POLITECNICO DI TORINO
Repository ISTITUZIONALE

Scheduling a shared resource among parallel dedicated machines: a red-blue knapsack modeling approach

Original
Scheduling a shared resource among parallel dedicated machines: a red-blue knapsack modeling approach / Della Croce, F., Grosso, A., T'Kindt, V., Ciron, T.. - In: ANNALS OF OPERATIONS RESEARCH. - ISSN 0254-5330. - ELETTRONICO. - (2025). [10.1007/s10479-025-06944-7]

Availability:
This version is available at: 11583/3006238 since: 2025-12-31T13:09:19Z

Publisher:
Springer

Published
DOI:10.1007/s10479-025-06944-7

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

10 October 2026

Annals of Operations Research
https://doi.org/10.1007/s10479-025-06944-7

ORIGINAL RESEARCH

Scheduling a shared resource among parallel dedicated machines: a red-blue knapsack modeling approach

Federico Della Croce[1] · **Andrea Grosso**[2] · **Vincent T'kindt**[3] · **T. Ciron**[3]

Received: 15 May 2025 / Accepted: 3 November 2025
© The Author(s) 2025

Abstract
We consider a scheduling problem with dedicated parallel machines and a shared resource. Each job to be scheduled is assigned to a specific machine and can possibly be rejected if it cannot be scheduled within a predetermined time frame. Some of the jobs need an additional shared resource to be processed: this resource has to be "conservatively" scheduled, i.e must be assigned to each machine in a single non-overlapping time window. The goal is to schedule the largest possible amount of jobs (weighted by priority) in the given time frame. The problem originates from an operating room scheduling application and can be modelled as a multiple knapsack problem with additional constraints related to the scheduling of the shared resource. We establish the complexity of the problem, develop an effective MILP model and a non standard column generation approach that considerably shorten the computation times required to compute optimal solutions.

Keywords Dedicated parallel machines · Exact approach · Column generation · Knapsack problem

✉ Federico Della Croce
federico.dellacroce@polito.it

Andrea Grosso
andrea.grosso@unito.it

Vincent T'kindt
vincent.tkindt@univ-tours.fr

T. Ciron
thomas.ciron@outlook.com

[1] DIGEP, Politecnico di Torino, Turin, Italy

[2] Department of Mathematics, Università di Torino, Turin, Italy

[3] University of Tours, Tours, France

Published online: 17 November 2025

1 Introduction

We consider a deterministic scheduling problem where *m* parallel *dedicated* machines are continuously available in a given time frame, with a set on *N* jobs aiming at being processed. The machines are *dedicated,* meaning that each job is rigidly assigned to one of the machines, meaning that set *N* is partitioned into disjoint sets $N_1, N_2, \ldots, N_m$, with jobs in N_k bound to be processed by machine *k*. Additionally, a specific shared resource, needed to process certain jobs, is available and has to be scheduled. Jobs are characterized by an integer processing time and a "profit" (or priority) and can be rejected (i.e., delayed to another time frame or redirected to other hospitals). The objective is to schedule the shared resource and to select the jobs to be processed in order to maximize the total profit related to these jobs. In our specific setting, the shared resource is a piece of equipment that can be used on one machine only at a time, and has to be scheduled so that each machine can access it in a single continuous time window (to be established) and then must release it, in order to avoid (because of a choice of the management) passing the shared equipment back-and-forth several times.

