Computers & Operations Research 191 (2026) 107440

Contents lists available at ScienceDirect

Computers and Operations Research

journal homepage: www.elsevier.com/locate/cor

Exact algorithms for energy-constrained scheduling on identical parallel machines

Jean-François Côté [a], Giulia Dotti [b,*], Vinícius Loti de Lima [c,1], Carlo Alberto Magni [b], Manuel Iori [d]

[a] CIRRELT, Université Laval, 2325 Rue de la Terrasse, Québec, G1V 0A6, Canada
[b] School of Doctorate E4E (Engineering for Economics - Economics for Engineering), University of Modena and Reggio Emilia, Largo Marco Biagi 10, Modena, 41121, Italy
[c] Middle Mile Science, Amazon, Bellevue, WA, United States of America
[d] Department of Sciences and Methods for Engineering, University of Modena and Reggio Emilia, via Giovanni Amendola 2, Reggio Emilia, 42122, Italy

ARTICLE INFO

Keywords:
Identical parallel machine scheduling
Renewable resources
Energy-aware production planning
Mixed Integer Linear Programming (MILP)
Constraint programming (CP)
Branch-and-bound (B&B)
Exact algorithm

ABSTRACT

We address the problem of scheduling non-preemptive jobs on identical parallel machines under a single renewable resource constraint, with the objective of minimizing the makespan. This setting captures relevant applications in energy-aware production planning, where energy usage must not exceed a fixed limit at any time. We introduce two mathematical formulations and propose an exact algorithm that integrates a branch-and-bound, advanced bounding techniques, and a constraint programming model. Extensive computational experiments on two benchmark sets from the literature show that the proposed approach outperforms existing exact methods, solving more instances to optimality and achieving consistently smaller optimality gaps within limited computation time.

1. Introduction

Parallel Machine Scheduling Problems (PMSPs) are a fundamental class of combinatorial optimization problems with broad real-world applications in sectors such as manufacturing and cloud computing (Baker and Trietsch, 2019). PMSPs are typically classified based on machine types, with a well-studied subset being identical PMSPs, where all machines are identical and thus process jobs in the same amount of time. A typical identical PMSP involves scheduling a set of independent jobs on multiple identical machines. Each job has a specific processing time and must be completed without interruption. A well-researched objective in the literature is the minimization of makespan, defined as the completion time of the last job in the schedule. According to the three-field notation by Graham et al. (1979), we can refer to this problem as $P\|C_{\max}$. The simpler version with only two machines was proven $\mathcal{NP}$-hard in Lenstra et al. (1977).

To address practical requirements, PMSPs have been extended to include additional constraints, such as precedence relations and setup times. One important extension is the Resource-Constrained Parallel Machine Scheduling Problems (RCPMSPs), which incorporates limitations on the additional resources required for job processing. This scenario is common in manufacturing and logistics (Edis et al., 2013). Resources in RCPMSPs can be classified as either renewable (e.g., operators or tools that can be reused in successive across time intervals) or non-renewable resources (e.g., consumable that cannot be reused). Several studies, such as Fanjul-Peyro et al. (2017), have developed exact methods and heuristics to effectively address these problems.

In this paper, we focus on the identical PMSP with a single renewable resource, optimizing the makespan. Following the classification scheme introduced by Blazewicz et al. (1983), who extended Graham et al.'s (1979) notation, we denote this problem as $P|res1..|C_{\max}$. This problem is especially relevant in the context of Energy-Efficient Scheduling (EES), where energy can be modeled as a renewable resource. EES is a growing area of interest in the scheduling literature, as scheduling strategies play a critical role in optimizing energy usage (Gahm et al., 2016). Research in this domain typically follows two main approaches: minimizing total energy consumption and managing the distribution of energy demand. The first approach focuses on reducing overall energy usage by minimizing idle times and optimizing machine setup sequences. The second aims to balance energy demand over time, introducing mechanisms such as time-of-use pricing, where energy costs vary depending on the time of consumption. A critical aspect of energy demand management is regulating peak loads to respect

* Corresponding author.
E-mail address: giulia.dotti@unimore.it (G. Dotti).
[1] This work is unrelated to the author's affiliation with Amazon.

https://doi.org/10.1016/j.cor.2026.107440
Received 4 August 2025; Received in revised form 24 February 2026; Accepted 25 February 2026
Available online 26 February 2026

the limits set by energy contracts, as companies often face penalties for surpassing the maximum threshold (Módos et al., 2021). As a result, feasible schedules must ensure that the total energy consumption across all machines remains below this limit in each time interval. Therefore, energy can be treated as a single renewable resource, allowing us to frame the problem as an identical machine scheduling problem under energy constraints.

To address $P|res1..|C_{\max}$, we rely on its structural analogy with the closely related parallel processor scheduling problem with contiguity constraints ($P|cont|C_{\max}$). In this setting, each job must be assigned to a set of identical and contiguous parallel machines, with the objective of minimizing the makespan. In addition to being studied as an independent problem, $P|cont|C_{\max}$ also plays a central role in exact algorithms for packing problems, where it is often used as the master problem in Benders' decomposition approaches, such as the one proposed (Côté et al., 2014). A comprehensive overview of exact methods for packing problem is provided in Iori et al. (2021). Motivated by this connection, we adapt and extend several effective techniques from the packing literature to solve $P|res1..|C_{\max}$.

On this basis, this paper lies at the intersection of RCPMSPs with renewable resources and EES, focusing on optimizing makespan while incorporating energy constraints. Its major contributions are as follows:

1. we investigate the generalized version of the identical PMSP with renewable resources, presenting and comparing two modeling approaches: a Mixed Integer Linear Programming (MILP) model and a Constraint Programming (CP) model;
2. we develop enhanced bounds calculation techniques, including a column generation method to compute lower bounds and a simulated metaheuristic algorithm to compute upper bounds;
3. we propose a novel branch-and-bound (B&B) algorithm, incorporating dynamic programming (DP) cuts inspired by the cutting and packing literature, as well as a timetabling pruning algorithm from cumulative scheduling studies;
4. we design a comprehensive algorithm that combines multiple techniques, adapting it to the characteristics of specific problem instances;
5. we evaluate the performance of the proposed methods on benchmark instances from both the RCPMSP and the EES literature, demonstrating the superior computational efficiency of the comprehensive algorithm.

The structure of the paper is as follows. Section 2 reviews the relevant literature. Section 3 details the problem formulation and presents the mathematical models for optimization. Section 4 describes our algorithm, with a focus on the B&B approach. Section 5 analyzes the experimental results, and Section 6 concludes the study with suggestions for future research directions.

2. Literature review

The literature on PMSPs is extensive, encompassing different problem settings and solution approaches. This section reviews key contributions in the two areas relevant to our work. First, we focus on RCPMSP, highlighting studies that address identical and unrelated machines. Then, we review EES, focusing on strategies to manage energy demand under operational constraints. Finally, we provide a brief overview of the literature on $P|cont|C_{\max}$ and its connection to packing problems, as this domain offers algorithmic techniques that inform parts of our solution approach.

2.1. Resource-constrained parallel machine scheduling problems

The RCPMSP has been studied over the decades, with various approaches addressing different problem variants. In the following, we outline the key advancements in the field, focusing on identical and unrelated machine settings. For more details, we refer the interested reader to the surveys of Edis et al. (2013) and Geurtsen et al. (2023). In the following, we give more details that are related to our work.

In the case of identical parallel machines, research has focused primarily on problem-specific variants derived from real-world applications rather than on the general case. For example, Garey and Graham (1975) introduced the renewable resource setting in multiprocessor scheduling, incorporating precedence relationships. Later, Ventura and Kim (2003) explored a scenario with a single type of resource, limiting the resource usage of each job to at most one unit. Over time, researchers have introduced even more complex constraints. For example, Edis and Ozkarahan (2012) addressed eligibility constraints, which restrict certain jobs to be processed only on a subset of machines. They integrated Integer Linear Programming (ILP) and CP models by partitioning the problem into two subproblems: one for assigning jobs to eligible machines and another for sequencing the operations. Similarly, Caselli et al. (2024) combined ILP, CP, and Benders' decomposition to address constraints such as precedence, machine eligibility, and contiguity. However, the generalized version of the identical RCPMSP with one renewable resource remains relatively unexplored.

On the other hand, more studies can be found on the unrelated RCPMSP with a single additional resource, where both the processing time and resource consumption of each job depend on the machine. Fanjul-Peyro et al. (2017) proposed two ILP models and three matheuristic strategies for small and medium-sized instances. Their approach was later computationally outperformed by Fleszar and Hindi (2018), who developed a two-stage method: in the first stage, they combined MILP and CP to pre-assign jobs to machines and determine an initial schedule; in the second stage, if the solution was suboptimal, they refined it using a CP model with a hot start based on the first stage. This study also examined a two-machine version of the problem, introducing an MILP model inspired by the project scheduling problem with precedence constraints. The same set of instances was later tested by Villa et al. (2018), who proposed several heuristics based on two main strategies: either incorporating resource constraints during the constructive process or ignoring them initially and repairing the solution afterward. Additionally, Vallada et al. (2019) introduced an enriched scatter search and an enhanced iterated greedy metaheuristic algorithm. In this work, we compare our approach against the exact results obtained for this set of instances, adapting them to the scenario with identical parallel machines. The recent work of Mor and Berlińska (2025) studied scheduling on parallel dedicated machines with a continuous non-renewable resource, focusing on makespan, total load, and total weighted completion time.

Furthermore, many studies have explored the incorporation of additional constraints, often addressed using heuristic and metaheuristic algorithms. For instance, setup times have been studied through exact methods by Fanjul-Peyro (2020), through three metaheuristics by Yepes-Borrero et al. (2020), while job preemption has been modeled via MILP in Burdett et al. (2021). Eligibility restrictions have been handled by Afzalirad and Shafipour (2018) using genetic algorithms, whereas specific scenarios, such as single-copy resource constraints (Abdeljaoued et al., 2020) and preventive maintenance scheduling (Lei and He, 2022), have been tackled with metaheuristic techniques. More complex cases involving multiple constraints, such as sequence-dependent setup times, varying release dates, and machine eligibility have been investigated through metaheuristic methods by Afzalirad and Rezaeian (2016), Yunusoglu and Topaloglu Yildiz (2022), and Shafiee et al. (2025).

