024), the extensive research conducted over the last 20 years has driven the development of more sophisticated algorithms and a deeper understanding of the characteristics of scheduling problems.

Among these, the parallel machine scheduling problem has been widely studied due to its direct applicability to real production systems where several machines operate simultaneously. Recent work, such as Durasević and Jakobović (2023), presents a detailed review of heuristic and metaheuristic algorithms specifically designed for the unrelated parallel machine scheduling problem. This analysis highlights the computational difficulty of the problem and the different approaches taken to address it. Additionally, Ying et al. (2024) provides a comprehensive analysis of the different contributions to the parallel machine problem in recent years, highlighting the increasing attention that variations of the problem, which take setup times into account, are receiving. This section reviews parallel machine scheduling problems and their main extensions related to setup times, additional resources, and their integration.

2.1 Parallel machine scheduling with setup times

Setup times have been incorporated into the UPMSP to better represent industrial scenarios where machines require reconfiguration between jobs. For example, Saraç and Tutumlu (2022) propose a bi-objective mathematical model for UPMSP with setup times, while Li et al. (2024) develop exact formulations that account for family setups and machine costs. Metaheuristic approaches also play a key role: Yilmaz Eroglu et al. (2014) introduce a genetic algorithm with local search for UPMSP with sequence-dependent setups, whereas Báez et al. (2019) combine Greedy Randomized Adaptive Search Procedure (GRASP) and Variable Neighbourhood Search to tackle PM problems with dependent setup times, demonstrating that hybridization can outperform both exact and single-heuristic approaches. Industrial extensions have also emerged, such as the textile application in Berthier et al. (2022), who integrate machine eligibility and dual resource types into the UPMSP, and the hybrid large neighbourhood and tabu search metaheuristic proposed by Fang et al. (2022). Other recent studies include multi-objective formulations for parallel machine scheduling (Lei et al., 2021;

Srinath et al., 2023), as well as exact and matheuristic approaches that address cost, time-window, or due dates (Chen et al., 2024; Fonseca et al., 2024; Mor et al., 2025). However, these works primarily focus on setup-sequence dependencies and rarely incorporate explicit resource constraints affecting setup feasibility.

2.2 Parallel machine scheduling with additional resources

In parallel machine scheduling, the introduction of additional resources, such as operators, tools, or auxiliary equipment, adds another layer of difficulty. Some studies have addressed problems with resources required for processing but not for setups. For instance, Vallada et al. (2019) and Villa et al. (2018) propose heuristics for parallel machines with additional processing resources, while Li et al. (2019) introduce a fuzzy swarm optimization algorithm for uniform machines under resource consumption constraints. Munoz et al. (2022) and Mor and Berlińska (2025) focus on dual resource-constrained environments, whereas Shafiee et al. (2025) consider renewable resources that alternate among unrelated machines. Although the focus of this review is on parallel machine settings, other scheduling problems involving resource constraints have also been explored, for example, project and flexible job-shop scheduling (Klein, 2025; Klein et al., 2024; Perrachon et al., 2025) or flow-shop configurations with batch delivery constraints (Zeng et al., 2025).

2.3 Integrating setup and resource constraints in parallel machine problems

The explicit integration of resource constraints within UPMSP models remains limited. Fanjul-Peyro (2020) classify resources into three categories: processing-specific resources (UPMR) (Vallada et al., 2019), setup-specific resources (UPMSR-S) (Yepes-Borrero et al., 2020), and dual-resource problems involving both processing and setups (UPMSR-P+S) (Lopez-Esteve et al., 2023). While these formulations mark important progress, few algorithms directly manage setup-resource feasibility. Some related studies also consider shared entities during setups. Heinz et al. (2022) model common servers as shared resources in identical-machine environments, while Avgerinos et al. (2023) introduce setup operators in job-splitting settings. However, these assumptions differ from the UPMSR-S framework, which deals with unrelated machines and does not allow job splitting.

2.4 Positioning of the present study

Table 1 summarizes the main features of the studies that are, in our view, the most closely related to the problem addressed in this paper. All of them consider sequence dependent setups, but they differ in how resource constraints are modelled. Among these studies, only Yepes-Borrero et al. (2020) address the same UPMSR-S problem, explicitly considering renewable resources required during setups. The other works deal with different variants, either without resource constraints, with identical machines, or with job-splitting assumptions, which makes direct comparison difficult. Although parallel machine scheduling has been extensively researched, variants that jointly include setup dependencies and renewable resource constraints remain comparatively underexplored, despite their closer alignment with real industrial settings. Advancing algorithms for these problems is therefore essential to improve operational efficiency and competitiveness.

Table 1 Comparison of the most relevant studies on parallel machine scheduling with setups

Reference	Problem type	Resources	Method	Objective
Vallada and Ruiz (2011)	UPMS	None	GA	$C_{\max}$
Báez et al. (2019)	UPMS	None	GRASP+VNS	$C_{\max}$
Avgerinos et al. (2023)	UPMS (job-splitting)	Setup	CP+Heuristic	$C_{\max}$
Lopez-Esteve et al. (2023)	UPMSR-P+S	Proc. + Setup	GRASP	$C_{\max}$
Heinz et al. (2022)	PMSR-S	Setup	CP	$C_{\max}$
Yepes-Borrero et al. (2020)	UPMSR-S	Setup	GRASP	$C_{\max}$
This work	**UPMSR-S**	**Setup**	**E-GRASP**	$C_{\max}$

In this work, we propose an enhanced metaheuristic that more effectively handles infeasibilities caused by limited setup resources. The approach is general and can be incorporated into other heuristics facing similar renewable-resource constraints. It includes an improved construction phase and a selective repair strategy, resulting in a more efficient method for solving the UPMSR-S.

3 Problem description

In this section, we present a formal definition of the UPMSR-S problem.

In this problem, we are given a set N of n jobs (indexed by j and k) that must be processed on a set M of m machines (indexed by i). Each job must be processed exactly once on any of the available machines. Each machine can process only one job at a time, and no pre-emption is allowed. The machines are unrelated, meaning that the processing time for a job can vary across different machines. In addition, to process two jobs consecutively on the same machine, a setup must be performed. Setup times are sequence-dependent, as the time required to prepare the machine for a given job varies according to the job that was immediately previously processed. That is, the setup time can vary depending on the particular pair of jobs j and k being scheduled. Furthermore, these setup operations require additional resources, such as workers or tools, and these resources are limited.

The processing times are denoted by p_{ij}, representing the time it takes to process job j on machine i. The setup time on machine i, s_{ijk} , denotes the time required to prepare the machine for processing job k given that job j was processed immediately beforehand. The resources required for setups are denoted by r_{ijk}, representing the resource consumption for the setup between jobs j and k on machine i. The total available resources are limited by a maximum value ($R_{\max}$), which cannot be exceeded at any given time. The objective is to minimize the makespan, which is the maximum completion time among all machines.

Given the resource constraints, there may be infeasible solutions that require more resources to execute. This adds scheduling difficulty to the problem, as it is not only necessary to determine on which machine and in what order to process the jobs, but also when to start the setups to satisfy the resource consumption constraints. Figure 1 illustrates two different solutions for a problem with 4 machines and 8 jobs. The blue boxes represent each job, while the yellow boxes represent the setups that must be performed on the machines to process two consecutive jobs. Within the setup boxes, the number of resources r_{ijk} required for each setup is displayed. In Figure 1a with a maximum resource availability $R_{\max}$ of 3, the first solution is infeasible, as it requires four resources between time instants t_3 and t_4 to execute

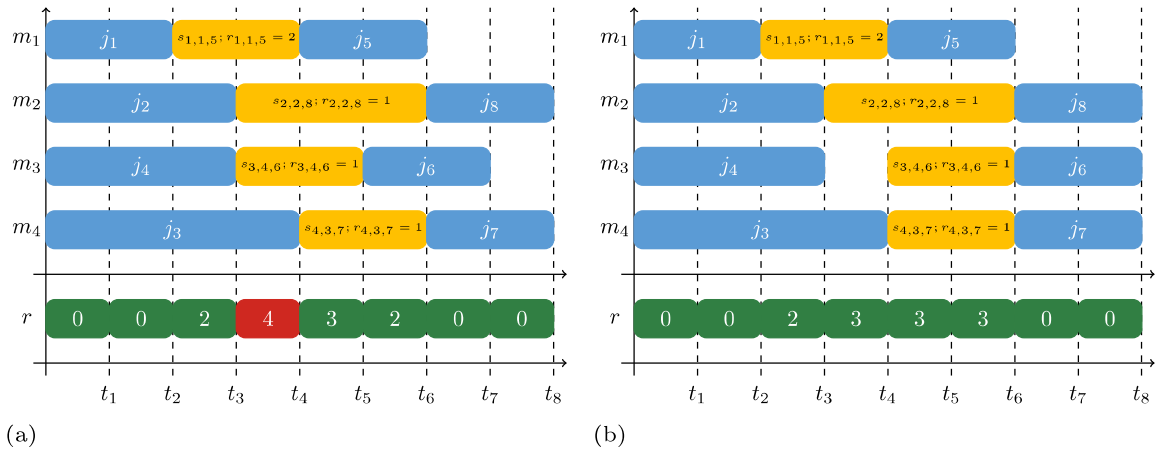


Fig. 1 Example of a UPMSR-S problem with $R_{\max} = 3$. In Figure 1a, the solution is infeasible because the resource constraints are exceeded between time t_3 and t_4. In Figure 1b, the solution is feasible, as the setups are adjusted to respect the resource constraints

the setups simultaneously. In Figure 1b the start of the setup on machine 2 is postponed, thereby satisfying the resource constraint and resulting in a feasible solution.