Since the pioneering work of Glass et al. (2000), a significant number of papers on dedicated parallel machines scheduling with various distinct additional constraints have been published. First, the case where the processing of jobs is subject to sequence-dependent setups realized by a single server has been considered (see e.g. (Druetto et al., 2025; Huang et al., 2010)). When multiple servers can be used and job splitting is allowed the reader can refer to Kim and Lee (2021) for a survey of recent works. Also, jobs can be subject to precedence constraints and scheduled on dedicated parallel machines (Agnetis et al., 2012; Lee & Jang, 2019) or subject to conflict graphs (Bianchessi & Tresoldi, 2021; Hong & Lin, 2018; Zhang et al., 2023). When scheduling dedicated parallel machines, fixed job sequences can be given. For example, in Cheng et al. (2019, 2021) different scenario are investigated and complexity results, polynomially solvable special cases and heuristic approaches for $\mathcal{NP}$-hard cases are provided. The problem tackled in this paper accounts for scheduling jobs on dedicated parallel machines where some jobs require the use of a shared resource, or server. Similar problems have been already considered in the literature. Kellerer and Strusevich (2003b) consider the minimization of the makespan in the case of dedicated parallel machines with an additional non fully shared single resource where only a subset of the jobs consumes the resource. They study different variants of this problem and provide their complexity status, heuristics and a polynomial-time approximation scheme. The case when the shared resource can be used to reduce some job processing times is dealt with in Kellerer and Strusevich (2008). The more general case with several shared resources has also been investigated by Kellerer and Strusevich Kellerer and Strusevich (2003a) when the makespan is minimized. A complexity classification of different variants is provided. In our problem, jobs can be rejected from the current time frame in case of insufficient available resources. Scheduling with job rejection has been the matter of numerous publications (see e.g. (Engels et al., 2003) for a survey). The particular setting of dedicated parallel machines has been less studied. In Mor and Mosheiov (2024) several objective functions are considered and pseudo-polynomial time dynamic programming algorithms are proposed.

The problem tackled in this paper arises in a practical setting offered by a public hospital located in Northern Italy; it consists in deriving a daily schedule for a set of operating rooms, where each operating room is associated with a surgical speciality and a set of sur-

geries to be taken into consideration and possibly being performed in the considered day. Usually there is no chance of scheduling all such surgeries in a single day, so each surgical intervention has a nominal processing time and a nominal "profit", the latter being a function of the intervention priority. The main requirement is that the overall intervention time in each operating room does not exceed the related operating room common time availability (usually 6-8 h), and the goal is to schedule the set of surgeries with the maximum total profit. In addition, a subset of the interventions requires the use of a shared resource among the operating rooms — in the considered case, a *brilliance amplifier* — so that no two interventions using that shared resource can be processed simultaneously. For practical reasons, the shared resource is not allowed to go back and forth among the operating rooms, hence each operating room is allowed to use the shared resource for a time window to be computed, and all surgeries that require the resource must be performed within such a window. We denote this problem as the Shared Resource Operating Rooms scheduling (SRORS) problem.

Operating rooms scheduling has been widely investigated in the last 50 years. We mention here five reference surveys on the topic published between 2010 and 2025, namely (Al Amin et al., 2025; Cardoen et al., 2010; Guerriero & Guido, 2011; Harris & Claudio, 2022) and Samudra et al. (2016), but we could not manage to find there applications sufficiently close to the one in our hand. In the most general context the SRORS problem is quite complex, usually involving uncertainty in surgery times and/or several complicating constraints often related to adherence to wider master schedules, additional requirements like post-operation bed availabilities in the wards and so on. The problem, as stated in our context, somehow frees us from such complications, making the solution procedure more like a "tactical" method for scenario evaluation or for taking quick decisions. Correspondingly, we tackle the problem with deterministic data, filling a niche into the literature about dedicated machine scheduling. The contribution of the paper is threefold:

- The complexity of the problem is determined, establishing its NP-hardness (in the strong sense) and identifying pseudopolynomially solvable cases.
- A MILP model is developed, that can efficiently handle all practical instances delivered to us, in short computation times.
- A dedicated column-generation procedure is proposed, that — atypically, thanks to strong dominance relations — is able to generate all relevant columns/partial schedules for the problem up to very large instances. This procedure rules out the need for heuristics for practical size instances.