2.2. Energy-efficient scheduling

Efforts to improve energy efficiency in scheduling have gained increasing importance, as minimizing energy consumption while maintaining high performance is essential in industrial applications. For

additional details on energy-efficient scheduling, we refer to the survey of Gahm et al. (2016). Energy demand-side management strategies, such as peak load reduction and load shifting, have been widely adopted. To achieve this, many countries have adopted time-of-use (TOU) electricity pricing, incentivizing manufacturers to shift their energy consumption from peak times to off-peak periods. Therefore, many contributions aim at minimizing the total energy cost of the schedule. For example, Shrouf et al. (2014), investigated energy-efficient scheduling for single-machine environments, proposing a genetic algorithm. Extending this line of research to unrelated parallel machines, Ding et al. (2016) utilized Dantzig–Wolfe decomposition and column generation heuristics, while Saberi-Aliabad et al. (2020) introduced a two-stage heuristic, and an MILP model with several dominance rules and valid inequalities to improve computational performance. In a related recent work, Mucciarini et al. (2024) proposed a time-indexed MILP formulation and a matheuristic approach for parallel machine scheduling with variable energy consumption functions, addressing realistic production settings where energy demand may fluctuate over time.

Although TOU electricity pricing has been widely studied in this context, the present work focuses on limiting peak energy consumption, which can help reduce costs and improve grid reliability. This energy constraint is central to the study by Módos et al. (2019), who explored scheduling problems on dedicated machines with energy consumption limits per metering interval, proposing exact methods, including a CP and an iterative MILP model. Later, Módos et al. (2021) examined the same problem, providing new complexity insights and developing heuristic algorithms. In a related line of research, Li and Liu (2024) addressed an identical parallel machine problem with a peak power constraint and a common deadline, although their objective focuses on maximizing job value rather than minimizing the makespan. Finally, many studies combine time- and energy-related objectives. For example, Fang and Lin (2013) propose a formal formulation, two heuristic algorithms, and a particle swarm optimization algorithm to minimize the sum of weighted tardiness and power consumption. However, most studies have focused on bi-objective optimization. For example, Li et al. (2016) explored bi-objective heuristics addressing tardiness and energy efficiency. Among more recent work, Chou et al. (2020) proposed a multi-objective scheduling algorithm to enhance production and energy efficiency, while respecting the energy consumption constraint, whereas (Gaggero et al., 2023) minimize the makespan and the total energy consumption by introducing a mathematical model, a heuristic, and a dynamic programming algorithm. More recently, Cheng et al. (2025) studied an energy-aware parallel machine scheduling problem in shared manufacturing environments, integrating makespan, energy, and sharing-cost objectives through MILP formulations and a tailored heuristic, thereby extending energy-efficient scheduling to collaborative production contexts.

While several studies have explored energy-aware scheduling in various settings, to the best of our knowledge, only (Min et al., 2025) have explicitly addressed the case of identical parallel machines with energy modeled as a single renewable resource. They demonstrated that, in this scenario, minimizing total energy consumption is equivalent to minimizing the makespan. They proposed a branch-and-bound (B&B) algorithm with three dominance properties and three lower bounds, as well as a simulated annealing algorithm to solve larger instances. Their study introduced new benchmark instances, which we also consider in our evaluation. Motivated by the growing interest in exact approaches for these problems, our work extends this line of research by exploring more efficient strategies to address the problem under a peak-load constraint on instantaneous energy usage.

2.3. Packing problems

To address $P|\text{res1..}|C_{\max}$, we draw inspiration from the closely related $P|\text{cont}|C_{\max}$. In this scheduling problem, each job is characterized not only by a processing time but also by the number of contiguous identical parallel machines it requires to be executed. The objective is to minimize the makespan. This problem arises in applications such as parallel processing on GPUs and is central to several packing problems. Specifically, $P|\text{cont}|C_{\max}$ can be viewed as a relaxation of the Strip Packing Problem (SPP), where a set of rectangular items must be packed without overlap into a strip of fixed width and infinite height, minimizing the total height used. A standard relaxation discretizes the strip into unit-width vertical columns and slices each item into vertical slices of width one. Each slice is then interpreted as an individual job to be scheduled on a machine for a duration equal to the height of the item. The contiguity constraint ensures that all slices of the same item are assigned to adjacent columns, thus linking SPP and $P|\text{cont}|C_{\max}$.

This connection has been thoroughly explored in the literature. We refer the interested reader to the recent survey by Akçay and Delorme (2025), which provides a well-structured and comprehensive literature review on parallel processor scheduling and bin packing problems with contiguity constraints, including $P|cont|C_{\max}$. Building on this connection, several works in the packing literature use $P|cont|C_{\max}$ as the master problem in decomposition methods for SPP. Typically, the x-coordinate of items is fixed by solving $P|cont|C_{\max}$, while a separate subproblem reconstructs the solution, verifying feasibility along the vertical axis. Several exact methods have been proposed to solve this master problem, including CP (Clautiaux et al., 2008), B&B algorithms (Côté et al., 2014), and arc-flow-based ILP formulations (Delorme et al., 2017). In other works, $P|cont|C_{\max}$ has been used to derive strong lower bounds for SPP, such as in Martello et al. (2003) through a B&B approach and in Boschetti and Montaletti (2010) by means of an ILP model. For a comprehensive review of exact solution methods for two-dimensional packing problems, we refer the reader to the survey by Iori et al. (2021). Among the approaches discussed in the literature, B&B algorithms have shown excellent performance, especially on small instances. For this reason, we extend these techniques to solve the $P|res1..|C_{\max}$.

3. Problem definition and mathematical models

Let J represent a set of n jobs to be scheduled on m identical parallel machines. Each job $j \in J$ is characterized by a processing time p_j and requires a constant amount r_j of a renewable resource consumed per unit of time during its execution. Accordingly, the resource consumption is instantaneous and remains constant over the entire processing time of the job. Jobs are assumed to be non-preemptive, meaning that once a job starts, it runs to completion without interruption. A schedule is considered feasible if each job is assigned to a machine, with each machine processing at most one job at any given time. Additionally, at any given time, the total resource consumption of all scheduled jobs must not exceed the maximum resource capacity limit $R_{\max}$. The objective is to create a feasible schedule that minimizes the makespan, defined as the completion time of the last scheduled job. These modeling assumptions are general and potentially apply to any renewable resource whose availability is limited at each time instant. In particular, when the resource represents electrical energy, r_j can be interpreted as the power demand of job j, and the capacity $R_{\max}$ as a maximum admissible power level imposed by the energy infrastructure or by contractual peak-load limits. As previously noted, this problem is commonly referred to as $P|res1..|C_{\max}$ in the scheduling literature. We consider the scheduling horizon T as an upper bound on the optimal makespan, computed as the sum of all processing times.

In the following sections, we propose two mathematical models that do not include explicit job–machine assignment variables because the machines are identical, and both processing times and resource consumptions are therefore machine-independent. This implicit formulation reduces the number of decision variables without loss of generality, as any feasible schedule can be equivalently represented by distributing jobs among the identical machines. This approach is common in time-indexed formulations for identical parallel machines (see Van den Akker et al., 2000).

3.1. Time-indexed MILP model

In scheduling, time-indexed (TI) models have been formally introduced in the seminal paper by Sousa and Wolsey (1992). They have been intensively used since then, either to directly solve a variety of problems, including resource-constrained project scheduling (see, e.g., Artigues, 2017) and satellite scheduling (e.g., Ferrari et al., 2025), or to serve as the basis for more elaborated solution techniques such as column generation, as shown in Van den Akker et al. (2000). Furthermore, they have been largely used in the cutting and packing literature, under the name position-indexed models (e.g., De Carvalho, 2002).

In the TI model, let x_{jt} be a binary variable that takes the value 1 if job j starts processing at the beginning of time period t, and 0 otherwise. Let $C_{\max}$ represent the makespan. Problem $P|res1..|C_{\max}$ can be modeled as follows:

$$\text{(TI)} \quad \min C_{\max} \tag{1}$$

$$\text{s.t.} \quad \sum_{t=0}^{T-p_j} x_{jt} = 1 \qquad j = 1,\dots,n \tag{2}$$

$$\sum_{j=1}^{n} \sum_{i=\max(0,t-p_j+1)}^{t} x_{ji} \le m \qquad t = 0,\dots,T-1 \tag{3}$$

$$\sum_{j=1}^{n} \sum_{i=\max(0,t-p_j+1)}^{t} r_j x_{ji} \le R_{\max} \qquad t = 0,\dots,T-1 \tag{4}$$

$$C_{\max} \ge \sum_{t=0}^{T-p_j} t x_{jt} + p_j \qquad j = 1,\dots,n \tag{5}$$

$$x_{jt} \in \{0,1\} \qquad j = 1,\dots,n, \quad t = 0,\dots,T-1 \tag{6}$$

The objective function (1) minimizes the makespan. Constraints (2) impose that each job is scheduled on exactly one machine, while Constraints (3) impose that the number of active jobs at any time does not exceed the number of available machines. Constraints (4) guarantee that the amount of renewable resources used for each time interval does not exceed the threshold ($R_{\max}$). Since energy is modeled as a renewable resource shared by all machines, these constraints enforce a system-level power capacity limit. Constraints (5) define the makespan as the completion time of the last scheduled job. Constraints (6) describe the domain of the variables. This formulation enforces non-preemptive processing by definition: the binary start variables x_{jt} select exactly one start time for each job, namely, $S_j = \sum_{t=0}^{T-p_j} t x_{jt}$. The job is then processed continuously over the interval $[S_j, S_j + p_j)$. During this interval machine and resource capacities are enforced, which prevents any interruption or splitting of jobs. Starting from the values of the variables x_{jt}, it is always possible to reconstruct a feasible schedule in which each job is uniquely assigned to a machine and each machine processes at most one job at a time. This assignment can be obtained by iteratively considering job in non-decreasing order of start time and assigning each job to the first machine available at its start time.