The detailed mixed integer linear programming model of the UPMSR-S is given by Yepes-Borrero et al. (2020). Before detailing the model, the following variables are defined:

- $Y_{ij} \in \{0, 1\}$: Binary variable that takes the value 1 if job j is processed on machine i, and 0 otherwise.
- $X_{ijk} \in \{0, 1\}$: Binary variable that takes the value 1 if job k follows job j on machine i, and 0 otherwise.
- $H_{ijkt} \in \{0, 1\}$: Binary variable that takes the value 1 if the setup between jobs j and k on machine i ends at time t, and 0 otherwise.
- $C_{\max} \geq 0$: Non-negative variable representing the maximum completion time (makespan) of the schedule.

Moreover, the set $N_0 = N \cup \{0\}$ is defined, where 0 represents a dummy job. Each machine starts and finishes processing at job 0, which has zero processing and setup times as well as zero resource consumption. The parameters are set as $p_{i0} = s_{i0k} = s_{ik0} = r_{i0k} = r_{ik0} = 0$, $\forall i \in M; \forall k \in N_0$.

The mathematical formulation for the UPMSR-S problem is as follows:

$$\min C_{\max}, \tag{1}$$

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

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

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

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

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

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

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

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

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

$$Y_{ij} \in \{0, 1\}, \forall i \in M,\, j \in N$$

$$X_{ijk} \in \{0, 1\}, \forall i \in M,\, j \in N, k \in N$$

$$H_{ijkt} \in \{0, 1\}, \forall i \in M,\, j \in N, k \in N, t \leq t_{\max}.$$

$$C_{\max} \geq 0.$$

Objective (1) aims to minimize the makespan. Constraints (2) ensure that each machine i is assigned only one job at the first position of its sequence. Constraints (3) guarantee that every job j is assigned to exactly one machine. Constraints (4) ensure that every job j processed on machine i has exactly one successor k. Similarly, constraints (5) ensure that each job k on machine i has exactly one predecessor j. Constraints (6) specify that for each machine i and successive jobs j and k, the setup time between these jobs must end at a specific point before $t_{\max}$. Constraints (7) enforce that the setup time between jobs j and k on machine i is completed as early as possible. Constraints (8) limit the number of resources used at any moment to be less than or equal to $R_{\max}$. Finally, constraints (9) ensure that $C_{\max}$ is greater than or equal to the time at which all setups, including the final dummy setup, are completed.

With this model, only very small instances can be solved (see Yepes-Borrero et al. (2020)); therefore, to address larger and more realistic instances, it is necessary to propose heuristic algorithms.

4 Proposed metaheuristic

In this section, we present an enhanced metaheuristic algorithm that improves the methods presented in Yepes-Borrero et al. (2020). The motivation for this improvement comes from the limitations observed in the original approach, mainly related to the repair process required to address infeasibilities caused by violations of the setup-resource constraint, which was frequently triggered and computationally expensive.

The improvements introduced in the proposed algorithm focus on making the solution repair process significantly more efficient. First, the algorithm is designed to generate solutions that require fewer adjustments to become feasible. Second, it reduces the number of solutions that are actually repaired. As a result, the algorithm can perform many more iterations within the Greedy Randomized Adaptive Search Procedure (GRASP) framework (Feo & Resende, 1989), leading to a more comprehensive exploration of the solution space.

Figure 2 shows the flowchart of the proposed algorithm. The method begins with an enhanced construction phase, aimed at generating initial solutions with low resource consumption and short makespan. These solutions are then refined through an enhanced local search phase, focused on further improving solution quality while maintaining low resource consumption. Both phases are designed to produce solutions that require few adjustments to become feasible. Instead of repairing every solution, we evaluate each one to determine whether it is promising in terms of makespan and resource consumption. Only those solu-

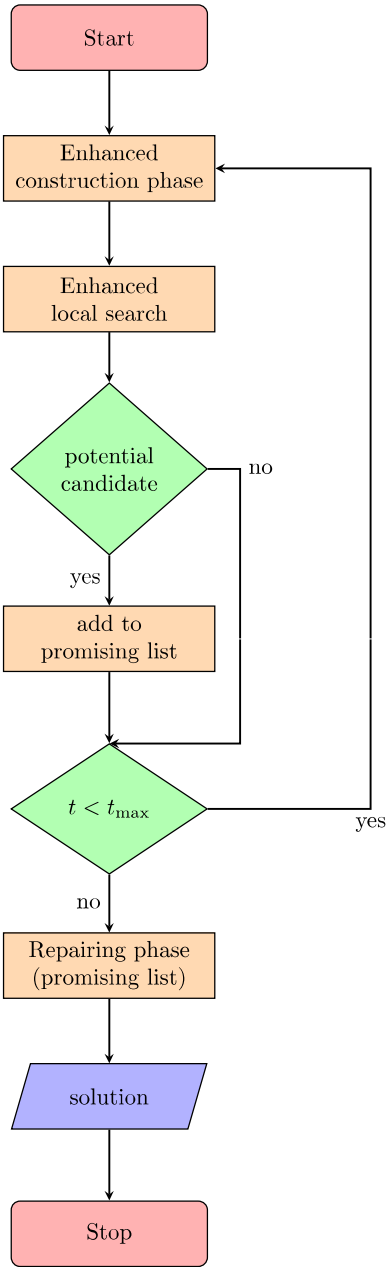


Fig. 2 Flowchart of the E-GRASP to solve the UPMSR-S problem

tions considered promising are stored in a list. At the end of the algorithm execution, only the solutions from this list proceed to the repair phase, significantly reducing the number of repairs.

Although the new algorithm proposed in this work builds on the approach of Yepes-Borrero et al. (2020), there are several key differences that lead to a substantially different behavior and efficiency. The most relevant changes are: (i) the introduction of the β parameter in both the construction and local search phases, (ii) a simplified neighbourhood evaluation to reduce computational overhead, and (iii) a selective repair mechanism driven by the promising list. These modifications aim to generate better initial solutions, reduce the number of required repairs, and allow more iterations within the same runtime.

Before describing the phases of the algorithm, it is important to clarify how the solutions are encoded. Each machine is represented as a list, where the jobs assigned to it are placed sequentially. If a job appears in a machine's list, it indicates that the job is processed on that machine, with its position in the list representing the processing order. For each job, we track

the process start time, setup start time, and final end time. This encoding allows efficient scheduling, ensuring that all necessary times and constraints are respected throughout the algorithm's different phases, such as the construction phase and local search. Hereafter, we will provide a detailed explanation of the proposed algorithm, emphasizing the differences compared to the original algorithm.

4.1 Enhanced Construction phase

In the construction phase, the objective is to generate initial solutions by assigning and sequencing all jobs to the machines. As in Yepes-Borrero et al. (2020), setups are scheduled following an *earliest-start* policy, in which setups begin as soon as the preceding operations allow, which provides a computationally efficient way to construct the initial sequences without explicitly checking global resource feasibility at each insertion. Consequently, the solutions generated during this phase are not necessarily feasible, as they may require more resources than those available, so these solutions are evaluated and, if necessary, repaired in the repair phase that we will explain in Section 4.3.

Since the solutions generated in this phase are not always feasible, it is crucial to ensure that they are promising before moving on to the repair phase. A promising solution is one that has a low makespan and low resource consumption. To evaluate whether a solution is promising, Yepes-Borrero et al. (2020) propose a λ value. This value combines the information from both setup times and resource consumption into a single metric. The idea is to guide the insertion process, favouring jobs that not only minimize the increase in makespan but also maintain low resource consumption, so that a lower λ value implicitly indicates shorter setups and reduced resource usage.

In this work, we explore the impact of adjusting the weight given to setups and resource consumption in the λ calculation. Specifically, we introduce a new parameter, β, which allows us to control the weight we give to resource consumption and setups in the insertion process. For this purpose, we propose a modified λ^{β} value, defined as:

$$\begin{aligned}\lambda^{\beta}_{i,j,k} &= C'_i + p_{ij} \\ &+ \beta\left[\left(\theta_{s(i,k-1,k)} \cdot \theta_{r(i,k-1,k)}\right) + \left(\theta_{s(i,k,k+1)} \cdot \theta_{r(i,k,k+1)}\right) - \left(\gamma_{s(i,k)} \cdot \gamma_{r(i,k)}\right)\right],\end{aligned} \tag{10}$$

where C'_i represents the current completion time of machine i in the partial solution after job j is inserted, while p_{ij} is the processing time of job j on machine i. β is the new scaling factor introduced to adjust the weight of the setup times and resource consumption. The terms $\theta_{s(i,k-1,k)}$ and $\theta_{s(i,k,k+1)}$ denote the setup times required when inserting job j in position k on machine i, and $\theta_{r(i,k-1,k)}$ and $\theta_{r(i,k,k+1)}$ correspond to the resources needed for these setups. Finally, $\gamma_{s(i,k)}$ and $\gamma_{r(i,k)}$ represent the setup time and resources that are no longer necessary due to the insertion of job j in position k on machine i.

The new β parameter allows us to adjust how much we penalize insertions with longer setups and higher resource consumption. Higher β values will penalize insertions with higher resource consumption and longer setup times. Several values of β were tested to calibrate its influence on the algorithm's performance and the results of this calibration are shown in Section 5.1.

The construction phase starts with the list of pending jobs, which initially consist of the entire set of N jobs. For each pending job, we evaluate every possible insertion position within the current partial solution, considering all machines. For each possible insertion, the corresponding $\lambda^{\beta}_{i,j,k}$ value is computed. The best candidate for insertion is the job j at

Algorithm 1: Enhanced construction phase.