The remainder of the paper works as follows. In Sect. 2, a formal problem definition is given as well as complexity results and a fully polynomial-time approximation scheme. Section 3 presents an Integer Linear Programming formulation of the problem. An unconventional column generation procedure is developed in Sect. 4. Finally, these two exact approaches are experimentally compared in Sect. 5.

2 Problem definition and computational complexity results

2.1 Problem definition

The SRORS problem can be defined as a parallel machines scheduling problem with a shared resource where the operating rooms are machines and the surgical interventions are jobs to be scheduled. Notice that, as outlined in the introduction, the machines are dedicated as the assignment of the surgical interventions to the rooms is fixed. We formally state the problem as follows, also introducing the notation used throughout the paper.

A set of dedicated machines, numbered $1, \ldots, m$, is given: each machine $k = 1, \ldots, m$ has its own set of jobs $N_k = B_k \cup R_k$ partitioned into a set of *blue jobs* B_k that do not need the shared resource to be processed, and a set of *red jobs* R_k that need the shared resource. Both blue and red jobs $j \in N_k$ have a given profit π_j that is gained with their execution, and a processing time p_j. We denote by $\pi(S) = \sum_{j \in S} \pi_j$ and $p(S) = \sum_{j \in S} p_j$ the total profit and the total processing time of any set S of jobs. The execution of red jobs on different machines cannot overlap in time. Each machine is allowed to use the shared resource for a certain time window, that has to be computed as part of the solution: this requirement comes from the fact that moving the shared resource between rooms must be limited thus forcing to perform contiguously all the red jobs performed by the same machine. Each machine is continuously available from time 0 up to a time limit W. The goal of the SRORS problem is to find out, for each machine k, a selection of jobs organized into a block sequence $\sigma^{(k)} = (\beta_1, \rho, \beta_2)$ with (possibly empty) blocks $\beta_1, \beta_2 \subseteq B_k$ and $\rho \subseteq R_k$. Also, let us refer to $[t_1(\sigma^{(k)}), t_2(\sigma^{(k)})]$ as the processing time window of the red jobs in ρ: for any distinct machines k and k', we must have $[t_1(\sigma^{(k)}), t_2(\sigma^{(k)})] \cap [t_1(\sigma^{(k')}), t_2(\sigma^{(k')})] = \emptyset$. Computing a solution must be done so that the total profit

$$f(\sigma^{(1)}, \ldots, \sigma^{(m)}) = \sum_{k=1}^{m} \pi(\sigma^{(k)})$$

is maximized.

In order to simplify notation, we drop the machine superscript whenever there is no ambiguity. Remarkably, the SRORS problem can also be seen as involving the solution of knapsack problems (for each machine, blues jobs and red jobs have to be selected within given capacities) and of a single machine scheduling problem (scheduling of the blocks of red jobs).

2.2 Problem complexity with a fixed number of machines

Being a generalization of the 0/1 Knapsack problem (see, for instance, Pferschy (2021)), the SRORS problem is NP-hard. We show that for a constant number m of machines the SRORS problem is solvable in pseudo-polynomial time, thus assessing for this case NP-hardness in the ordinary sense.

Proposition 1 *For m constant, the SRORS problem is NP-hard in the ordinary sense.*

Proof We first consider the case $m = 2$ where, without loss of generality, we can assume that machine 1 is allowed to use the shared resource in some interval $[0, t]$ while the time window of machine 2 covers the complementary interval $[t, W]$. Hence the optimal sequences are $\sigma(1) = \rho_1\beta_1$ and $\sigma(2) = \beta_2\rho_2$. For machine index $k = 1, 2$, denote by $F_\beta^{(k)}(t)$ the optimal profit of a subset $\beta_k \subseteq B_k$ jobs on machine k with total processing time $\leq \tau$; this is the statement of a classical knapsack problem