A formal proof of the correctness of model (1)–(6) is provided in Appendix.

3.2. Constraint-programming model

In the constraint-programming (CP) model, for each job j, we define a time interval variable x_j that represents its execution. This interval is associated with three integer variables: StartOf(x_j), denoting the job's start time, EndOf(x_j), representing its completion time, and LengthOf(x_j), specifying its duration. These variables are constrained by the relation StartOf(x_j) + LengthOf(x_j) = EndOf(x_j), which ensures coherence between the job's start, duration, and end times.

Consequently, each job is executed over a single, uninterrupted time interval, prohibiting preemption. Given this interval-based representation, the function $pulse(x_j, q_j)$ is used to model the time-dependent usage of shared capacities associated with job execution. Specifically, $pulse(x_j, q_j)$ adds a constant amount q_j at each time instant between StartOf(x_j) and EndOf(x_j), and is zero otherwise. Summing these terms over all jobs enforces instantaneous capacity limits on shared resources, such as machines or energy.

Problem $P|res1..|C_{\max}$ can then be modeled as:

$$\text{(CP)} \quad \min \max_{j \in J} \text{EndOf}(x_j) \tag{7}$$

$$\text{StartOf}(x_j) \ge 0 \qquad j = 1,\dots,n \tag{8}$$

$$\text{EndOf}(x_j) \le T \qquad j = 1,\dots,n \tag{9}$$

$$\text{LengthOf}(x_j) = p_j \qquad j = 1,\dots,n \tag{10}$$

$$\sum_{j=1}^{n} \text{pulse}(x_j, 1) \le m \tag{11}$$

$$\sum_{j=1}^{n} \text{pulse}(x_j, r_j) \le R_{\max} \tag{12}$$

The objective function (7) minimizes the makespan, that is, the completion time of the last job in the schedule. Constraints (8) impose that each job starts after the beginning of the time interval and constraints (9) ensure that all jobs end within the time horizon. Constraints (10) set the length of the interval variable to be the processing time of jobs. Constraints (11) impose that at most m jobs can be scheduled simultaneously, thus guaranteeing that each machine can process at most one job at a time. Constraints (12) impose the limit on the maximum number of renewable resources that can be used simultaneously, which in this work specifically accounts for energy availability. Similarly to the MILP formulation, a feasible schedule can be reconstructed by assigning each job j to the first machine that is available at StartOf(x_j) and keeping it on that machine until completion.

4. An exact algorithm for $P|res1..|C_{\max}$

In this section, we present an exact algorithm for $P|res1..|C_{\max}$. Algorithm 1 presents an informal pseudo-code that adapts to the characteristics of each instance. The algorithm follows different paths according to the characteristics of the instance at hand. For instances with two machines, we adopt the MILP model proposed by Fleszar and Hindi (2018) for the Unrelated Parallel Machine Problem (UPMR) with $m = 2$. This method explores all possible machine–job assignments and determines an optimal schedule using an MILP model originally developed for project scheduling with precedence constraints. Although the UPMR with $m = 2$ is $\mathcal{NP}$-hard, the method remains computationally efficient in this restricted setting due to the small number of machines. Adapting the algorithm to the identical parallel machine setting is straightforward and does not require modifying the model. For instances with more than two machines, we apply a binary search where the selection of the algorithm depends on the number of jobs. We first compute a lower bound L and an upper bound U (as described in Section 4.1) and use them to set a tentative solution value B. If the number of jobs does not exceed a predetermined threshold value μ, we solve the instance using a combinatorial B&B method (Section 4.2). For larger instances, the CP model described in Section 3.2 is used solely as a feasibility checker, with no active objective function, to verify whether a solution with makespan $C_{\max} \le B$ exists. If a feasible solution is found, the upper bound is updated to $C_{\max}$; otherwise the lower bound is increased to $B+1$. The algorithm stops when $L = U$ or the time limit is reached.

4.1. Bounds

Before the execution of the exact algorithm, a preprocessing phase computes a lower bound and a heuristic solution.

Algorithm 1 Algorithm for the solution of $P|res1..|C_{\max}$

```
input: n, m, p, r, R_max, t_lim, μ
if m = 2 then
    execute the MILP model by Fleszar and Hindi (2018) for UPMR with m = 2
else
    compute initial lower bound L and upper bound U
    while L < U do
        B := ⌊(L + U)/2⌋
        if n ≤ μ then
            execute combinatorial B&B with time horizon B
        else
            solve CP model with time horizon B
        end if
        if solution is feasible with makespan C_max then
            U := C_max
        else
            L := B + 1
        end if
    end while
end if
return U
```

Lower bounds. We initially calculate three well-known lower bounds from the literature. The first bound, $L_0 = \max_{j\in J} p_j$, ensures that the makespan is at least as large as the longest job processing time. The second bound, $L_1 = \lceil \sum_{j\in J} p_j/m \rceil$, continuously distributes the total job processing time across all machines. Finally, $L_2 = \lceil \sum_{j\in J} p_j r_j / R_{\max} \rceil$ computes the minimum required resource consumption over the time horizon, divided by the maximum resource availability per time unit.

We then compute a fourth bound, L_3, using a pattern-based formulation within a column generation framework. A pattern is defined as a feasible combination of jobs, each of unitary length, that can be processed concurrently without exceeding machine and resource capacities. We describe the pattern q as a column vector $(a_{1q}, \dots, a_{jq}, \dots, a_{nq})^T$, where a_{jq} takes the value 1 if job j is assigned to pattern q, and 0 otherwise. A pattern q is valid if $\sum_{j\in J} a_{jq} \le m$ and $\sum_{j\in J} a_{jq} r_j \le R_{\max}$. Let Q denote the set of all valid patterns. We introduce the decision variable z_q, which indicates the number of times pattern q is used in the solution. The resulting pattern-based model is then:

$$(L_3)\ \min \sum_{q\in Q} z_q \tag{13}$$

$$\text{s.t.} \sum_{q\in Q} a_{jq} z_q \ge p_j \qquad j \in J \tag{14}$$

$$z_q \ge 0, \text{integer} \qquad q \in Q \tag{15}$$

The objective function (13) minimizes the number of selected patterns. Constraints (14) ensure that the chosen patterns cover the entire processing time of each job. Constraints (15) define the domains of the decision variables. Solving L_3 provides a valid lower bound on the minimum number of such patterns, and consequently on the makespan of the original scheduling instance. However, since the problem is NP-hard, we solve its continuous relaxation using column generation. We begin by creating n patterns, each one containing exactly one job. Then, at each iteration, new patterns are introduced by solving the Cardinality-Constrained Knapsack Problem (CCKP), a well-studied variant of the knapsack problem (Cacchiani et al., 2022a,b). The CCKP is defined as follows: given n items, each with an associated profit and weight, the objective is to select a subset of items that maximizes the total profit without exceeding both a total weight limit and a maximum number of selected items. In our context, each item corresponds to a job j in the original scheduling instance. The profit of item j is given by the dual variable π_j associated with constraints (14), the weight corresponds to the resource consumption r_j, the capacity of the knapsack corresponds to the resource limit $R_{\max}$, and the maximum number of the selected items equals the number of machines m. Let variables v_j take the value 1 if job j is included in the pattern, and 0 otherwise. The CCKP can then be modeled as:

$$(\text{CCKP})\ 1 - \max \sum_{j\in J} \pi_j v_j \tag{16}$$

$$\text{s.t.} \sum_{j\in J} r_j v_j \le R_{\max} \tag{17}$$

$$\sum_{j\in J} v_j \le m \tag{18}$$

$$v_j \in \{0,1\} \qquad j \in J \tag{19}$$

The objective function (16) identifies a pattern of minimum reduced cost. Constraint (17) ensures that the resource consumption does not exceed the capacity. Constraint (18) enforces the machine capacity limits and constraints (19) define the binary nature of the variables. After solving (16)–(19), a new pattern v_j^* is generated and incorporated into L_3 whenever the corresponding reduced cost is negative.

Similarly to other column generation approaches, as described in Lübbecke and Desrosiers (2005), we introduce multiple columns at once, extracting K promising patterns per iteration, where K is the number of solutions obtained by solving the CCKP. To achieve this, we solve the CCKP using a dedicated DP approach, where each state is defined by the number of selected items and the total resource consumption. We maintain a priority queue that stores the best feasible solutions, ordered by decreasing reduced cost. At each step of the DP, we add items to feasible states, retaining only the best Γ solutions per unique (cardinality, resource) pair to limit state explosion. The column generation process terminates when no new columns with negative reduced cost can be generated. The worst-case pseudo-polynomial complexity of the DP is $O(nmR_{\max})$.

After computing the lower bounds above, the initial lower bound L is set to $\max\{L_0, L_1, L_2, L_3\}$.

Upper bound. Problem $P|res1..|C_{\max}$ can be visualized on a two-dimensional plane, with time on the x-axis and resource consumption on the y-axis, as shown in Fig. 1. The time horizon is divided into equal-width slices, referred to as *columns*, each with a width of 1 and height $R_{\max}$. Each job j is represented by p_j adjacent rectangles, each with a width of 1 and a height of r_j, scheduled contiguously across the time horizon. A scheduling solution is defined by the start time s_j of each job j, occupying the time intervals from s_j to $s_j + p_j - 1$. To account for the number of parallel machines, each column can accommodate up to m jobs. Furthermore, the cumulative height of jobs scheduled within a column must not exceed $R_{\max}$. As shown in the figure, the resulting schedule exhibits the classical staircase structure described in Martello et al. (2003).