$PendingJobs \leftarrow N$;
foreach $i \in M$ **do**
  $PartialSolution_i \leftarrow \emptyset$;
end
while $PendingJobs \neq \emptyset$ **do**
  foreach $j \in PendingJobs$ **do**
    foreach $i \in M$ **do**
      for $k = 0$ **to** $|PartialSolution_i|$ **do**
        Calculate $\lambda^{\beta}_{i,j,k}$ with Equation (10);
      end
    end
  end
  $RCL(\alpha) \leftarrow \left\{(i,j,k) : \lambda^{\beta}_{i,j,k} \leq \min\left(\lambda^{\beta}_{i,j,k}\right) + \alpha\left(\max\left(\lambda^{\beta}_{i,j,k}\right) - \min\left(\lambda^{\beta}_{i,j,k}\right)\right)\right\}$;
  Randomly choose $(i^*, j^*, k^*) \in RCL(\alpha)$;
  Update $PartialSolution_{i^*}$: Assign job j^* on machine i^* in position k^*;
  Remove j^* from $PendingJobs$;
end

position k on machine i that yields the lowest $\lambda^{\beta}_{i,j,k}$ value. However, as this is a GRASP algorithm, instead of selecting the best candidate directly, a candidate is chosen at random from a restricted candidate list (RCL). The RCL consists of the top candidates that meet a threshold determined by $UpperBound = \min\left(\lambda^{\beta}_{i,j,k}\right) + \alpha\left(\max\left(\lambda^{\beta}_{i,j,k}\right) - \min\left(\lambda^{\beta}_{i,j,k}\right)\right)$. The parameter $\alpha \in [0, 1]$ controls the size of the restricted candidate list. Lower values lead to a more greedy behavior (smaller RCL), while higher values introduce greater randomness (bigger RCL). One candidate is then drawn uniformly at random from the RCL, after which the corresponding job is assigned and the job is removed from the list of pending jobs.

4.2 Enhanced Local search

The local search phase aims to enhance the current solution by performing job swaps across all machines. Unlike the approach proposed in Yepes-Borrero et al. (2020), it was simplified to improve efficiency. While the original version explored both internal and external job insertions, the enhanced algorithm only performs swap moves between pairs of jobs. This modification significantly reduces the number of neighbourhood evaluations and therefore the computational time per iteration. Additionally, we incorporate the new β factor (explained in Section 4.1) to weigh the importance of setup times and resource consumption in each potential swap. For each job, a potential swap is considered with every other job on all machines. For each possible swap between job j on machine i and job k on machine i', a value, denoted as $LS(j_i, k_{i'})$, is computed. This value $LS\left(j_i, k_{i'}\right)$ is defined as:

$$LS\left(j_i, k_{i'}\right) = \Delta p\left(j_i, k_{i'}\right) + \beta \cdot \left(\Delta s\left(j_i, k_{i'}\right) \cdot \Delta r\left(j_i, k_{i'}\right)\right), \tag{11}$$

where:

- $\Delta p\left(j_i, k_{i'}\right)$ indicates the difference in processing times when swapping job j from machine i with job k from machine i'. This difference is calculated as the sum of the new processing times $p_{i'j}$ and p_{ik} after the swap, minus the sum of the old processing times p_{ij} and $p_{i'k}$ before the swap. It is important to note that if $i = i'$, then $\Delta p\left(j_i, k_{i'}\right) = 0$.
- β is the same factor that was explained in the construction phase.

- $\Delta s\left(j_i, k_{i'}\right)$ indicates the difference in setup times when swapping job j from machine i with job k from machine $i^{'}$. This difference represents the new setup times that must be performed due to the swap, minus the setup times that are no longer necessary.
- $\Delta r\left(j_i, k_{i'}\right)$ indicates the difference in resource consumption when swapping job j from machine i with job k from machine $i^{'}$. This difference represents the resources needed for the new setups that must be performed due to the swap, minus the resources consumed by the old setups that are no longer necessary.

After evaluating all possible swaps, the swap that yields the most negative $LS\left(j_i, k_{i'}\right)$ value (indicating the greatest improvement) is executed. The process is iteratively repeated until no further swaps yield negative $LS\left(j_i, k_{i'}\right)$ values, at which point the local search terminates. Algorithm 2 illustrates the general procedure of the local search.

Algorithm 2: Enhanced local search.

```
Improvement ← TRUE;
while Improvement ≠ FALSE do
    Improvement ← FALSE;
    BestImprovement ← ∞;
    foreach j ∈ N do
        foreach k ∈ N \ {j} do
            Calculate LS(j_i, k_i');
            if LS(j_i, k_i') < BestImprovement then
                Improvement ← TRUE;
                BestImprovement ← LS(j_i, k_i');
                BestSwap ← (j_i, k_i');
            end
        end
    end
    Do BestSwap;
end
```

4.3 Repairing phase

Since the generated solutions do not necessarily satisfy the resource constraint, it is necessary to evaluate the resource consumption at each time instant. If the constraint is not satisfied at some point, the solution must be repaired. For this purpose, we employ the same mechanism proposed by Yepes-Borrero et al. (2020).

The algorithm evaluates the resource consumption at each time instant. When the resource constraint is not satisfied, the algorithm delays by one time unit the start of the setup that started later among all the setups that are running at that time instant. Then, the resource consumption is evaluated again at the same time instant. If the constraint is satisfied, the process advances to the next time instant. This procedure is repeated until the resource constraint is satisfied throughout the solution. Algorithm 3 illustrates the general procedure of the repairing phase.

Algorithm 3: Repairing phase.

```
t ← 0;
while t < C_max do
    if Resources consumption > R_max then
        Postpone the start of the setup that began the latest at time instant t;
        Update C_max;
    else
        t ← t + 1;
    end
end
```

4.4 Enhanced algorithm

The enhanced metaheuristic algorithm, as illustrated in Algorithm 4, iteratively performs the construction and local search phases, running for a predefined runtime (RT). A key innovation in this new approach is the selective repair mechanism, where only the solutions in the promising list at the end of the runtime are repaired, in contrast to the method in Yepes-Borrero et al. (2020), which repairs all solutions immediately.

As seen at the start of Algorithm 4, the algorithm begins by initializing the $PromisingList$ to store the most promising solutions and setting the $BestSolution$ to an infinite value. During the construction phase, in each iteration, a new solution ($NewSolution$) is generated. While this step is similar to the original algorithm, it is important to note that the assignment rule has been modified due to the introduction of the new β parameter in this work. Following the construction phase, the local search seeks to improve the solution by performing job swaps across machines. This phase differs from the original algorithm by being faster, as it performs fewer swaps per iteration.

Once the local search is completed, the solution is evaluated to determine if it is promising. The most promising ρ solutions (those with the lowest makespan) are stored in the $PromisingList$ until the algorithm reaches the total RT. It is important to clarify that solutions with the lowest makespan are considered promising because both the construction and local-search phases follow the same evaluation principle introduced through the λ metric, which penalizes long setups and high resource consumption. As a result, the candidate solutions reaching this stage already exhibit low resource usage, and selecting those with the smallest makespan naturally favours both short completion times and low resource consumption.

As shown in Algorithm 4 the $PromisingList$ is continuously updated throughout the runtime. Only at the end of RT, the solutions in the $PromisingList$ move on to the repairing phase. This step differs significantly from the previous algorithm, where every solution was repaired immediately, limiting the number of iterations due to high computational costs during the repair phase. Finally, after repairing the solutions in the list, the best solution is chosen by comparing the outcomes of all repaired solutions.

This approach ensures that more solutions are explored before the costly repair phase, allowing for a more comprehensive search of the solution space. Furthermore, the introduction of the new β parameter enhances this process by allowing for adjusted penalties on sequences with high resource consumption, ultimately contributing to potentially better results. By focusing only on promising solutions and adjusting the trade-off between makespan and resource consumption, this new method is expected to yield higher-quality solutions compared to previous algorithms. Several values of ρ were tested to calibrate its

influence on the algorithm's performance and the results of this calibration are shown in Section 5.1.

Algorithm 4: Enhanced metaheuristic.

```
PromisingList ← ∅ ;
BestSolution ← ∞ ;
while (Execution Time) < RT do
    Construct a new solution, NewSolution, using the construction phase;
    Improve NewSolution using the local search phase;
    if NewSolution is promising then
        Add NewSolution to PromisingList;
        if PromisingList exceeds size ρ then
            Remove the least promising solution from PromisingList;
        end
    end
end
foreach solution in PromisingList do
    Repair the solution using the repairing phase;
    if the repaired solution is better than BestSolution then
        Update BestSolution with this repaired solution ;
    end
end
return BestSolution ;
```

5 Experimental analysis

To evaluate the performance of the metaheuristic proposed in this paper, hereafter referred to as Enhanced GRASP (E-GRASP), we conducted experiments on the same large-scale instances introduced by Yepes-Borrero et al. (2020). These instances already model additional setup resources and generate processing and setup times in accordance with established benchmarks (e.g., Taillard (1993), Vallada and Ruiz (2011), Báez et al. (2019), Berthier et al. (2022), Fanjul-Peyró et al. (2019)).

For comparison, we include three reference algorithms. The GRASP of Yepes-Borrero et al. (2020) (denoted as Baseline GRASP or B-GRASP) is used as the main benchmark, since it was designed for the same UPMSR-S problem addressed in this work. In addition, two algorithms from the UPMS family were adapted because, in our judgment, they represent the most similar approaches in the literature, as they consider unrelated parallel machines with sequence-dependent setups.

The hybrid GRASP+VNS of Báez et al. (2019) (Báez-Hybrid), originally designed for the UPMS without resource constraints, was adapted by adding a repair phase to each generated solution to enforce feasibility with respect to setup resources. An additional variant incorporating the E-GRASP repair mechanism was tested, but this version did not yield better results, since the algorithm was not capable of generating sufficiently promising solutions for the repair procedure to be effective.

The algorithm of Lopez-Esteve et al. (2023) (Lopez-GRASP), developed for the UPMS with both processing and setup resources (UPMSR-P+S), was modified by removing all components related to processing-resource evaluation and repair, focusing only on setup-

resource feasibility. As in the previous case, this adaptation did not lead to improvements, as the algorithm was unable to generate promising solutions for the repair phase.