$$F_\beta^{(k)}(\tau) = \max\left\{\sum_{j\in B_k} \pi_j x_j : \sum_{j\in B_k} p_j x_j \leq \tau,\ x_j \in \{0,1\},\ j \in B_k\right\}$$

that can be solved by recursively filling a dynamic programming table. We introduce analogous problems for $F_\rho^{(k)}(\tau)$ for red jobs in R_k, $k = 1, 2$. All values for $F_\beta^{(k)}(\tau)$, $F_\rho^{(k)}(\tau)$ can be computed in $O(nW)$ time with the corresponding solutions stored in the dynamic programming tables of size $O(nW)$. Then, by computing

$$\min\left\{F_\rho^{(1)}(t) + F_\beta^{(1)}(W-t) + F_\beta^{(2)}(t) + F_\rho^{(2)}(W-t) : t = 0, \ldots, W\right\}$$

the combined optimal schedule for both machines is found in time $O(W)$. The heaviest part of the computation is made up of the knapsack recursions; hence the overall procedure has a running time limited by $O(nW)$, showing that the $m = 2$ case is solvable in pseudo-polynomial time, as claimed above.

Now, let us consider the case when $m > 2$ with m constant. For each operating room $k = 1, \ldots, m$, it is possible to compute and store in $O(nW)$ time the best possible solutions of the red block for every size $t \in \{0 \ldots, W\}$. Hence, for all pairs t_1, t_2 denoting the sizes of the first and second blue blocks, respectively, $F_\rho^{(k)}(W - t_1 - t_2))$ can then be derived in constant time. But then, a dynamic programming algorithm on the blue jobs that selects or not each of these jobs and in case of selection places it either at the beginning of block $\beta_1^{(k)}$ or at the end of block $\beta_2^{(k)}$, can find the optimal block sequence for machine k for every interval t_1, t_2 with $0 \leq t_2 - t_1 \leq W]$ in $O(nW^2)$ time. Correspondingly, it is possible to show that if the sequence of red blocks is given, then the optimal solution value can be computed in $O(mnW^2)$ time. As the number of red blocks sequences is bounded above by $m!$, then the m-SRORS problem is pseudo-polynomially solvable for m constant (and thus it is weakly NP-hard). □

2.3 Complexity for an arbitrary number of machines

When the number of machines m is part of the instance, and following a similar course of action of Theorem 4 in Kellerer and Strusevich (2003b), we prove the following proposition.

Proposition 2 *SRORS is strongly NP-hard.*

Proof For the proof, we refer to the decision version of SRORS, where a subset of jobs must be selected and scheduled so that $\sum_{i=1}^{m}\sum_{j\in S_i} \pi_j \geq K$ for a given K, with S_i the jobs selected on machine i.

We reduce to SRORS the classical 3-PARTITION problem:

Given a multiset of integers $\mathcal{N} = \{a_1, a_2, \ldots, a_{3n}\}$ such that $\sum_{a_i \in N} a = nC$, with $C/4 < a_i < C/2$ for all $i \in \{1, 2, \ldots, 3n\}$. find a partition of $\mathcal{N}$ into n triplets $\mathcal{N}_1, \mathcal{N}_2, \ldots \mathcal{N}_n$ such that $\sum_{a_i \in \mathcal{N}_k} a_i = C$ for all k.

Given an instance of 3-partition, we define a SRORS problem with $4n + 3$ machines, consisting of three subset of machines $M_1 = \{\omega_1, \omega_2\}, M_2 = \{\alpha_1, \ldots, \alpha_{n+1}\}, M_3 = \{\gamma_1, \ldots, \gamma_{3n}\}$. We set the time horizon at $W = 2(n + 1)C$.

In the following, for the sake of conciseness, we highlight red jobs in boldface, so that we can avoid continuously referring to sets R and B.

Machine ω_1 has a set of two jobs $\Omega_1 = \{1, 2\}$ with $\pi_1 = C$, $\pi_2 = W - C$; machine ω_2 has a set of two jobs $\Omega_2 = \{1, 2\}$ with $\pi_1 = W - C$, $\pi_2 = C$.