To produce a heuristic solution with limited computational effort, we use a three-step heuristic algorithm inspired by Leung et al. (2011). In the first step, jobs are initially randomly ordered, and the objective function is evaluated using a constructive heuristic that incrementally schedules jobs by selecting the best-fitting one at each iteration based on a scoring system, with ties broken according to the initial sequence. Additional details on the constructive algorithm are provided later in this section. Since the initial sequence strongly affects performance, in the second step a local search is used to swap each pair of jobs and accept improvements. Finally, in the third step, the best solution obtained from the local search is further improved using a simulated annealing (see, e.g., Nikolaev and Jacobson, 2010). The algorithm starts by setting an initial temperature $T_0 = 0.5$ and defining the current solution as the one returned by the local search. It then performs 100 iterations for each temperature level. At each iteration, a new candidate solution is generated by swapping two randomly-selected jobs in the current sequence. Each new sequence is evaluated using the same best-fit algorithm described later in this section. If the new solution

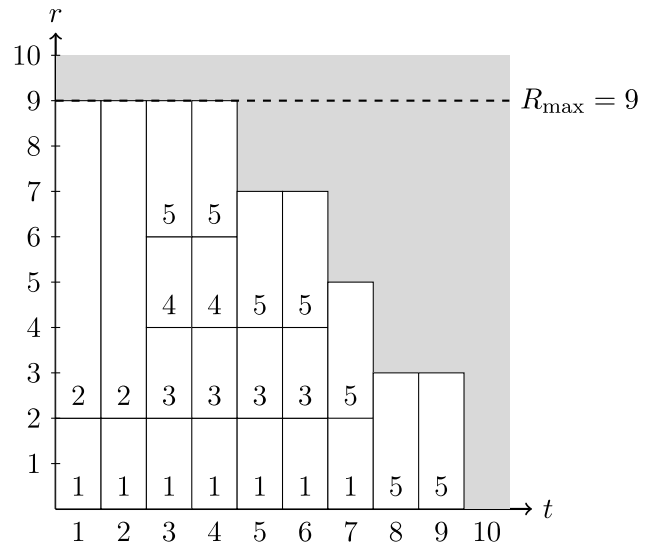

Fig. 1. Two-dimensional representation of a schedule.

improves the makespan, it replaces the current one. Otherwise, it is accepted with probability $e^{-\Delta_k/T_k}$, where Δ_k is the increase in makespan and T_k the current temperature. After 100 iterations, the temperature is updated according to $T_{k+1} = \alpha T_k$, with $\alpha = 0.93$. The process continues for 100 iterations or until the lower bound is reached, with the best sequence returned as the final solution.

At a given iteration of the constructive heuristic, a partial scheduling can be defined by its staircase configuration. Specifically, let $\bar{r}_t$ be the total resource consumed by jobs in each column t. The set of horizontal segments of height $\bar{r}_t$ forms the staircase, with the so-called *open stairs* defining segments with $\bar{r}_t < R_{\max}$. Let c be the leftmost column of the earliest open stair, and w the length of this stair. For example, in Fig. 1 we have $\bar{r}_t = R_{\max}$ for $t = \{1, \dots, 4\}$, $\bar{r}_t = 7$ for $t = \{5,6\}$, $\bar{r}_t = 5$ for $t = \{7\}$, and $\bar{r}_t = 3$ for $t = \{8,9\}$. The open stairs are thus the last three mentioned segments. Hence, the earliest open stair starts in column $c = 5$, and has a length of $w = 2$. At each iteration, the algorithm selects, based on a scoring system, the most suitable job to be scheduled in c. The score v_j assigned to job j is:

$$v_j = \begin{cases} 3, & \text{if } p_j = w \text{ and } r_j = R_{\max} - \bar{r}_c \\ 2, & \text{if only } r_j = R_{\max} - \bar{r}_c \\ 1, & \text{if only } p_j = w \\ 0, & \text{otherwise} \end{cases} \qquad j \in J$$

The job j with the highest score is selected and scheduled. Ties are broken by selecting the job with the smallest index in the sequence provided by the metaheuristic of stage 1. After scheduling a job, the resource consumption $\bar{r}_t$ for each interval t in which the job is scheduled (i.e., $t \in \{s_j, \dots, s_j + p_j - 1\}$) is updated to $\bar{r}_t + r_j$. Then, if none of the remaining unscheduled jobs can fit within the current open stair, the stair is closed, preventing further scheduling attempts. The algorithm then identifies the next open stair and reiterates the process until all jobs are scheduled.

4.2. Combinatorial branch-and-bound

This section presents the B&B approach implemented at Line 9 of Algorithm 1 to solve the recognition version of $P|res1...|C_{\max}$ in which the makespan is fixed to B. The algorithm adapts the method proposed in Côté et al. (2014) for SPP, but includes new strategies and new fathoming criteria.

Preprocessing. At the root node, the B&B applies a preprocessing lifting technique to reduce computational time. For each job j, we solve a subset sum problem where p_j represents both the weight and the profit of the items, and the bin capacity is set to B. The problem is solved by means of DP, following the algorithm proposed by Martello et al. (1999). The solution estimates the maximum achievable total processing time p_j^{tot} by combining different jobs, under the assumption that j is scheduled. If no combination of jobs exactly reaches the time horizon B, then p_j is updated to $p_j + B - p_j^{tot}$.

Bounding. At each node of the B&B tree, the algorithm identifies the leftmost column s in the first open stair and applies four fathoming criteria. First, the node is pruned if any unscheduled job satisfies $t + p_j > B$ or if the total area of unscheduled jobs exceeds the remaining available area above the staircase. Then, we incorporate the timetabling techniques introduced by Baptiste et al. (2001) for CP. These techniques generate a profile of the minimum use of resources over time by identifying mandatory scheduling intervals, allowing us to prune nodes that violate these constraints. Furthermore, *DP cuts* from Kenmochi et al. (2009) are used. These cuts, derived from subset sum problems, estimate the minimum waste in any row or column of the partially scheduled horizon. Based on the waste estimation, the stair is lifted, and the resulting area of the modified stair is used to refine the area-based bound. Due to their high computational cost, DP cuts are applied only if the first three criteria fail to prune the node.

Branching. Jobs are ordered by non-increasing resource consumption, with ties resolved first by non-increasing processing time and then by increasing index. The B&B algorithm then attempts to schedule each unscheduled job in the left-most column s of the earliest open stair, creating a descendant node for each job. The node for the unscheduled job j is not created if:

1. the scheduling of j in s violates resource constraint, that is, if $r_j > R_{\max} - \bar{r}_s$;
2. the number of scheduled jobs covering s is equal to m;
3. there is an unscheduled job k with $p_k = p_j$, $r_k = r_j$ and $k < j$;
4. a job k with a higher index ($k > j$) is already scheduled starting from time s;
5. a job k with a higher index ($k > j$) completes at time s and has the same resource requirement ($r_k = r_j$);
6. there is an empty space contiguous with the current column s whose height can accommodate j, meaning that there are at least r_j unused units of resources in column $s - 1$.

The first two criteria ensure feasibility, the subsequent three prevent the exploration of symmetric solutions where the two jobs j and k are swapped, and the last one discards inefficient solutions.

Finally, the B&B algorithm generates a node where no job is scheduled in the stair, setting the resource usage to $R_{\max}$ to mark its columns as closed if no unscheduled job fits entirely within the open space before $s + w$, where w is the length of the stair being closed.

The B&B terminates whenever it finds a feasible solution or proves that no feasible solution exists. Otherwise, it continues until the time limit is reached.

5. Computational results

In this section, we present a computational evaluation of the proposed algorithms and compare them with methods from the existing literature. All algorithms have been coded in C++ and run on an Intel(R) Xeon(R) Gold 6252N CPU at 2.30 GHz and 16 GB RAM under Microsoft Windows 10 Pro. The MILP was solved using Gurobi Optimizer version 11.0.2, while the CPs were solved using IBM ILOG CPLEX Optimization Studio version 22.1.1 and OR-Tools CP-SAT solver version 9.9.3963. All algorithms have been tested using a single core by setting the number of threads to one in the solvers configuration

(Threads = 1). Based on the preliminary test results, we set the parameters to $\mu = 16$ and $\Gamma = 25$, where μ is the threshold used to select the solution method (B&B or CP), and Γ limits the number of retained states in the DP procedure used to compute the lower bound. To evaluate the proposed approach, we conducted computational experiments on benchmark instances adapted from the literature. These instances cover different scenarios and were modified to fit the identical parallel machine scheduling problem. All instances and the source code are publicly available at https://github.com/regor-unimore/Energy-Constrained-Parallel-Machine-Scheduling.

5.1. Computational results on the first set of instances

Our first experimental benchmark consists of the instances originally proposed by Fanjul-Peyro et al. (2017). Specifically, the small-sized instances contain 8, 12, and 16 jobs, while the medium-sized instances consider 20, 25, and 30 jobs. Each instance is further characterized by the number of machines available, respectively 2, 4, and 6. Processing times and resource consumptions are generated from five and two different distributions, respectively. Each combination of these parameters is replicated five times, resulting in 450 small and 450 medium instances. The original instances can be found at http://soa.iti.es/problem-instances. It should be noted that these instances were originally designed for the unrelated RCPMSP with additional resources. Therefore, we adapted each instance to fit our problem by considering only the processing time and resource consumption associated with the first machine. We compare our results with the state-of-the-art method proposed by Fleszar and Hindi (2018), running their algorithm on our adapted instances. To ensure a fair comparison, we used the same updated solver versions as in our study and imposed a time limit of 120 s, matching the total time reported for their multistage approach.

Our first set of experiments, summarized in Table 1, aims to identify the most effective CP solver by comparing the performance of OR-Tools CP-SAT solver and IBM ILOG CPLEX Optimization Studio. The table presents results obtained under two different configurations using CP solvers: (i) the CP model employed in a pure minimization setting across all 900 instances, and (ii) the CP model integrated within the binary search scheme of Algorithm 1, evaluated on a subset of 300 instances. This reduced subset excludes the instances in which Algorithm 1 uses the B&B ($n \leq 16$) or the MILP ($m = 2$) instead of the CP. The last two lines provide total or average values for all columns, referring to the entire set of 900 instances (a/t) or to the subset of 300 instances solved by the CP in our algorithm (a/t^*).