It is important to highlight that the enhancements introduced in E-GRASP are not limited to the repair mechanism. The inclusion of the new β factor, which penalizes sequences with high setup-resource requirements, allows the algorithm to guide the search toward promising regions of the solution space. This combination of selective repair and guided penalization is expected to enable E-GRASP to obtain higher-quality solutions than the reference methods.

The test instances are defined by varying the number of jobs $n \in \{50, 100, 150, 200, 250\}$ and the number of machines $m \in \{10, 15, 20, 25, 30\}$. Setup times were drawn from uniform distributions with four different ranges: $1-9$, $1-49$, $1-99$, and $1-124$. Processing times were uniformly drawn between 1 and 99. For each instance, the maximum resource availability ($R_{\max}$) was randomly set between 3 and 4. The resource consumption of each setup was also generated using a uniform distribution between 1 and $R_{\max}$.

By combining all possible values for the number of jobs (5), machines (5), and setup times distributions (4), a total of 100 different configurations was obtained. For each configuration, 10 random replicates were generated, resulting in a total of 1000 instances. Additionally, an extra replicate was generated for each configuration to create a separate set of instances, which was used to calibrate the algorithm parameters. This means that a total of 1000 instances were generated to evaluate the algorithms and 100 additional instances for calibration.

To compare the quality of the solutions, we use the Relative Percentage Deviation (RPD), which is calculated as follows:

$$RPD = \frac{Solution\_Value - Best\_Known\_Value}{Best\_Known\_Value} \cdot 100, \tag{12}$$

where $Best\_Known\_Value$ is the best makespan obtained among all methods tested for each instance.

All the experiments were carried out on virtual machines, each equipped with 2 virtual processors and 16 GB of RAM, running Windows 10 Enterprise 64-bit. All algorithms were implemented in C# using Microsoft Visual Studio 2022.

Before conducting the final comparison, we calibrated E-GRASP to identify its optimal settings, tuning the parameter α alongside the newly introduced β and ρ factors. The other algorithms (B-GRASP (Yepes-Borrero et al., 2020), Báez-Hybrid (Báez et al., 2019), and Lopez-GRASP (Lopez-Esteve et al., 2023)) were configured using the parameter values recommended in their original publications.

5.1 Algorithm calibration

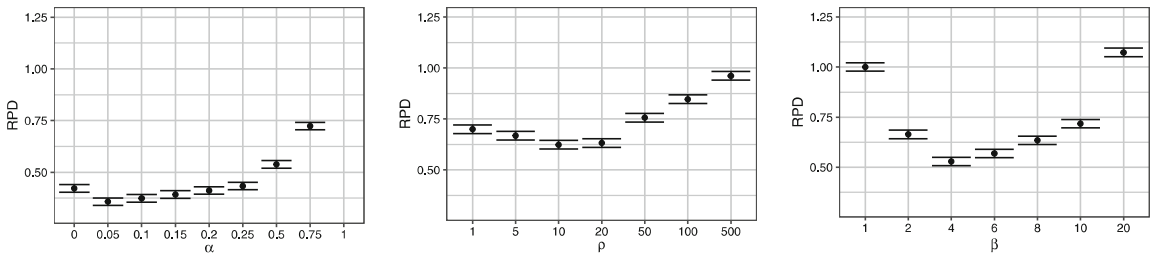
For the calibration of the algorithm parameters, we tested different values for α, the size of the promising solution list (ρ), and the scaling factor β. Specifically, for α, we evaluated 9 different values: {0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.5, 0.75, 1}. For ρ, we tested 7 values: {1, 5, 10, 20, 50, 100, 500}. For β, we also evaluated 7 levels: {1, 2, 4, 6, 8, 10, 20}.

We conducted a full factorial design of experiments, testing all possible configurations of the algorithm, resulting in $9 \cdot 7 \cdot 7 = 441$ different parameter combinations. Each configuration was run on the 100 instances from the calibration set, leading to a total of 44100 executions. All configurations have the same stopping criterion, which is based on the number of jobs in the instance: $t = n$ seconds, resulting in more than 62 days of CPU time to complete the calibration.

To compare the performance of all configurations, we performed an analysis of variance (ANOVA) (Montgomery, 2019) to determine whether there were significant differences

Table 2 ANOVA p-values for calibration factors (response variable: RPD)

Factor	p-value
α	< 0.001
ρ	< 0.001
β	< 0.001



(a) (b) (c)

Fig. 3 RPD Means plots with Fisher's LSD 95% confidence intervals for α (Figure 3a), ρ (Figure 3b) and β (Figure 3c) factors in metaheuristic calibration. Noting that in Figure 3a the average RPD for $\alpha = 1$ is greater than 3

between them. The response variable for the ANOVA was the RPD. Table 2 shows the p-values obtained from the ANOVA, indicating that all three factors are statistically significant and that variations in their values have a meaningful impact on the algorithm performance.

Figure 3 shows the mean RPD plots with Fisher's least significant difference (LSD) 95% confidence intervals for the three calibrated parameters. Non-overlapping intervals indicate significant differences between groups. In Figure 3a, the intervals for the α parameter show that the best level is $\alpha = 0.05$. Higher values of α correspond to larger RCL sizes, which introduce more randomness in the construction phase. The results suggest that controlling the RCL size is important, as smaller lists lead to better solutions. However, the results also indicate that setting $\alpha = 0$ (completely greedy construction phase) should be avoided, as it yields inferior results.

Figure 3b shows the results for ρ, with the best level being $\rho = 10$. This suggests that maintaining a moderately sized list of promising solutions is more efficient. Larger values of ρ can lead to higher computational costs, as managing and updating large lists becomes computationally expensive.

Finally, Figure 3c shows the intervals for the β parameter, where the best level is $\beta = 4$. The value $\beta = 1$ indicates that the parameter has no effect. Increasing β introduces more penalties for long setups and high resource consumption, which is beneficial to a point. However, excessively high values, such as $\beta = 20$, lead to worse solutions, likely due to over-penalizing setups and resource constraints.

5.2 Comparative Analysis of the Algorithms

To evaluate the performance of E-GRASP, we compare it with B-GRASP (Yepes-Borrero et al., 2020), Báez-Hybrid (Báez et al., 2019), and Lopez-GRASP (Lopez-Esteve et al., 2023) on the full set of 1000 instances. The stopping criterion for E-GRASP was $t = n$ seconds, plus the additional time required to repair the $\rho = 10$ solutions in the promising list. Since this repair time cannot be known in advance and varies across instances, it was measured after execution and averaged for each instance size. To ensure a fair comparison,

Table 3 Comparison of average RPD (%) between the E-GRASP, the B-GRASP (Yepes-Borrero et al., 2020), Báez-Hybrid (Báez et al., 2019), and the Lopez-GRASP (Lopez-Esteve et al., 2023) algorithms across different instance sizes. Best results in bold.

Jobs	Machines	E-GRASP RPD (%)	B-GRASP Yepes-Borrero et al. (2020) RPD (%)	Báez-Hybrid Báez et al. (2019) RPD (%)	Lopez-GRASP Lopez-Esteve et al. (2023) RPD (%)
50	10	**3.39**	5.07	26.93	20.03
	15	**0.82**	16.86	53.90	48.13
	20	**0.00**	21.19	63.47	63.29
	25	**0.20**	17.24	56.90	56.22
	30	**2.52**	9.21	38.84	57.31
100	10	**1.23**	9.10	46.22	25.16
	15	**0.11**	32.07	98.13	70.19
	20	**0.00**	54.59	133.56	112.99
	25	**0.00**	63.32	151.86	140.95
	30	**0.00**	68.93	161.98	163.59
150	10	**0.97**	13.01	57.51	20.83
	15	**0.00**	38.94	114.08	77.44
	20	**0.00**	58.70	152.82	122.65
	25	**0.00**	72.57	167.17	148.08
	30	**0.00**	75.89	177.90	164.97
200	10	**1.05**	14.59	58.86	20.87
	15	**0.02**	42.77	119.59	80.60
	20	**0.00**	67.27	154.10	124.15
	25	**0.00**	70.61	169.71	145.80
	30	**0.00**	72.12	177.75	163.98
250	10	**0.71**	13.82	59.27	21.53
	15	**0.00**	47.78	125.37	86.09
	20	**0.00**	63.90	157.33	126.24
	25	**0.00**	70.36	168.33	143.14
	30	**0.00**	71.52	178.92	155.96

this average repair time was then added to the base runtime of $t = n$ seconds assigned to the reference algorithms (B-GRASP (Yepes-Borrero et al., 2020), Báez-Hybrid Báez et al. (2019), and Lopez-GRASP Lopez-Esteve et al. (2023)), so that all methods were executed under equivalent total runtimes.

Table 3 shows the average RPD results for all algorithms under different numbers of jobs and machines. E-GRASP consistently outperforms the other algorithms across all tested instance sizes, demonstrating robustness and efficiency regardless of the problem size. Although Báez et al. (2019) and Lopez-GRASP (Lopez-Esteve et al., 2023) were included in the comparison, their performance was significantly worse than that of E-GRASP and B-GRASP (Yepes-Borrero et al., 2020). As shown in Table 3, their RPD values are considerably higher. This can be explained by the fact that these algorithms were originally designed for related problems, and their adaptations do not properly address the specific characteristics

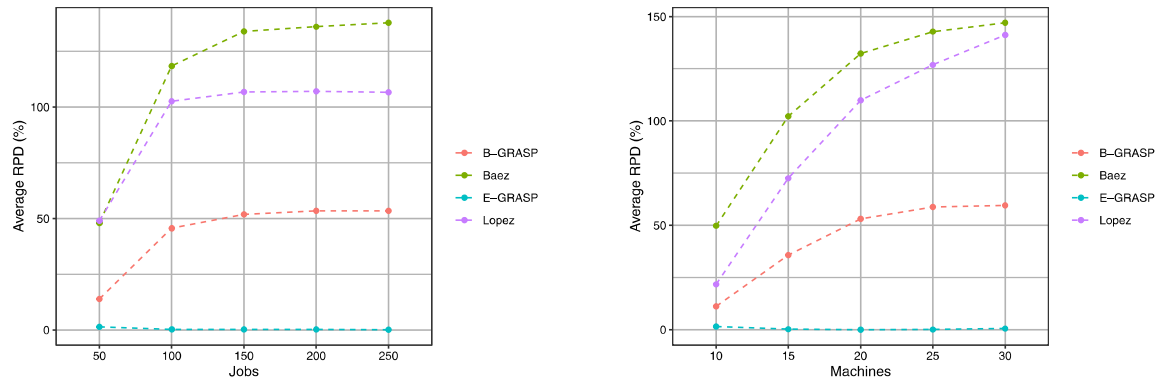


Fig. 4 Evolution of the RPD for all algorithms as a function of jobs (Figure 4a) and machines (Figure 4b)

of the UPMSR-S. These results underline the importance of designing specific algorithms for the UPMSR-S, with mechanisms that explicitly handle the type of additional resources involved in this problem.