Each machine in $M_1 = \{\alpha_1, \ldots, \alpha_{n+1}\}$ has a triplet of jobs to schedule $A_i = \{1, 2, 3\}$, with $p_1 = (2i - 1)C$, $p_2 = C$, $p_3 = W - p_2 - p_1$.

Each machine in $M_3 = \{\gamma_1, \ldots, \gamma_{3n}\}$ has to schedule a single red job $\Gamma_k = \{1\}$ with $p_1 = a_i$.

All jobs in this SRORS problem have profit $\pi_j = 1$, and the target profit K is set to $K = 6n + 4$, which amounts to requiring that all jobs must be scheduled.

Any feasible schedule must exhibit the characteristics sketched in Fig. 1.

- The red jobs on machines ω_1, ω_2 must be scheduled as first and last, respectively (the symmetric situation is possible, but does not change what follows).
- The jobs on machine α_1 must be scheduled in a $(1, 2, 3)$ sequence (or the symmetric $(3, 2, 1)$; this implies that the jobs on α_n must also be scheduled in a $(1, 2, 3)$ (or, symmetrically, $(3, 2, 1)$.
- Iterating the argument on machines α_2, α_{n-1} and so on, the schedules on the ω and α machines have the shape of Fig. 1 (or, possibly, the symmetric schedule).

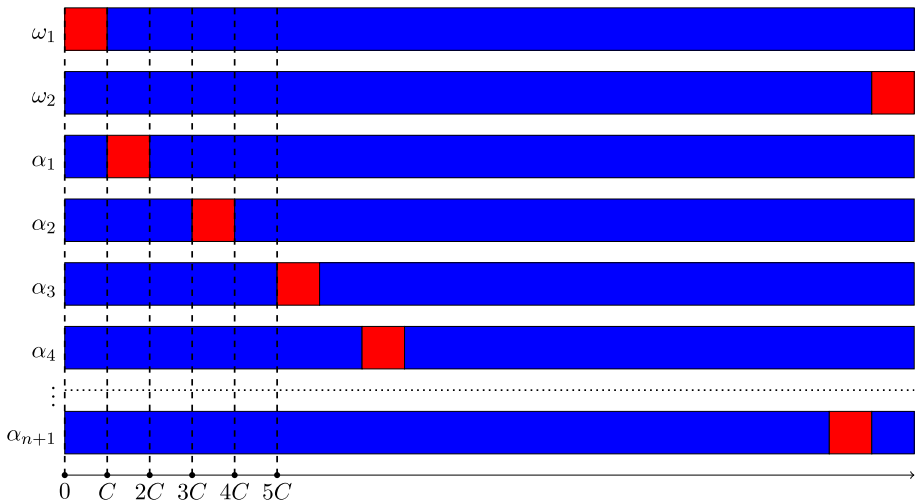

Fig. 1 Schedule scheme

As a result, the shared resource is only available for processing the jobs on the γ machines in the time intervals $[2C, 3C], [4C, 5C], \ldots$, each having a width of C units of times. So the jobs on the γ machines can be all scheduled if and only if they fit in n time slots of width C, solving the instance of 3-PARTITION. □

We can also derive the following result. Let denote by SRORS$_{min}$ the minimization version of the problem where the goal is to minimize the total profit of the rejected jobs.

Corollary 1 *The SRORS$_{min}$ problem is not approximable in polynomial or pseudo-polynomial time unless $\mathcal{P} = \mathcal{NP}$.*

Proof This result directly follows from the proof of the proposition above where it is shown that the SRORS problem is strongly $\mathcal{NP}$-hard when the number of rejected jobs is zero. This rules out the existence of an approximation algorithm for SRORS$_{min}$. □

2.4 A fully polynomial-time approximation scheme for the 2-machine case

In this section, we focus on the special case with $m = 2$ that we refer for short to as 2-SRORS. We show the existence of a fully polynomial-time approximation scheme (FPTAS) to solve it. As in the previous section, the development relies on the knapsack-like structure of 2-SRORS.