For clarity, we adopt the following nomenclature: 'CP-OR' refers to model (7)–(12) solved using the OR-Tools CP-SAT solver, while 'CP-IBM' denotes the same model solved using IBM ILOG CPLEX Optimization Studio. 'BB-OR' and 'BB-IBM' refer to our approach, described in Section 4, when combined with the CP-OR and CP-IBM configurations, respectively.

The first columns of the table describe the main features of the instances: the number of jobs (n) and the number of machines (m) considered for each scenario. For every (n, m) combination, 50 instances are provided. The following columns report, for each algorithm, the number of instances solved to proven optimality within the time limit (Opt), the average CPU time in seconds ($T(s)$), which also includes the instances terminated due to the time limit, and the average percentage gap (Gap), calculated as $\frac{UB-LB}{UB} \cdot 100$. We omitted the number of instances where at least one feasible solution was found, as all approaches identify at least one feasible solution.

The table clearly shows that CP-IBM consistently outperforms CP-OR, solving a higher number of instances to optimality (545 compared to 385) and requiring less average computation time (50.44 s against 71.20 s). The performance gap becomes more evident as the problem size increases, with CP-OR struggling to find optimal solutions within the time limit. The comparison between BB-IBM and BB-OR confirms this trend. Due to these significant differences, we select IBM ILOG CPLEX Optimization Studio for comparative analysis in the remainder of this section.

Our next set of experiments, reported in Table 2, compares the performance of the discussed approaches.

In addition to the previously defined methods, we refer to model (1)–(6) as 'MILP' and to the approach proposed by Fleszar and Hindi (2018) as 'FH'.

As shown in Table 2, MILP achieves optimality for most small instances with moderate computational times. However, as the number of jobs increases, the proportion of instances solved to optimality drops considerably, particularly in scenarios with fewer machines. In 30 instances, MILP was able to return only a heuristic solution without providing a lower bound. For these cases, the optimality gap is set to 100%. CP-IBM performs notably better, solving 545 out of the 900 instances. FH improves the results further, demonstrating high efficiency in identifying near-optimal solutions. It reaches optimality in most instances while maintaining low computational times. Moreover, even as the number of unsolved instances increases, the average gap remains relatively small, with a mean of 1.11% and a maximum of 20.27%. Notably, only 20 out of 900 instances exceed a gap of 10%. BB-IBM outperforms the other approaches, optimally solving 718 out of 900 instances. Specifically, it performs similarly to FH on medium instances, but consistently finds better solutions as n decreases. Although its effectiveness slightly declines for larger instances, it remains highly competitive, achieving a significantly lower optimality gap than FH, with an average of 0.35%, a maximum of 11.24%, and only 1 instance exceeding 10%. Although no single approach is superior in every instance, our method consistently achieves optimal solutions much more often than the others. Specifically, CP-IBM and FH are the only algorithms that solve 4 and 7 instances to optimality, respectively, while BB-IBM is the only one to find the optimal solution in 40 instances, outperforming the alternatives. To evaluate the effect of the preprocessing step described in Section 4.2, we tested the B&B algorithm without it. The number of instances solved to proven optimality decreased by one out of 360, while the average computation time increased by 0.61 s.

In addition to the experiments reported above, we solved the MILP model (1)–(6) and the CP model (7)–(12) directly with the respective solvers, but initialized them with the same LB and UB values used in our algorithm. Bound initialization substantially improves MILP performance (optimal solutions from 295 to 348, average gap from 30.78% to 3.27%, and average time from 87.03 s to 78.30 s), whereas it does not benefit CP-IBM (optimal solutions from 545 to 534, average time from 50.44 s to 51.90 s, and average gap from 1.93% to 1.78%). This behavior confirms the comparative conclusions discussed above.

We can further investigate how the number of jobs and machines affects the performance of the algorithms, as shown in Fig. 2. In detail, the figure reports the impact of the number of jobs on the performance of the algorithms. As expected, for all algorithms the number of instances solved to proven optimality decreases as the number of jobs increases. MILP, for example, experiences a significant drop in performance, solving 128 instances with 8 jobs but only 7 with 30 jobs. However, the remaining approaches show a less drastic decrease compared to MILP, with BB-IBM maintaining the highest performance across all job sizes.

In addition, the figure provides an analysis of the impact of the number of machines on the performance of the algorithms. In this case, most algorithms benefit from an increasing number of machines. MILP, for instance, solves only 47 instances with 2 machines, but this number increases to 150 when 6 machines are available. Similarly, CP-IBM improves from 160 to 203 solved instances as the number of machines grows. For the case with two machines, both FH and BB-IBM solve all 300 instances. This identical performance is expected, as BB-IBM integrates FH in this case. However, when the number of machines increases to 4 or 6, FH solves 138 instances, while BB-IBM maintains significantly stronger performance, solving 209 instances in both cases.

Table 1
Results for the first set of instances: comparison between the CP solvers.

n	m	CP-OR			CP-IBM			BB-OR			BB-IBM		
		Opt	T (s)	Gap	Opt	T (s)	Gap	Opt	T (s)	Gap	Opt	T (s)	Gap
8	2	50	0.9	0.00	50	0.2	0.00						
	4	50	0.2	0.00	50	0.1	0.00						
	6	50	0.3	0.00	50	0.0	0.00						
12	2	32	50.8	11.42	49	8.3	0.01						
	4	44	23.9	7.83	50	0.7	0.00						
	6	49	5.6	1.43	50	0.2	0.00						
16	2	17	79.6	11.69	26	64.7	4.71						
	4	11	102.3	51.97	35	45.6	0.66						
	6	30	57.8	22.29	49	14.3	0.01						
20	2	20	72.1	10.09	14	90.1	7.87						
	4	4	112.7	64.30	21	72.7	0.96	23	71.2	1.15	30	60.4	0.86
	6	4	112.3	52.84	27	60.7	1.13	28	60.3	1.36	34	50.9	1.00
25	2	8	100.9	11.08	12	91.3	8.47						
	4	0	120.0	78.16	9	100.6	1.27	7	104.1	1.37	10	99.2	1.20
	6	0	120.0	64.08	15	87.7	1.28	14	89.8	1.52	18	85.6	1.23
30	2	16	82.2	10.29	9	98.5	6.62						
	4	0	120.1	79.33	17	79.4	0.77	17	79.7	0.88	21	72.0	0.76
	6	0	120.1	71.55	12	93.0	0.93	10	101.8	1.32	8	101.2	1.24
a/t		**385**	**71.2**	**30.46**	**545**	**50.4**	**1.93**						
a/t*		**8**	**117.5**	**63.38**	**101**	**82.4**	**1.06**	**99**	**84.6**	**1.17**	**121**	**78.2**	**1.05**

Table 2
Results for the first set of instances: impact of the number of jobs and machines.

n	m	MILP			CP-IBM			FH			BB-IBM		
		Opt	T (s)	Gap	Opt	T (s)	Gap	Opt	T (s)	Gap	Opt	T (s)	Gap
8	2	29	55.0	4.84	50	0.2	0.00	50	0.0	0.00	50	0.0	0.00
	4	49	11.6	0.11	50	0.1	0.00	49	2.1	0.38	50	0.0	0.00
	6	50	4.2	0.00	50	0.0	0.00	48	4.8	0.52	50	0.0	0.00
12	2	9	107.0	23.09	49	8.3	0.01	50	0.0	0.00	50	0.0	0.00
	4	25	73.4	3.22	50	0.7	0.00	34	24.4	2.84	50	0.1	0.00
	6	50	15.3	0.00	50	0.2	0.00	33	21.9	3.24	50	0.0	0.00
16	2	6	112.8	40.09	26	64.7	4.71	50	0.0	0.00	50	0.0	0.00
	4	11	101.3	12.91	35	45.6	0.66	17	41.7	2.18	48	10.3	0.01
	6	29	66.1	3.32	49	14.3	0.01	16	43.0	2.96	49	9.2	0.02
20	2	1	118.1	56.61	14	90.1	7.87	50	0.0	0.00	50	0.0	0.00
	4	8	107.8	27.93	21	72.7	0.96	12	51.1	1.62	30	60.4	0.86
	6	13	94.3	9.74	27	60.7	1.13	14	44.5	1.45	34	50.9	1.00
25	2	1	120.0	78.77	12	91.3	8.47	50	0.0	0.00	50	0.0	0.00
	4	1	118.3	54.29	9	100.6	1.27	9	64.1	1.59	10	99.2	1.20
	6	6	110.4	34.37	15	87.7	1.28	17	46.2	1.10	18	85.6	1.23
30	2	1	122.5	88.54	9	98.5	6.62	50	0.0	0.00	50	0.0	0.00
	4	4	111.6	57.99	17	79.4	0.77	16	55.7	1.01	21	72.0	0.76
	6	2	117.2	58.16	12	93.0	0.93	8	67.5	1.36	8	101.2	1.24
a/t		**295**	**87.0**	**30.78**	**545**	**50.4**	**1.93**	**573**	**25.9**	**1.13**	**718**	**27.2**	**0.35**

5.2. Computational results on the second set of instances

The second set of benchmark instances, introduced by Min et al. (2025) for exact solution approaches, includes a wide range of configurations. In these 720 instances, the number of jobs n takes value in $\{5, 10, 15, 20, 25, 30\}$, while the number of machines m is either 2 or 3. Processing times and resource requirements are sampled from a uniform distribution. Two resource limit settings are considered, referred to as *low* and *high*, which correspond to the 5th and 7th quintiles, respectively, after partitioning the total required energy into nine equal intervals. Moreover, Min et al. (2025) introduces a consumption rate v for idle machines, fixed at 2 across all instances. To align these values with our formulation (which does not explicitly account for idle machine consumption), we adjusted the resource parameters accordingly. Specifically, the instances proposed by Min et al. (2025) (with the resource constraint given in (20)) have been modified to match our problem's constraint (4) by subtracting v from each job's requirement and mv from the overall resource limit.

$$\sum_{j=1}^{n}((r_j - v)\sum_{i=\max(0,t-p_j+1)}^{t} x_{ji}) \leq R_{\max} - mv, \quad t = 0, \ldots, T-1 \tag{20}$$

Table 3 summarizes the computational results obtained for these instances.