Furthermore, while E-GRASP already outperforms all other methods, its advantage becomes even more pronounced as the number of machines increases. This is particularly noticeable in cases with 20 or more machines, where the RPD of E-GRASP is significantly lower, or even equal to 0 in many instances, meaning that it finds the best solutions consistently. On the other hand, B-GRASP (Yepes-Borrero et al., 2020), Báez-Hybrid Báez et al. (2019), and Lopez-GRASP Lopez-Esteve et al. (2023) yield much higher RPD values, with B-GRASP (Yepes-Borrero et al., 2020) being the closest competitor yet still substantially inferior.

Another important observation is that the E-GRASP is especially effective in larger instances. In configurations with 100 or more jobs, the proposed algorithm maintains low RPD values, while the other algorithms show a substantial increase, suggesting that the new approach handles the increase in problem size much more efficiently.

Although the superiority of E-GRASP over the other algorithms is evident from the results, we also conducted a hypothesis test for the difference in means. As expected, the test confirmed that there are statistically significant differences between the mean RPD of E-GRASP and that of B-GRASP (Yepes-Borrero et al., 2020), further reinforcing the advantage of the proposed approach.

It is also worth noting that, for smaller instances (such as those with 50 jobs and 10 machines), E-GRASP remains highly competitive. Although the differences between the algorithms are less pronounced in these less difficult cases, E-GRASP still outperforms B-GRASP (Yepes-Borrero et al., 2020) and maintains a considerable advantage over Báez et al. (2019) and Lopez-Esteve et al. (2023).

Figure 4 presents the average RPD for the four algorithms, varying the number of jobs and machines. Even for instances with 50 jobs, the differences between E-GRASP and the other methods are already significant. However, as the number of jobs increases, these differences become even more pronounced. For instances with 150 or more jobs, E-GRASP achieves average RPD values close to 0%, while the other algorithms exhibit RPD values exceeding 50%, even in the case of B-GRASP (Yepes-Borrero et al., 2020), which performs better than Báez et al. (2019) and Lopez-GRASP (Lopez-Esteve et al., 2023) but still far from the results of E-GRASP. A similar trend is observed when increasing the number of machines. With 10 machines, the differences are smaller, but as the number of machines grows, E-GRASP

Table 4 Paired t-test p-values on the RPD values comparing E-GRASP and B-GRASP across job sizes

Jobs	p-value
50	< 0.001
100	< 0.001
150	< 0.001
200	< 0.001
250	< 0.001

Table 5 Paired t-test p-values on the RPD values comparing E-GRASP and B-GRASP across numbers of machines

Machines	p-value
10	< 0.001
15	< 0.001
20	< 0.001
25	< 0.001
30	< 0.001

consistently maintains RPD values near 0%, while B-GRASP (Yepes-Borrero et al., 2020) reaches almost 60% for instances with 25 and 30 machines.

This behaviour can be explained by the nature of larger instances, particularly those with more machines, where a greater number of simultaneous setups must be managed across different machines, increasing the computational effort required by the solution repair process. Similarly, instances with a higher number of jobs require a greater number of setups, further complicating repairs. The results suggest that E-GRASP significantly outperforms the other methods by adding improvements that simplify the repair process. By repairing fewer solutions, the enhanced algorithm allocates more time to performing additional iterations, leading to a more extensive exploration of the solution space. Additionally, the penalization of sequences with longer setup times and higher resource consumption (through the β factor) simplifies the repairing process, contributing to the superior performance of E-GRASP across all tested scenarios.

To statistically verify the differences between the two best performing algorithms, we performed paired t-tests on the RPD values of E-GRASP and B-GRASP, grouping the instances by the number of jobs and by the number of machines. Tables 4 and 5 show the corresponding p-values for each group. In all cases, the results confirm that the differences between the two algorithms are statistically significant ($p < 0.001$).

5.3 Sensitivity to resource availability

To further analyze the influence of setup-resource availability on algorithmic performance, a dedicated sensitivity study was conducted. A representative subset of instance sizes was selected, namely (50, 10), (50, 15), (100, 15), (150, 20), (200, 25), and (250, 30), and for each size five random instances were generated.

The number of renewable resource units was varied across seven levels relative to the number of machines: $R_{\max} = 0.1\,m$, $R_{\max} = 0.2\,m$, $R_{\max} = 0.3\,m$, $R_{\max} = 0.4\,m$, $R_{\max} = 0.5\,m$, $R_{\max} = 0.6\,m$, and $R_{\max} = 0.7\,m$. These levels were chosen to represent scenarios ranging from highly restrictive ($0.1\,m$) to less restrictive ($0.7\,m$). The setup resource requirements r_{ijk} were generated as in the original benchmark instances, with values

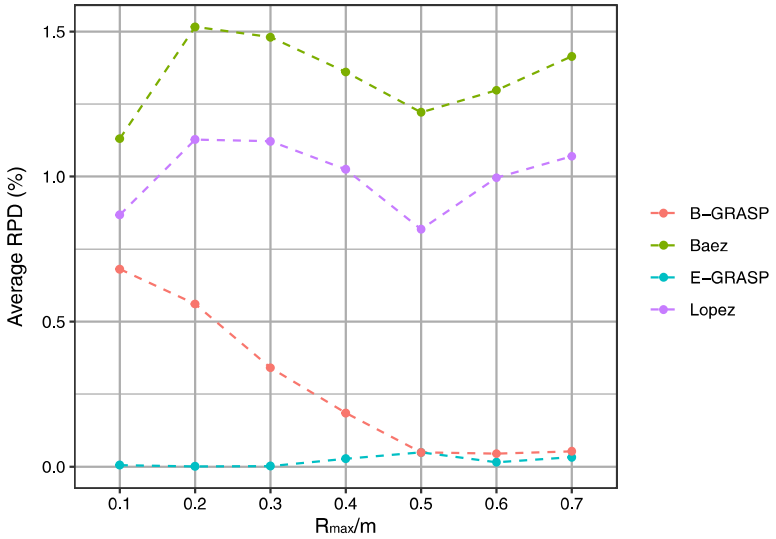


Fig. 5 Evolution of the RPD for all algorithms as a function of the resource availability ratio $R_{\max}/m$

sampled uniformly from $[1, R_{\max}]$. In total, 210 instances were generated for this sensitivity analysis.

Table 6 and Figure 5 summarize the mean relative percentage deviations obtained for different values of $R_{\max}$. It can be observed that when the resource is highly restrictive ($R_{\max} = 0.1m$–$0.3m$), E-GRASP clearly outperforms all other algorithms, showing differences of more than one order of magnitude in the mean RPD with respect to the next best approach. As the resource becomes less restrictive, the advantage of E-GRASP gradually decreases, and from $R_{\max} = 0.5m$ onward its performance becomes very similar to that of B-GRASP (Yepes-Borrero et al., 2020). This behaviour is consistent with the design of E-GRASP, which explicitly targets the setup-resource infeasibility during construction and repair. When resources are scarce, these mechanisms provide an important advantage, whereas when $R_{\max}$ increases and the problem becomes less restrictive, the repair phase is easier and the advantage of E-GRASP becomes less pronounced.

It is worth noting that E-GRASP remains competitive even in the least restrictive cases ($R_{\max} = 0.5m$–$0.7m$), indicating that the additional mechanisms do not degrade performance under less restrictive conditions.

Regarding the other algorithms, their performance is consistently worse across all configurations. Although Báez-Hybrid (Báez et al., 2019) and Lopez-GRASP (Lopez-Esteve et al., 2023) are strong methods for similar unrelated parallel machine scheduling problems, they are not tailored to handle the renewable setup-resource constraint. Their solutions yield much higher RPD values even in moderately restrictive scenarios, highlighting the importance of explicitly considering setup-resource feasibility in this problem variant.

It is worth emphasizing that modelling a small number of setup resources is consistent with real industrial conditions. In many manufacturing environments, such as ceramics, textiles, or electronics, the number of specialized operators or tooling units is usually small compared with the number of production machines, since such resources are expensive and shared across several lines. This assumption is also consistent with recent literature, where both Avgerinos et al. (2023) and Heinz et al. (2022) consider problems in which the number of setup resources is very limited compared with the number of machines. These studies confirm that relatively low resource levels are representative of practical and research-relevant settings. Therefore, testing $R_{\max}$ ratios between $0.1m$ and $0.7m$ provides a realistic spectrum of scenarios ranging from highly restrictive to moderately relaxed conditions.

Table 6 Mean RPD by $R_{\max}$ level. The best value for each row is highlighted in bold.