First of all, we introduce a pseudo-polynomial algorithm for 2-SRORS that relies on profit-based recursion, similarly to what can be done for the knapsack problems.

For machine index $k = 1, 2$, let $p_{\beta}^{(k)}(z)$ be the minimum total processing time of a subset of blue jobs in B_k with profit z; this is the statement of a knapsack problem that can be solved by a profit-based recursion in time $O(|B_k|^2 \max_{j \in B_k})$, filling a dynamic programming table of analogous size $O(|B_k|^2 \max_{j \in B_k})$.

Similarly, we introduce $W_{\rho}^{(k)}(v)$ as the minimum total processing time of a set of red jobs in R_k whose total profit is v. Consider Algorithm 1 denoted Solve-2-SRORS.

For the sake of simplicity, let $\pi_{\max} = \max\{\pi_j : j \in N = N_1 \cup \cdots \cup N_m\}$. Generating the dynamic programming tables involved in line 1 takes time $O(n^2 \pi_{\max})$. The solutions corresponding to the $W_{\rho}^{(k)}$, $W_{\beta}^{(k)}$ can be sorted in time $O(n\pi_{\max} \log n\pi_{\max})$ and searched in $O(\log n\pi_{\max})$.

Hence steps 1 and 2 take at most time $O(n^2 \pi_{\max}) + O(n\pi_{\max} \log n\pi_{\max}) = O(n^2 \pi_{\max})$. Then the steps on lines 4–7 take $O(\log n\pi_{\max})$ each. Since for the maximum value v^* we have $v^* \leq n\pi_{\max}$, the whole procedure has a running time bounded by $O(n^2 \pi_{\max}) + O(n\pi_{\max} \log n\pi_{\max})$.

1: Compute the dynamic programming tables for $W_\rho^{(k)}(v)$, $W_\beta^{(k)}(z)$, $k = 1, 2$;
2: Sort the solutions in tables $W_\rho^{(k)}(v)$, $W_\beta^{(k)}(z)$ by size;
3: **for** $v = 0, \ldots, v^*$ **do**
4: Determine the red block $\rho_1 \subseteq R_1$ with $\pi(\rho) = v$ and its size $\tau = W_\rho^{(1)}(v)$;
5: Determine the blue block $\beta_1 \subseteq B_1$ with size $W_\beta^{(1)} \leq W - \tau$ and maximum profit;
6: Determine the red block $\rho_2 \subseteq R_2$ with size $W_\rho^{(2)} \leq W - \tau$ and maximum profit;
7: Determine the blue block $\beta_2 \subseteq B_2$ with size $W_\beta^{(2)} \leq \tau$ and maximum profit.
8: Compose the solution $\sigma^{(1)} = \rho_1\beta_1$, $\sigma^{(2)} = \beta_2\rho_2$;
9: **end for**
10: **return** The best solution generated in the above loop;

Algorithm 1 SOLVE-2-SRORS (using profit-based tables)

The approximation takes place by rescaling the profits into

$$\pi'_j = \left\lfloor \frac{\pi_j}{K} \right\rfloor \qquad \text{for each } j \in N, \text{ with } K = \frac{\varepsilon \pi_{\max}}{n},$$

then applying SOLVE-2-SRORS to the rescaled instance, getting a solution $S' = (\rho_1\beta_1; \beta_2, \rho_2)$ that maximizes $\pi'(S')$ for the rescaled problem and is a feasible solution for the original problem. By working along the lines of the classical proof for the knapsack problem, the following proposition establishes that the outlined procedure is an FPTAS for 2-SRORS.