The table is structured similarly to Table 2, with the addition of 'MLK', which corresponds to the B&B approach proposed in Min et al. (2025). For each combination of n jobs and m machines, the reported metrics, based on the 60 studied instances, include the number of optimal solutions found, the average computational time, and the average percentage gap. Note that the average percentage gap for MLK is not provided, as it was omitted in the original study. It is also important to note that the time limit was set to 120 s for MILP, CP-IBM, FH, and BB-IBM, whereas the MLK results reported in Min et al.

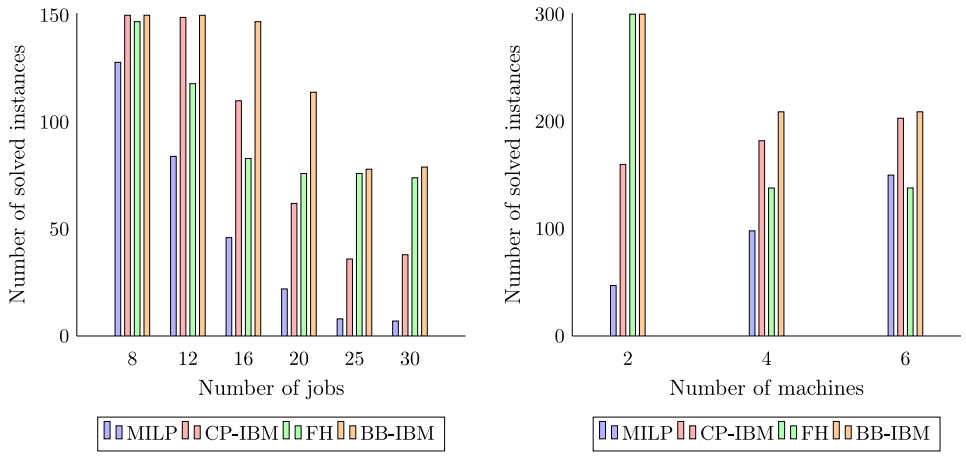

Fig. 2. Impact of job and machine counts on instances solved to proven optimality.

Table 3
Results for the second set of instances.

n	m	MILP			CP-IBM			FH			BB-IBM			MLK	
		Opt	T (s)	Gap	Opt	T (s)	Gap	Opt	T (s)	Gap	Opt	T (s)	Gap	Opt	T (s)
5	2	60	1.6	0.00	60	0.0	0.00	60	0.0	0.00	60	0.0	0.00	60	0.0
	3	60	1.1	0.00	60	0.0	0.00	60	0.1	0.00	60	0.0	0.00	60	0.0
10	2	47	38.6	1.64	60	0.3	0.00	60	0.0	0.00	60	0.0	0.00	60	0.7
	3	47	38.0	1.02	60	0.5	0.00	56	6.2	0.68	60	0.0	0.00	60	2.8
15	2	34	89.8	11.20	59	2.9	0.05	60	0.0	0.00	60	0.0	0.00	58	130.8
	3	39	66.6	6.36	57	8.3	0.12	38	25.5	1.27	60	32.8	0.23	54	684.0
20	2	6	119.1	33.91	58	4.0	0.19	60	0.0	0.00	60	0.0	0.00	56	319.4
	3	22	107.5	19.72	52	16.1	0.32	40	23.9	0.58	52	33.6	0.53	41	1409.5
25	2	0	120.2	81.23	59	2.1	0.10	60	0.0	0.00	60	0.0	0.00	50	889.2
	3	3	119.6	52.02	55	10.9	0.14	44	22.1	0.45	54	29.8	0.45	28	2269.8
30	2	0	120.3	95.54	59	2.0	0.05	60	0.0	0.00	60	0.0	0.00	38	1693.2
	3	0	120.3	83.71	55	10.1	0.09	48	16.8	0.37	54	27.5	0.47	20	2724.3
a/t		**318**	**78.6**	**32.20**	**694**	**4.8**	**0.09**	**688**	**3.6**	**0.11**	**700**	**3.5**	**0.06**	**585**	**843.6**

(2025) were obtained using a time limit of 3600 s. For comparisons, MLK results were obtained using a personal computer with Intel(R) Core(TM) i7-9700 CPU @ 3.00 GHz and 32 GB RAM under Microsoft Windows 10.

As shown in Table 3, MILP performs considerably worse than the other algorithms, solving less than half of the instances handled by the alternatives. In contrast, CP-IBM, FH, and BB-IBM successfully solve nearly all instances, specifically, 694, 688, and 700 out of 720, respectively. Notably, BB-IBM achieves both a very low percentage gap (0.06%) and reduced computational times (3.50 s on average). Although MLK benefits from a much higher time limit, its performance deteriorates as the number of jobs increases; for instance, with 30 jobs, it manages to solve only about half of the cases. However, the approach remains effective for instances with a small number of jobs, highlighting the suitability of B&B methods for such cases. Similarly to the first benchmark set, we solved the MILP and CP-IBM formulations using the same bounds as in our algorithm: bound initialization markedly improves MILP performance (optimal solutions from 318 to 472, average gap from 32.20% to 0.34%, average time from 78.56 s to 57.13 s), whereas it does not benefit CP-IBM (optimal solutions from 694 to 693, average gap from 0.09% to 0.12%, average time from 4.76 s to 5.04 s).

Since the instances are generated to consider two different energy levels (e), we further investigate how these levels impact solution performance. Table 4 compares the performance of the different approaches by classifying the instances according to the resource level and number of machines. Let h indicate a high energy level and l a low energy level. For each energy–machine combination, 180 instances are provided.

As observed, all instances were solved to optimality by BB-IBM, except for those with a low resource level and three machines. This outcome indicates that the combination of limited resources and increased machine complexity significantly increases the difficulty of obtaining optimal solutions within the established time limits. A similar trend is observed across all other algorithms, whose performance also deteriorates with lower resources and a higher number of machines. An exception is the MILP approach, which performs comparatively better in the case of three machines and high resource availability. For this benchmark, although disabling the preprocessing step described in Section 4.2 did not change the number of instances solved to proven optimality (out of 180), it increased the average computation time by 0.30 s.

We further explore the impact of the number of jobs and machines on performance, as illustrated in Fig. 3. In particular, the figure shows that, as in the first set of instances, performance generally worsens as the number of jobs increases. MILP is especially affected, solving only 12 instances with 30 jobs compared to 240 with 5 jobs. In contrast, CP-IBM, FH, and BB-IBM follow a slightly different pattern: their performance first declines, reaching a low point around 15 jobs for FH and around 20 jobs for CP-IBM and BB-IBM, before slightly improving or remaining stable as the number of jobs grows further.

In addition, the figure analyzes the effect of the number of machines on the number of instances solved to proven optimality. In general, the performance of most algorithms slightly decreases when moving from 2

Table 4
Impact of the level of resources and the number of machines.

e	*m*	MILP			CP-IBM			FH			BB-IBM			MLK	
		Opt	T (s)	Gap	Opt	T (s)	Gap	Opt	T (s)	Gap	Opt	T (s)	Gap	Opt	T (s)
h	2	86	34.1	76.40	180	0.1	0.00	180	0.0	0.00	180	0.0	0.00	168	363.7
	3	112	21.1	63.91	180	0.3	0.00	179	0.4	0.00	180	0.2	0.00	154	719.7
l	2	61	40.4	86.82	175	3.7	0.13	180	0.0	0.00	180	0.0	0.00	154	647.4
	3	59	33.2	87.10	159	15.0	0.22	149	13.8	0.45	160	13.8	0.23	109	1643.7
a/t		**318**	**32.2**	**78.56**	**694**	**4.8**	**0.09**	**688**	**3.6**	**0.11**	**700**	**3.5**	**0.06**	**585**	**843.6**

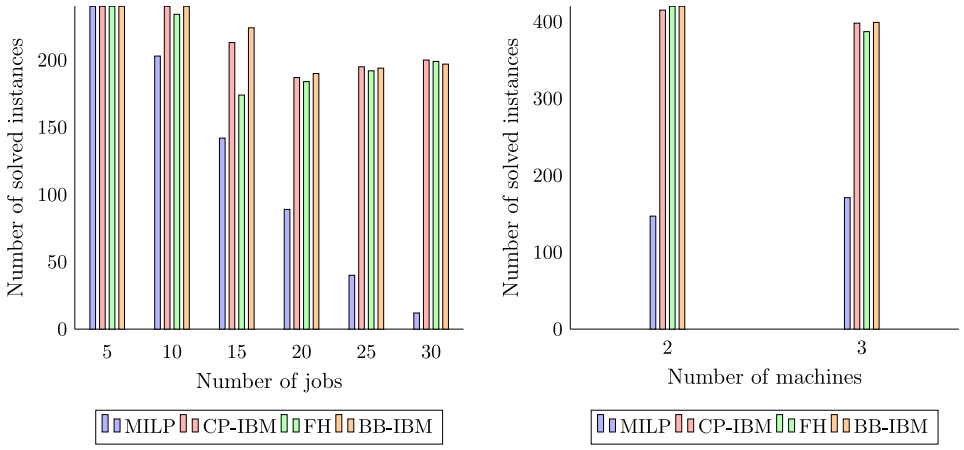

Fig. 3. Impact of job and machine counts on instances solved to proven optimality.

to 3 machines. FH and BB-IBM solve all 420 instances when 2 machines are available, but this number drops to 387 and 399, respectively, with 3 machines. A similar trend is observed for CP-IBM, which solves 415 instances with 2 machines and 398 with 3 machines. MILP, instead, exhibits moderate improvements, increasing from 147 to 171 optimally solved instances as the number of machines grows. Overall, BB-IBM consistently achieves the best performance in both configurations.