$R_{\max}/m$	E-GRASP	B-GRASP Yepes-Borrero et al. (2020)	Báez-Hybrid Báez et al. (2019)	Lopez-GRASP Lopez-Esteve et al. (2023)
0.10	**0.57**	68.08	113.09	86.81
0.20	**0.14**	56.08	151.60	112.81
0.30	**0.24**	34.10	148.07	112.18
0.40	**2.78**	18.48	136.08	102.53
0.50	**4.91**	**4.91**	122.17	81.88
0.60	**1.58**	4.51	129.77	99.59
0.70	**3.27**	5.32	141.45	107.01

Table 7 Average performance of E-GRASP and the MILP model on small instances.

		MILP				E-GRASP		
Jobs	Machines	RPD	RPD(gap)	Time (s)	Opt. (%)	RPD	RPD(gap)	Time (s)
6	2	0.00	0.00	16.01	100.00	5.53	5.53	6.00
	3	0.00	0.00	7.18	100.00	2.41	2.41	6.01
8	2	0.00	0.00	811.55	100.00	5.09	5.09	8.00
	3	0.00	0.00	153.58	100.00	5.41	5.41	8.01
10	2	3.15	58.40	4934.12	50.00	1.64	58.14	10.01
	3	1.36	21.16	3046.49	83.33	2.33	19.02	10.02
12	2	10.31	73.42	7200.00	0.00	0.65	71.72	12.01
	3	10.12	49.82	7200.00	0.00	0.40	46.43	12.02

5.4 Comparison with the MILP model on small instances

To further evaluate the quality and computational efficiency of the proposed heuristic, we compared E-GRASP with the exact MILP model presented in Section 3. The MILP was solved using Gurobi 12.0.3 with a time limit of 7200 seconds and a relative MIP gap tolerance of 10^{-4}. Because of the combinatorial nature of the problem, the MILP could only solve very small instances to proven optimality. We therefore considered eight configurations, with the number of jobs $\in \{6, 8, 10, 12\}$ and the number of machines $\in \{2, 3\}$, and generated six instances for each configuration ($\{(j, m) : j \in jobs, m \in machines\}$).

Table 7 reports, for each configuration, the average relative percentage deviation (RPD) and the deviation with respect to the MILP lower bound, denoted as RPD(gap), for both the MILP and E-GRASP. The column *Opt.%* indicates the percentage of instances for which the MILP reached proven optimality, while the last two columns for MILP and E-GRASP show the average computational times in seconds.

The MILP solved all instances with six and eight jobs to proven optimality. In these cases, E-GRASP remained close to the optimal solutions, with average RPD values below 6%. For the 8-job configurations with either 2 or 3 machines, however, the MILP required considerable computation time, on average 811 s and 154 s per instance, whereas E-GRASP obtained comparable solutions in only 8 s.

As the instance size increases, the relative performance of both methods starts to diverge. For the configurations consisting of 10 jobs and 2 machines, E-GRASP produced better solutions on average than the MILP, with an average runtime of just 10 s compared with nearly 5000 s for the MILP. For the configurations consisting of 10 jobs and 3 machines, the MILP reached optimality in 83% of the instances, while E-GRASP achieved a mean RPD of only 2.33%, showing that it can closely approximate optimal solutions even when the problem becomes harder.

For the largest configurations, consisting of 12 jobs and either 2 or 3 machines, the MILP failed to prove optimality in any instance and produced very loose bounds even after the full 7200 s limit, whereas E-GRASP found much better solutions within a few seconds. The large MILP gaps observed in these cases confirm that the linear relaxation yields weak bounds and that exact approaches become unreliable for such instance sizes.

Overall, these results show that E-GRASP maintains good solution quality across all instance sizes while requiring negligible computation time, making it a practical and efficient alternative to exact optimization.

6 Conclusions and further research

In this paper, we have introduced a new efficient metaheuristic algorithm for solving the Unrelated Parallel Machine Scheduling problem with setup times and additional Resources in the Setups (UPMSR-S). The proposed algorithm has been thoroughly evaluated through an extensive computational campaign on a benchmark of 1000 instances. Furthermore, the algorithm parameters were rigorously calibrated through a series of exhaustive experiments, ensuring that the best values were selected to maximize its performance.

The results of these experiments, supported by statistical analysis, demonstrate that the new approach significantly outperforms all compared algorithms, with performance improvements exceeding 70% in several cases. This highlights the robustness of the algorithm across different instance sizes and its ability to adapt to the increasing difficulty of the UPMSR-S problem.

One of the key advantages of this algorithm lies in its practical relevance. By efficiently managing limited setups and resources, the enhanced algorithm addresses a critical need in real-world production environments, where resource constraints often represent a significant challenge. The algorithm's ability to deliver high-quality solutions in these restricted environments underscores its value for practical applications, making it a useful tool for industries where scheduling efficiency is vital.

As future work, the approach could be extended to other scheduling problems involving resource constraints. In particular, the proposed repair mechanism is not tied to the specific structure of the UPMSR-S and can be adapted to other settings where infeasibilities arise from limited renewable resources. This includes problems such as flowshop and jobshop scheduling. Additionally, alternative objectives such as minimizing tardiness or the total (weighted) completion time could be explored, and the method could be adapted to multi-objective formulations to assess its performance across a wider range of scheduling scenarios.

Funding Open access funding provided by FEDER European Funds and the Junta de Castilla y León under the Research and Innovation Strategy for Smart Specialization (RIS3) of Castilla y León 2021-2027. J. C. Yepes-Borrero was partially supported by the Spanish Ministry of Science and Innovation under the project "OPRES-Realistic Optimization in Problems in Public Health" (No. PID2021-124975OB-I00), partially financed with FEDER funds.

J. Alcaraz was supported by grants PID2022-136383NB-I00 funded by MICIN/AEI/10.13039/501100011033 and by ERDF/EU, and CIPROM/2024/34, funded by the Conselleria de Educación, Cultura, Universidades y Empleo, Generalitat Valenciana.
This article has received funding from the European Union's Horizon Europe research and innovation actions under grant agreement No. 101168560 (CoEvolution). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the Commission. Neither the European Union nor the granting authority can be held responsible for them.

Declarations

Conflict of Interest J. C. Yepes-Borrero, J. Alcaraz, M. López-García, and M. Villaizán-Vallelado declare that they have no conflict of interest.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

Informed consent Not applicable.

A Example instance and step-by-step E-GRASP of the UPMSR-S problem

A.1 Example Instance of the UPMSR-S Problem

To illustrate the proposed metaheuristic Enhanced GRASP (E-GRASP), we consider a specific instance of the Unrelated Parallel Machine scheduling problem with setup times and additional Resources in the Setups (UPMSR-S). In this instance, a set N of $n = 8$ jobs must be scheduled on a set M of $m = 4$ unrelated machines. Each job must be processed exactly once on one of the available machines, and each machine can process only one job at a time without pre-emption.

The processing time of job j on machine i is denoted by p_{ij} (Figure 6a), which varies across machines due to their heterogeneous characteristics. Moreover, whenever two jobs are processed consecutively on the same machine, a setup operation is required. The setup time s_{ijk} depends on the sequence of jobs, that is, on the ordered pair (j, k) (Figure 6b).

In addition to time-related constraints, setup operations consume limited auxiliary resources (e.g., labour, tools, or material handling equipment). The amount of resource required for the setup between jobs j and k on machine i is represented by r_{ijk} (Figure 6c). The total resource consumption at any given time cannot exceed the maximum available capacity, which in this example is fixed at $R_{\max} = 3$.

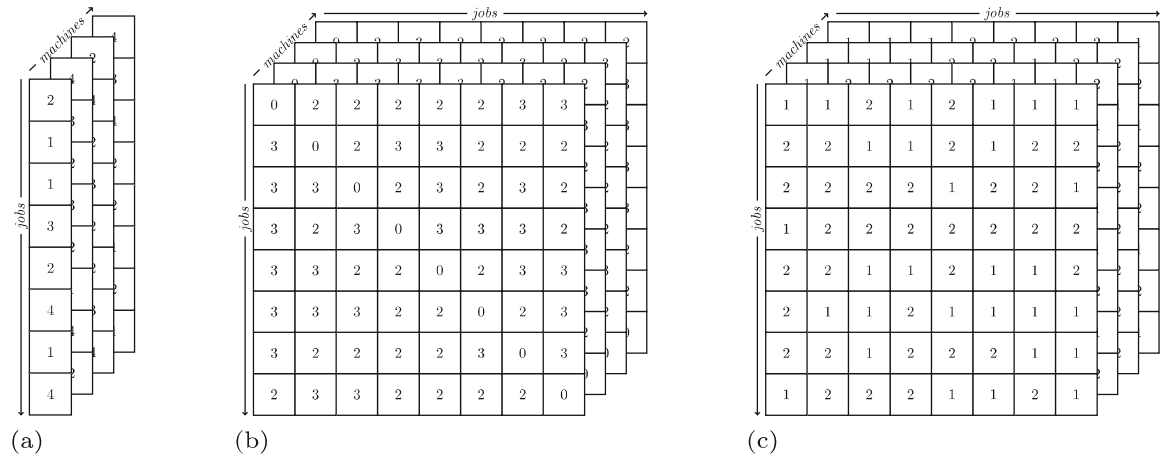


Fig. 6 Example data of a UPMSR-S problem with $R_{\max} = 3$, 8 jobs and 4 machines. In Figure 6a processing times of each job on each machine (p_{ij}). In Figure 6b setup times between a job (row) and its successor (column) on each machine (s_{ijk}). In Figure 6c resources between a job (row) and its successor (column) on each machine (r_{ijk})

A.2 Step-by-step of the E-GRASP metaheuristic

In the following sections, each component of the proposed E-GRASP algorithm is illustrated through specific examples. These components correspond to the main phases outlined in Figure 2, which summarizes the overall structure of the method. The phases include: the construction phase (Appendix A.2.1), the local search (Appendix A.2.2), the addition of solutions to the promising list (Appendix A.2.3), and the repair phase (Appendix A.2.4).

A.2.1 Construction phase

The construction phase, previously described in Section 4.1, is responsible for generating initial solutions. Its detailed procedure is described in Algorithm 1. In this section, we provide an illustrative example to clarify how the last job is inserted into a partially constructed solution.

Figure 7 illustrates an intermediate stage of the construction phase, where the last job (7) is being evaluated for insertion into each partial solution. The figure shows the computation of the evaluation criterion for each possible position, which corresponds to the values given by Equation (10). In this example, the parameter β is set to the best value obtained across the experiments reported in Section 5.1 ($\beta = 4$). The computed values in Figure 7 of the insertion of job 7 according to Equation (10) are:

$$\begin{aligned}\lambda_{i,j,k}^{\beta=4} = \{&\lambda_{1,7,0}^{4} = 31, \lambda_{1,7,1}^{4} = 19, \lambda_{1,7,2}^{4} = 19,\\ &\lambda_{2,7,0}^{4} = 20, \lambda_{2,7,1}^{4} = 16, \lambda_{2,7,2}^{4} = 24,\\ &\lambda_{3,7,0}^{4} = 34, \lambda_{3,7,1}^{4} = 34, \lambda_{3,7,2}^{4} = 34,\\ &\lambda_{4,7,0}^{4} = 30, \lambda_{4,7,1}^{4} = 14\}. \end{aligned} \tag{13}$$

Suppose that $\alpha = 0.1$. The higher α value therefore promotes greater exploration by enabling multiple feasible insertions at this stage. The set of insertions that would be consid-

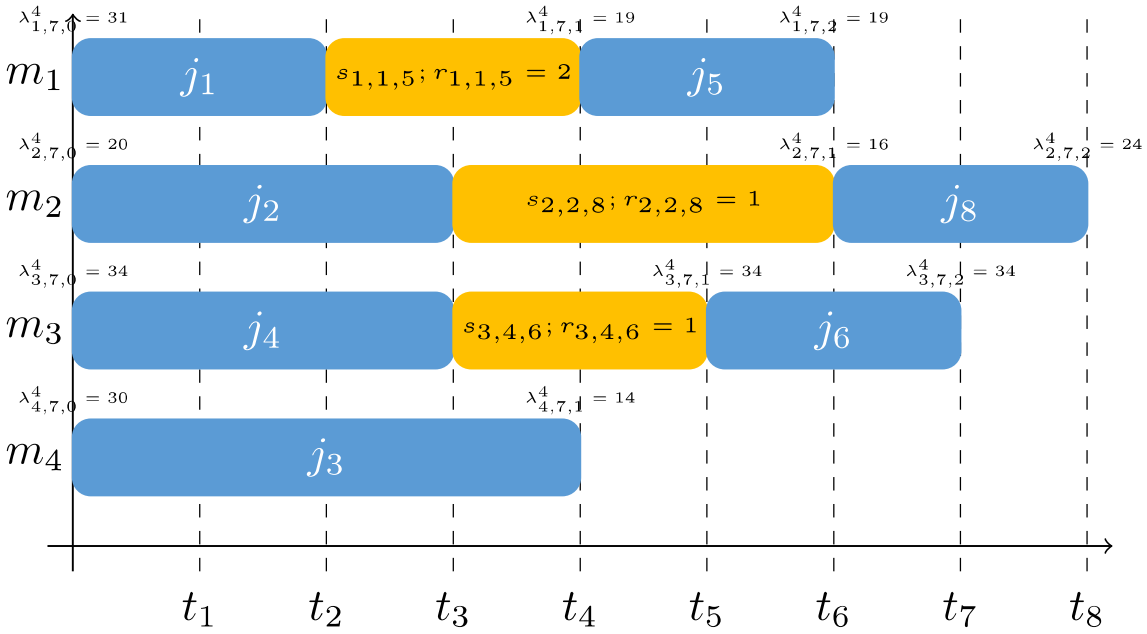


Fig. 7 Results obtained from the evaluation of the insertion of job 7 according to Equation (10), considering $\beta = 4$

ered after applying the filtering process to the list of potential candidates is given by:

$$\begin{aligned} & RCL\ (\alpha = 0.10) \\ & = \left\{(i, j, k) : \lambda^4_{i,j,k} \leq \min\left(\lambda^4_{i,j,k}\right) + 0.10\left(\max\left(\lambda^4_{i,j,k}\right) - \min\left(\lambda^4_{i,j,k}\right)\right)\right\} \\ & = \left\{(i, j, k) : \lambda^4_{i,j,k} \leq \lambda^4_{4,7,1} + 0.10 \cdot \left(\lambda^4_{3,7,0} - \lambda^4_{4,7,1}\right)\right\} \\ & = \left\{\lambda^4_{2,7,1} = 16, \lambda^4_{4,7,1} = 14\right\}. \end{aligned} \tag{14}$$

Figure 8 presents the resulting solution after inserting job 7 into the candidate positions identified in Equation (14). In particular, Figure 8a illustrates insertion $\lambda^4_{2,7,1}$, which shows the inclusion of job 7 on machine 2 between jobs 2 and 8. Figure 8b illustrates $\lambda^4_{4,7,1}$, corresponding to the placement of job 7 on machine 4 immediately after job 3. Both configurations represent feasible alternatives derived from the candidate set defined in Equation (14), and together they exemplify how the algorithm evaluates different insertion positions before completing the construction phase. The final choice between the feasible insertions $(\lambda^4_{2,7,1}, \lambda^4_{4,7,1})$ is determined by GRASP, which balances greediness and randomness to guide the construction of the solution.

A.2.2 Local search

The local search phase, previously described in Section 4.2, is responsible for improving the generated solution by performing job assignment swaps aimed at reducing the makespan. Its detailed procedure is described in Algorithm 2.

An example of this local search process is illustrated in Figure 8. If the construction mechanism had generated the solution depicted in Figure 8a, a possible improving swap would consist of assigning job 7 after job 3 (Figure 8b).

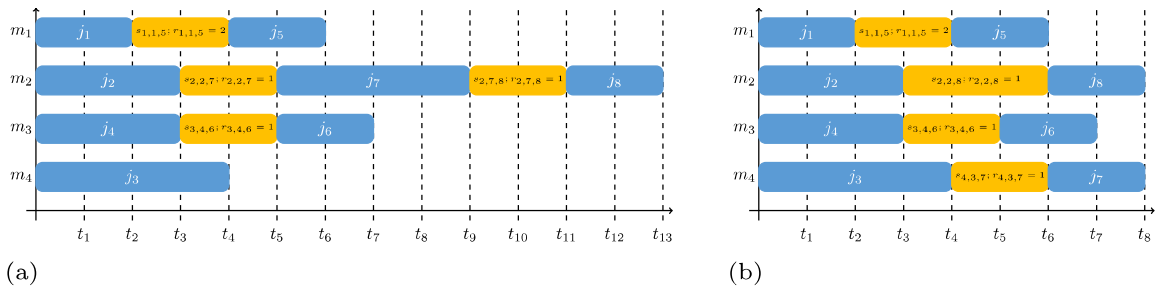


Fig. 8 The set of candidate solutions used for selecting the insertion position of job 7 is illustrated. Figure 8a shows the insertion of job 7 on machine 2 between jobs 2 and 8 ($\lambda^4_{2,7,1}$), while Figure 8b shows the insertion on machine 4 after job 3 ($\lambda^4_{4,7,1}$)

A.2.3 Add to promising list

The promising list phase is responsible for selecting the most promising solutions among those explored during the previous iterations. In this phase, only the solutions that exhibit a smaller makespan compared to the previously evaluated ones are retained for the subsequent repairing phase. According to the analysis presented in Section 5.1, the size of the promising list is set to ten elements ($\rho = 10$). This filtering mechanism ensures that the computational effort of the repairing process is focused exclusively on high-quality solutions.

A.2.4 Repairing phase

The repair phase, previously described in Section 4.3, ensures that the solutions generated during the previous stages satisfy the resource constraint. This is achieved by evaluating the total resource consumption at each time instant and applying corrective actions whenever the constraint is violated. Specifically, when an infeasibility is detected, the algorithm postpones by one time unit the start of the setup operation that began last among those active at that instant. This repair mechanism is summarized in Algorithm 3.

An example of this repair procedure is illustrated in Figure 1. In Figure 1a, the solution is infeasible because the resource constraint is violated between time instants t_3 and t_4. After applying the repair mechanism, the adjusted solution shown in Figure 1b becomes feasible, as the setups are delayed to ensure that the resource consumption does not exceed the limit $R_{\max} = 3$.