5.3. Convergence analysis

The efficiency of the binary search procedure employed in Algorithm 1 was further assessed by analyzing its convergence behavior. Each iteration of the binary search corresponds to a feasibility check of a candidate makespan, performed using either the CP or the B&B depending on the instance size, and thus measures how quickly the algorithm converges between the initial bounds. Instances with $m = 2$ were excluded from this analysis, as they are solved directly by the MILP model of Fleszar and Hindi (2018). For all other configurations, we computed the average number of iterations for instances solved to proven optimality (*Opt*) and for those terminated without proof of optimality (*Not Opt*). We also report the average initial ($U_0 - L_0$) and final ($U - L$) gaps between the upper and lower bounds, computed only for instances not solved to proven optimality, to quantify the contraction of the search interval.

Table 5 reports the convergence statistics for both benchmark sets of instances. For the first one, the algorithm converged within two to three iterations in most cases. The average number of iterations ranged from 0.75 to 3.34 for solved instances and remained below four for those not proven optimal, showing no clear dependence on the problem size. The initial bounds were already tight, with average values below 15 units, and the final gaps were further reduced at termination. As indicated by the reported values, the difference between upper and lower bounds was always smaller than the initial interval, confirming that the algorithm effectively reduces the optimality gap during the binary search process. For the second set, convergence was even faster,

Table 5
Average number of iterations for optimal and non-optimal instances.

n	*m*	Average iterations		Average $U_0 - L_0$	Average $U - L$
		Opt	Not Opt	Not Opt	Not Opt
8	4	3.16	–	–	–
8	6	2.72	–	–	–
12	4	3.12	–	–	–
12	6	3.24	–	–	–
16	4	2.60	1.57	1.50	1.00
16	6	3.35	3.00	8.00	1.00
20	4	1.97	1.60	14.80	11.00
20	6	2.53	1.81	13.75	9.13
25	4	1.30	1.63	12.70	9.78
25	6	2.00	2.22	12.19	7.97
30	4	0.95	1.45	11.03	9.59
30	6	0.75	1.95	11.10	8.02
5	3	2.35	–	–	–
10	3	2.07	–	–	–
15	3	0.73	–	–	–
20	3	0.02	1.38	11.88	10.88
25	3	0.00	1.33	15.50	12.17
30	3	0.06	1.00	17.50	17.50

with the number of iterations ranging from 0.00 to 2.35 for optimal cases. In several configurations, particularly those with 20 jobs or more, the algorithm completed without additional iterations after the initial bound check, indicating that the lower and upper bounds computed in the preprocessing phase were already nearly coincident. In the few unsolved cases, feasibility subproblems limited the number of iterations, although the distance between bounds remained small.

Additionally, Fig. 4 illustrates the overall distribution of the number of iterations for the two benchmark sets. In the first set, most instances (around 75%) converged within three iterations, and only a few required more than five. This pattern was even more evident in the second set, in which more than half of the instances (224 out of 350) were solved without any iteration, and nearly all remaining instances

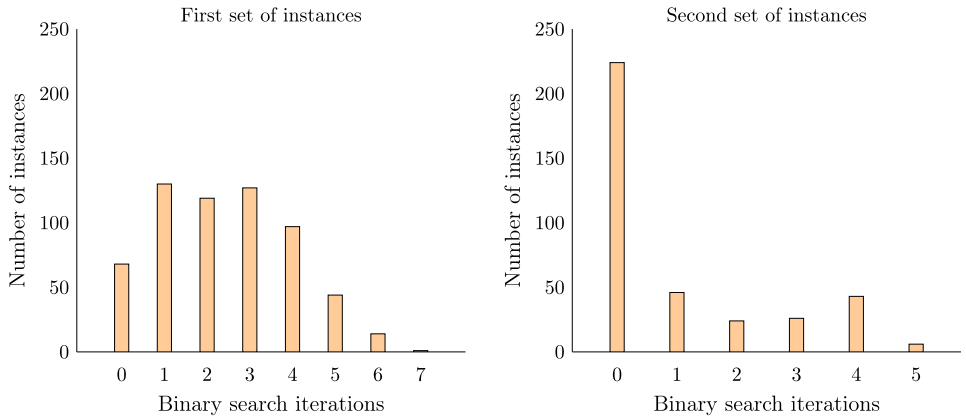

Fig. 4. Distribution of the number of iterations across the two benchmark sets.

converged within four. Overall, the binary search terminated after very few iterations, confirming the tightness of the initial bounds.

6. Conclusions

In this paper, we investigate the problem of scheduling jobs on identical parallel machines under a single renewable resource constraint, with the objective of minimizing the makespan. This realistic problem fits within both the Resource Constrained Parallel Machine Scheduling and the Energy Efficient Scheduling literature. We compare a time-indexed MILP model and a constraint programming formulation, and propose a comprehensive exact approach. The exact algorithm first computes advanced lower bounds, enhanced by a column generation strategy, and upper bounds obtained via a multi-stage heuristic that includes simulated annealing. Starting from these bounds, a binary search is applied that leverages either the CP model or a specifically developed combinatorial branch-and-bound, depending on the number of jobs. Computational experiments on two benchmark sets suggest that the proposed algorithm consistently outperforms previously published methods in terms of the number of instances solved to optimality. Preliminary tests on larger instances indicate that further research is necessary, as the proposed exact methods do not always succeed in providing an optimal solution.

CRediT authorship contribution statement

Jean-François Côté: Writing – review & editing, Validation, Supervision, Software, Methodology. **Giulia Dotti:** Writing – original draft, Validation, Software, Methodology. **Vinícius Loti de Lima:** Investigation, Conceptualization. **Carlo Alberto Magni:** Writing – review & editing. **Manuel Iori:** Writing – review & editing, Supervision.

Appendix

Lemma 1. *For any feasible solution x satisfying (2), define*

$$y_{jt} = \sum_{i=\max(0,\, t-p_j+1)}^{t} x_{ji}.$$

Then $y_{jt} = 1$ if and only if job j is active at time t, and $y_{jt} = 0$ otherwise.

Proof. By constraint (2), each job j starts exactly once: there exists a unique time S_j with $x_{j,S_j} = 1$ and $x_{jt} = 0$ for all $t \neq S_j$. Therefore:

$$y_{jt} = \sum_{i=\max(0,\, t-p_j+1)}^{t} x_{ji} = \begin{cases} 1 & \text{if } \max(0,\, t-p_j+1) \le S_j \le t, \\ 0 & \text{otherwise.} \end{cases}$$

It remains to show that the condition $t - p_j + 1 \le S_j \le t$ is equivalent to job j being active at time t. By definition, job j is active at time t if and only if

$$S_j \le t \le S_j + p_j - 1.$$

Subtracting $p_j - 1$ from the right inequality gives $S_j \ge t - p_j + 1$. Combined with the left inequality $S_j \le t$, we obtain $t-p_j+1 \le S_j \le t$, which is exactly the range of the summation index. Hence, $y_{jt} = 1$ if and only if job j is active at time t. □

Proposition 1. *Given a feasible non-preemptive schedule with start times S_j for all $j \in J$, define $x_{jt} = 1$ if $t = S_j$ and $x_{jt} = 0$ otherwise. Then x satisfies constraints (2)–(4).*

Proof. Constraint (2) holds by construction. By Lemma 1, $\sum_{j\in J} y_{jt}$ counts the number of jobs active at time t, which is at most m in a feasible schedule, satisfying (3). Similarly, the weighted sum $\sum_j r_j y_{jt}$ gives the instantaneous resource consumption at time t, which does not exceed $R_{\max}$, satisfying (4). □

Proposition 2. *Any feasible solution x satisfying (2)–(4) admits a feasible non-preemptive schedule on m identical machines.*

Proof. Given the processing intervals $I_j = [S_j, S_j + p_j)$ for each job $j \in J$, consider the interval graph $G = (V, E)$ where $V = J$ and an edge $(j, k) \in E$ exists if and only if $I_j \cap I_k \neq \emptyset$, i.e., jobs j and k overlap in time.

A clique in G corresponds to a set of mutually overlapping jobs, all of which must be active simultaneously at some time instant. By Lemma 1 and constraint (3), at most m jobs can be active at any time t, so the maximum clique size satisfies $\omega(G) \le m$.

Since interval graphs are perfect (Golumbic, 2004), their chromatic number equals their clique number: $\chi(G) = \omega(G) \le m$. Therefore, G admits a proper coloring with at most m colors, meaning no two overlapping jobs receive the same color. Assigning each color to a distinct machine yields a feasible non-preemptive schedule in which no two overlapping jobs share the same machine. □

Propositions 1 and 2 establish that the time-indexed formulation exactly characterizes the feasible domain: a schedule is feasible if and only if its corresponding matrix x satisfies the constraints. While the formulation does not include explicit job–machine assignment variables, any feasible solution always admits a constructive assignment of jobs to machines. This may be simply obtained by processing jobs in non-decreasing order of S_j, assigning each job to the first machine available at time S_j.

References

Abdeljaoued, M.A., Saadani, N.E.H., Bahroun, Z., 2020. Heuristic and metaheuristic approaches for parallel machine scheduling under resource constraints. Oper. Res. 20, 2109–2132.

Afzalirad, M., Rezaeian, J., 2016. Resource-constrained unrelated parallel machine scheduling problem with sequence dependent setup times, precedence constraints and machine eligibility restrictions. Comput. Ind. Eng. 98, 40–52.

Afzalirad, M., Shafipour, M., 2018. Design of an efficient genetic algorithm for resource-constrained unrelated parallel machine scheduling problem with machine eligibility restrictions. J. Intell. Manuf. 29, 423–437.

Akçay, F.B., Delorme, M., 2025. Solving the parallel processor scheduling and bin packing problems with contiguity constraints: Mathematical models and computational studies. European J. Oper. Res. 323 (3), 701–723.

Van den Akker, J.M., Hurkens, C.A.J., Savelsbergh, M.W.P., 2000. Time-indexed formulations for machine scheduling problems: Column generation. INFORMS J. Comput. 12 (2), 111–124.

Artigues, C., 2017. On the strength of time-indexed formulations for the resource-constrained project scheduling problem. Oper. Res. Lett. 45 (2), 154–159.

Baker, K.R., Trietsch, D., 2019. Principles of Sequencing and Scheduling, second ed. John Wiley & Sons, Inc., Hoboken, New Jersey.