References

Avgerinos, I., Mourtos, I., Vatikiotis, S., & Zois, G. (2023). Scheduling unrelated machines with job splitting, setup resources and sequence dependency. *International Journal of Production Research, 61*(16), 5502–5524. https://doi.org/10.1080/00207543.2022.2102948

Báez, S., Angel-Bello, F., Alvarez, A., & Melián-Batista, B. (2019). A hybrid metaheuristic algorithm for a parallel machine scheduling problem with dependent setup times. *Computers & Industrial Engineering, 131*, 295–305. https://doi.org/10.1016/j.cie.2019.03.051

Berthier, A., Yalaoui, A., Chehade, H., Yalaoui, F., Amodeo, L., & Bouillot, C. (2022). Unrelated parallel machines scheduling with dependent setup times in textile industry. *Computers & Industrial Engineering, 174*, Article 108736. https://doi.org/10.1016/j.cie.2022.108736

Chen, J. C., Chen, T.-L., Chen, Y.-Y., & Chung, M.-Y. (2024). Multi-resource constrained scheduling considering process plan flexibility and lot streaming for the cnc machining industry. *Flexible Services and Manufacturing Journal, 36*(3), 946–993. https://doi.org/10.1007/s10696-023-09514-w

Chen, J., Chu, C., Sahli, A., & Li, K. (2024). A branch-and-price algorithm for unrelated parallel machine scheduling with machine usage costs. *European Journal of Operational Research, 316*(3), 856–872. https://doi.org/10.1016/j.ejor.2024.03.011

Durasević, M., & Jakobović, D. (2023). Heuristic and metaheuristic methods for the parallel unrelated machines scheduling problem: a survey. *Artificial Intelligence Review, 56*(4), 3181–3289. https://doi.org/10.1007/s10462-022-10247-9

Elidrissi, A., Benmansour, R., & Sifaleras, A. (2023). General variable neighborhood search for the parallel machine scheduling problem with two common servers. *Optimization Letters, 17*(9), 2201–2231. https://doi.org/10.1007/s11590-022-01925-2

Fang, W., Zhu, H., & Mei, Y. (2022). Hybrid meta-heuristics for the unrelated parallel machine scheduling problem with setup times. *Knowledge-Based Systems, 241*, Article 108193. https://doi.org/10.1016/j.knosys.2022.108193

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*, Article 100022. https://doi.org/10.1016/j.eswax.2020.100022

Fanjul-Peyró, L., Ruiz, R., & Perea, F. (2019). Reformulations and an exact algorithm for unrelated parallel machine scheduling problems with setup times. *Computers & Operations Research, 101*, 173–182. https://doi.org/10.1016/j.cor.2018.07.007

Feo, T. A., & Resende, M. G. C. (1989). A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters, 8*(2), 67–71. https://doi.org/10.1016/0167-6377(89)90002-3

Florescu, A., & Barabas, S. (2022). Development trends of production systems through the integration of lean management and industry 4.0. *Applied Sciences, 12*(10), 4885. https://doi.org/10.3390/app12104885

Fonseca, G. H. G., Figueiroa, G. B., & Toffolo, T. A. M. (2024). A fix-and-optimize heuristic for the unrelated parallel machine scheduling problem. *Computers & Operations Research, 163*, Article 106504. https://doi.org/10.1016/j.cor.2023.106504

Geurtsen, M., Didden, J. B. H. C., Adan, J., Atan, Z., & Adan, I. (2023). Production, maintenance and resource scheduling: A review. *European Journal of Operational Research, 305*(2), 501–529. https://doi.org/10.1016/j.ejor.2022.03.045

Heinz, V., Novák, A., Vlk, M., & Hanzálek, Z. (2022). Constraint programming and constructive heuristics for parallel machine scheduling with sequence-dependent setups and common servers. *Computers & Industrial Engineering, 172*, Article 108586. https://doi.org/10.1016/j.cie.2022.108586

Klein, N. (2025). Integer programming for multi-mode resource-constrained project scheduling. *Annals of Operations Research*. https://doi.org/10.1007/s10479-025-06572-1

Klein, N., Gnägi, M., & Trautmann, N. (2024). Mixed-integer linear programming for project scheduling under various resource constraints. *European Journal of Operational Research, 319*(1), 79–88. https://doi.org/10.1016/j.ejor.2024.06.036

Lee, J.-H., & Jang, H. (2019). Uniform parallel machine scheduling with dedicated machines, job splitting and setup resources. *Sustainability, 11*(24), 7137. https://doi.org/10.3390/su11247137

Lei, D., Yuan, Y., & Cai, J. (2021). An improved artificial bee colony for multi-objective distributed unrelated parallel machine scheduling. *International Journal of Production Research, 59*(17), 5259–5271. https://doi.org/10.1080/00207543.2020.1775911

Lenstra, J. K., & Rinnooy Kan, A. H. G. (1977). Brucker, P.: Complexity of machine scheduling problems. Annals of Discrete Mathematics 1(C), 343–362 https://doi.org/10.1016/S0167-5060(08)70743-X

Li, K., Chen, J., Fu, H., Jia, Z., & Fu, W. (2019). Uniform parallel machine scheduling with fuzzy processing times under resource consumption constraint. *Applied Soft Computing, 82*, Article 105585. https://doi.org/10.1016/j.asoc.2019.105585

Li, K., Xie, F., Chen, J., Xiao, W., & Zhou, T. (2024). Mathematical models and an effective exact algorithm for unrelated parallel machine scheduling with family setup times and machine cost. *OR Spectrum*. https://doi.org/10.1007/s00291-024-00778-8

Lopez-Esteve, A., Perea, F., & Yepes-Borrero, J. C. (2023). GRASP algorithms for the unrelated parallel machines scheduling problem with additional resources during processing and setups. *International Journal of Production Research, 61*(17), 6013–6029. https://doi.org/10.1080/00207543.2022.2121869

Luo, D., Thevenin, S., & Dolgui, A. (2023). A state-of-the-art on production planning in industry 4.0. *International Journal of Production Research, 61*(19), 6602–6632. https://doi.org/10.1080/00207543.2022.2122622

Mahmoodi, E., Fathi, M., Tavana, M., Ghobakhloo, M., & Ng, A. H. C. (2024). Data-driven simulation-based decision support system for resource allocation in industry 4.0 and smart manufacturing. *Journal of Manufacturing Systems, 72*, 287–307. https://doi.org/10.1016/j.jmsy.2023.11.019

Montgomery, D. C. (2019). *Design and Analysis of Experiments*. Hoboken, NJ, USA: John Wiley & Sons Inc.

Mor, B., & Berlińska, J. (2025). Scheduling problems on parallel dedicated machines with non-renewable resource. *Annals of Operations Research, 346*(3), 2173–2193. https://doi.org/10.1007/s10479-025-06471-5

Mor, B., Mosheiov, G., & Shabtay, D. (2025). Scheduling problems on parallel machines with machine-dependent generalized due-dates. *Annals of Operations Research, 347*(3), 1455–1471. https://doi.org/10.1007/s10479-025-06468-0

Munoz, L., Villalobos, J. R., & Fowler, J. W. (2022). Exact and heuristic algorithms for the parallel machine total completion time scheduling problem with dual resources, ready times, and sequence-dependent setup times. *Computers & Operations Research, 143*, Article 105787. https://doi.org/10.1016/j.cor.2022.105787

Perrachon, Q., Olteanu, A.-L., Sevaux, M., Fréchengues, S., & Kerviche, J.-F. (2025). Industrial multi-resource flexible job shop scheduling with partially necessary resources. *European Journal of Operational Research, 320*(2), 309–327. https://doi.org/10.1016/j.ejor.2024.07.023

Saraç, T., & Tutumlu, B. (2022). A bi-objective mathematical model for an unrelated parallel machine scheduling problem with job-splitting. *Journal of the Faculty of Engineering and Architecture of Gazi University, 37*(4), 2293–2307. https://doi.org/10.17341/gazimmfd.967343

Shafiee, M., Amiri-Aref, M., & Klibi, W. (2025). The integration of shared renewable resources considering setup times for the parallel machine scheduling problem. *Computers & Industrial Engineering, 200*, Article 110828. https://doi.org/10.1016/j.cie.2024.110828

Srinath, N., Yilmazlar, I. O., Kurz, M. E., & Taaffe, K. (2023). Hybrid multi-objective evolutionary meta-heuristics for a parallel machine scheduling problem with setup times and preferences. *Computers & Industrial Engineering, 185*, Article 109675. https://doi.org/10.1016/j.cie.2023.109675

Stefansdottir, B., Grunow, M., & Akkerman, R. (2017). Classifying and modeling setups and cleanings in lot sizing and scheduling. *European Journal of Operational Research, 261*(3), 849–865. https://doi.org/10.1016/j.ejor.2017.03.023

Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research, 64*(2), 278–285. https://doi.org/10.1016/0377-2217(93)90182-M

Vallada, E., & Ruiz, R. (2011). A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. *European Journal of Operational Research, 211*(3), 612–622. https://doi.org/10.1016/j.ejor.2011.01.011

Vallada, E., Villa, F., & Fanjul-Peyro, L. (2019). Enriched metaheuristics for the resource constrained unrelated parallel machine scheduling problem. *Computers & Operations Research, 111*, 415–424. https://doi.org/10.1016/j.cor.2019.07.016

Villa, F., Vallada, E., & Fanjul-Peyró, L. (2018). Heuristic algorithms for the unrelated parallel machine scheduling problem with one scarce additional resource. *Expert Systems with Applications, 93*, 28–38. https://doi.org/10.1016/j.eswa.2017.09.054

Yan, X., Wang, T., & Shi, X. (2025). Optimal scheduling on unrelated parallel machines with combinatorial auction. *Annals of Operations Research, 344*(2), 937–963. https://doi.org/10.1007/s10479-024-06283-z

Yazdani, M., & Haghani, M. (2024). Exploring the evolution of machine scheduling through a computational approach. *Engineering Applications of Artificial Intelligence, 133*, Article 108572. https://doi.org/10.1016/j.engappai.2024.108572

Yepes-Borrero, J. C., Villa, F., Perea, F., & 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*, Article 112959. https://doi.org/10.1016/j.eswa.2019.112959

Yepes-Borrero, J. C., Perea, F., Ruiz, R., & Villa, F. (2021). Bi-objective parallel machine scheduling with additional resources during setups. *European Journal of Operational Research, 292*(2), 443–455. https://doi.org/10.1016/j.ejor.2020.10.052

Yilmaz Eroglu, D., Ozmutlu, H. C., & Ozmutlu, S. (2014). Genetic algorithm with local search for the unrelated parallel machine scheduling problem with sequence-dependent set-up times. *International Journal of Production Research, 52*(19), 5841–5856. https://doi.org/10.1080/00207543.2014.920966

Ying, K.-C., Pourhejazy, P., & Huang, X.-Y. (2024). Revisiting the development trajectory of parallel machine scheduling. *Computers & Operations Research, 168*, Article 106709. https://doi.org/10.1016/j.cor.2024.106709

Zeng, C., Liu, J., & Li, Q. (2025). A constraint programming approach for resource-constrained flexible assembly flow shop scheduling problem with batch direct delivery. *Computers & Operations Research, 173*, Article 106855. https://doi.org/10.1016/j.cor.2024.106855