Baptiste, P., Le Pape, C., Nuijten, W., 2001. Constraint-Based Scheduling: Applying Constraint Programming To Scheduling Problems. vol. 39, Kluwer Academic Publishers, Boston, Massachusetts.

Blazewicz, J., Lenstra, J.K., Rinnooy Kan, A.H.G., 1983. Scheduling subject to resource constraints: Classification and complexity. Discrete Appl. Math. 5 (1), 11–24.

Boschetti, M.A., Montaletti, L., 2010. An exact algorithm for the two-dimensional strip-packing problem. Oper. Res. 58 (6), 1774–1791.

Burdett, R.L., Corry, P., Eustace, C., Smith, S., 2021. Scheduling pre-emptible tasks with flexible resourcing options and auxiliary resource requirements. Comput. Ind. Eng. 151, 106939.

Cacchiani, V., Iori, M., Locatelli, A., Martello, S., 2022a. Knapsack problems — An overview of recent advances. Part I: Single knapsack problems. Comput. Oper. Res. 143, 105692.

Cacchiani, V., Iori, M., Locatelli, A., Martello, S., 2022b. Knapsack problems — An overview of recent advances. Part II: Multiple, multidimensional, and quadratic knapsack problems. Comput. Oper. Res. 143, 105693.

Caselli, G., Delorme, M., Iori, M., Magni, C.A., 2024. Exact algorithms for a parallel machine scheduling problem with workforce and contiguity constraints. Comput. Oper. Res. 163, 106484.

Cheng, J., Cheng, J., Lin, Y., Lu, S., Wu, P., 2025. MILP models and effective heuristic for energy-aware parallel machine scheduling with shared manufacturing. Expert Syst. Appl. 271, 126681.

Chou, Y.-L., Yang, J.-M., Wu, C.-H., 2020. An energy-aware scheduling algorithm under maximum power consumption constraints. J. Manuf. Syst. 57, 182–197.

Clautiaux, F., Jouglet, A., Carlier, J., Moukrim, A., 2008. A new constraint programming approach for the orthogonal packing problem. Comput. Oper. Res. 35 (3), 944–959.

Côté, J.-F., Dell'Amico, M., Iori, M., 2014. Combinatorial Benders' cuts for the strip packing problem. Oper. Res. 62 (3), 643–661.

De Carvalho, J.V., 2002. LP models for bin packing and cutting stock problems. European J. Oper. Res. 141 (2), 253–273.

Delorme, M., Iori, M., Martello, S., 2017. Logic based benders' decomposition for orthogonal stock cutting problems. Comput. Oper. Res. 78, 290–298.

Ding, J.-Y., Song, S., Zhang, R., Chiong, R., Wu, C., 2016. Parallel machine scheduling under time-of-use electricity prices: New models and optimization approaches. IEEE Trans. Autom. Sci. Eng. 13 (2), 1138–1154.

Edis, E.B., Oguz, C., Ozkarahan, I., 2013. Parallel machine scheduling with additional resources: Notation, classification, models and solution methods. European J. Oper. Res. 230 (3), 449–463.

Edis, E.B., Ozkarahan, I., 2012. Solution approaches for a real-life resource-constrained parallel machine scheduling problem. Int. J. Adv. Manuf. Technol. 58, 1141–1153.

Fang, K.-T., Lin, B.M.T., 2013. Parallel-machine scheduling to minimize tardiness penalty and power cost. Comput. Ind. Eng. 64 (1), 224–234.

Fanjul-Peyro, L., 2020. Models and an exact method for the unrelated parallel machine scheduling problem with setups and resources. Expert. Syst. Appl.: X 5, 100022.

Fanjul-Peyro, L., Perea, F., Ruiz, R., 2017. Models and matheuristics for the unrelated parallel machine scheduling problem with additional resources. European J. Oper. Res. 260 (2), 482–493.

Ferrari, B., Cordeau, J.-F., Delorme, M., Iori, M., Orosei, R., 2025. Satellite scheduling problems: A survey of applications in earth and outer space observation. Comput. Oper. Res. 173, 106875.

Fleszar, K., Hindi, K.S., 2018. Algorithms for the unrelated parallel machine scheduling problem with a resource constraint. European J. Oper. Res. 271 (3), 839–848.

Gaggero, M., Paolucci, M., Ronco, R., 2023. Exact and heuristic solution approaches for energy-efficient identical parallel machine scheduling with time-of-use costs. European J. Oper. Res. 311 (3), 845–866.

Gahm, C., Denz, F., Dirr, M., Tuma, A., 2016. Energy-efficient scheduling in manufacturing companies: A review and research framework. European J. Oper. Res. 248 (3), 744–757.

Garey, M.R., Graham, R.L., 1975. Bounds for multiprocessor scheduling with resource constraints. SIAM J. Comput. 4 (2), 187–200.

Geurtsen, M., Didden, J.B.H.C., Adan, J., Atan, Z., Adan, I., 2023. Production, maintenance and resource scheduling: A review. European J. Oper. Res. 305 (2), 501–529.

Golumbic, M.C., 2004. Algorithmic Graph Theory and Perfect Graphs. vol. 57, Elsevier.

Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., 1979. Optimization and approximation in deterministic sequencing and scheduling: A survey. Ann. Discret. Math. 5, 287–326.

Iori, M., De Lima, V.L., Martello, S., Miyazawa, F.K., Monaci, M., 2021. Exact solution techniques for two-dimensional cutting and packing. European J. Oper. Res. 289 (2), 399–415.

Kenmochi, M., Imamichi, T., Nonobe, K., Yagiura, M., Nagamochi, H., 2009. Exact algorithms for the two-dimensional strip packing problem with and without rotations. European J. Oper. Res. 198 (1), 73–83.

Lei, D., He, S., 2022. An adaptive artificial bee colony for unrelated parallel machine scheduling with additional resource and maintenance. Expert Syst. Appl. 205, 117577.

Lenstra, J.K., Rinnooy Kan, A.H.G., Brucker, P., 1977. Complexity of machine scheduling problems. Ann. Discret. Math. 1, 343–362.

Leung, S.C.H., Zhang, D., Sim, K.M., 2011. A two-stage intelligent search algorithm for the two-dimensional strip packing problem. European J. Oper. Res. 215 (1), 57–69.

Li, X., Liu, C., 2024. Energy-efficient scheduling in an identical parallel machine environment with peak power consumption and deadline constraints. Comput. Oper. Res. 170, 106777.

Li, Z., Yang, H., Zhang, S., Liu, G., 2016. Unrelated parallel machine scheduling problem with energy and tardiness cost. Int. J. Adv. Manuf. Technol. 84, 213–226.

Lübbecke, M.E., Desrosiers, J., 2005. Selected topics in column generation. Oper. Res. 53 (6), 1007–1023.

Martello, S., Monaci, M., Vigo, D., 2003. An exact approach to the strip-packing problem. INFORMS J. Comput. 15 (3), 310–319.

Martello, S., Pisinger, D., Toth, P., 1999. Dynamic programming and strong bounds for the 0-1 knapsack problem. Manag. Sci. 45 (3), 414–424.

Min, S.-H., Lee, S.-W., Kim, H.-J., 2025. Parallel machine scheduling with peak energy consumption limits. IEEE Trans. Autom. Sci. Eng. 22, 18063–18075.

Módos, I., Kalodkin, K., Sucha, P., Hanzálek, Z., 2019. Scheduling on dedicated machines with energy consumption limit. In: Proceedings of the 8th International Conference on Operations Research and Enterprise Systems. pp. 53–62.

Módos, I., Šucha, P., Hanzálek, Z., 2021. On parallel dedicated machines scheduling under energy consumption limit. Comput. Ind. Eng. 159, 107209.

Mor, B., Berlińska, J., 2025. Scheduling problems on parallel dedicated machines with non-renewable resource. Ann. Oper. Res. 346 (3), 2173–2193.

Mucciarini, M., Caselli, G., De Santis, D., Iori, M., Miranda-Bront, J.J., 2024. On incorporating variable consumption functions within energy-efficient parallel machine scheduling. arXiv preprint arXiv:2412.17055.

Nikolaev, A.G., Jacobson, S.H., 2010. Simulated annealing. In: Gendreau, M., Potvin, J.-Y. (Eds.), Handbook of Metaheuristics. In: International Series in Operations Research & Management Science, Vol. 146, Springer, Boston, MA.

Saberi-Aliabad, H., Reisi-Nafchi, M., Moslehi, G., 2020. Energy-efficient scheduling in an unrelated parallel-machine environment under time-of-use electricity tariffs. J. Clean. Prod. 249, 119393.

Shafiee, M., Amiri-Aref, M., Klibi, W., 2025. The integration of shared renewable resources considering setup times for the parallel machine scheduling problem. Comput. Ind. Eng. 200, 110828.

Shrouf, F., Ordieres-Meré, J., García-Sánchez, A., Ortega-Mier, M., 2014. Optimizing the production scheduling of a single machine to minimize total energy consumption costs. J. Clean. Prod. 67, 197–207.

Sousa, J.P., Wolsey, L.A., 1992. A time indexed formulation of non-preemptive single machine scheduling problems. Math. Program. 54 (1), 353–367.

Vallada, E., Villa, F., Fanjul-Peyro, L., 2019. Enriched metaheuristics for the resource constrained unrelated parallel machine scheduling problem. Comput. Oper. Res. 111, 415–424.

Ventura, J.A., Kim, D., 2003. Parallel machine scheduling with earliness–tardiness penalties and additional resource constraints. Comput. Oper. Res. 30 (13), 1945–1958.

Villa, F., Vallada, E., Fanjul-Peyro, L., 2018. Heuristic algorithms for the unrelated parallel machine scheduling problem with one scarce additional resource. Expert Syst. Appl. 93, 28–38.

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 Syst. Appl. 141, 112959.

Yunusoglu, P., Topaloglu Yildiz, S., 2022. Constraint programming approach for multi-resource-constrained unrelated parallel machine scheduling problem with sequence-dependent setup times. Int. J. Prod. Res. 60 (7), 2212–2229.