Proposition 3 *If S^* is the optimal solution for 2-SRORS, $\pi(S') \geq (1 - \varepsilon)\pi(S^*)$, and S' can be computed within a running time at most polynomial in n and $1/\varepsilon$.*

Proof Note that $-K' \leq K$ because of scaling and rounding. Thus,

$$\begin{aligned}\pi(S') &\geq K\pi'(S') \geq K\pi'(S^*) \geq \pi(S^*) - Kn = \pi(S^*) - \varepsilon\pi_{\max} \geq \\ &\geq \pi(S^*) - \varepsilon\pi(S^*) = (1 - \varepsilon)\pi(S^*).\end{aligned}$$

The running time is established by the time bound for SOLVE-2-SRORS,

$$\begin{aligned}O(n^2\pi'_{\max} + n\pi'_{\max}\log n\pi'_{\max}) &= O\left(n^2\frac{\pi_{\max}}{K} + n\frac{\pi_{\max}}{K}\log\frac{n\pi_{\max}}{K}\right) = \\ &= O\left(\frac{n^3}{\varepsilon} + \frac{n^2}{\varepsilon}\log\frac{n^2}{\varepsilon}\right). \ \square\end{aligned}$$

3 An ILP formulation

An Integer Linear Programming (ILP) formulation of the problem can be derived by using a pseudo-polynomial number of binary variables $z_{i,j,t}$ indicating if job *j* starts processing at time *t* in operating room *i*. However, such a formulation quickly fails to reach optimality in reasonable time already with a limited number of jobs and operating rooms. Instead, an improved knapsack-based formulation of the SRORS problem can be devised as follows.

For each blue job of machine k $j \in B_k$ we introduce two binary variables x^1_{kj} and x^2_{kj} such that $x^1_{kj} = 1$ iff $j \in \beta^{(k)}_1$, $x^2_{kj} = 1$ iff $j \in \beta^{(k)}_2$. For each red job j of machine k ($j \in B_k$) we introduce a binary variable $y_{kj} = 1$ iff $j \in \rho^{(k)}$. For each machine $k = 1, \ldots, m$, we introduce two continuous variables $t^{(k)}_1, t^{(k)}_2 \geq 0$ denoting the start time and end time of the red window for $\sigma^{(k)}$. Finally, for each pair k, ℓ (with $k < \ell$) of machines, we introduce a binary variable where $z_{k\ell} = 1$ iff machine k uses the shared resource before machine ℓ (this implies $t^{(k)}_2 \leq t^{(\ell)}_1$).

The following ILP model holds.

$$\text{maximize} \quad \sum_{k=1}^{m} \sum_{j \in B_k} \pi_j (x^1_{kj} + x^2_{kj}) + \sum_{k=1}^{m} \sum_{j \in R_k} \pi_j y_{kj} \tag{1}$$

$$\text{subject to} \quad \sum_{j \in B_k} p_j x^1_{kj} \quad \leq t^{(k)}_1 k = 1, \ldots, m \tag{2}$$

$$\sum_{j \in R_k} p_j y_{kj} \quad \leq t^{(k)}_2 - t^{(k)}_1, k = 1, \ldots, m \tag{3}$$

$$\sum_{j \in B_k} p_j x^2_{kj} \quad \leq W - t^{(k)}_2, k = 1, \ldots, m \tag{4}$$

$$\sum_{k=1}^{m} \sum_{j \in R_k} p_j y_{kj} \quad \leq W \tag{5}$$

$$t^{(k)}_1 \leq t^{(k)}_2 \qquad k = 1, \cdots, m \tag{6}$$

$$x^1_{kj} + x^2_{kj} \leq 1 \quad j \in B_k, \, k = 1, \ldots, m \tag{7}$$

$$t^{(k)}_2 \leq t^{(\ell)}_1 + W(1 - z_{k\ell}) \quad k, \ell = 1, \ldots, m, \, k < \ell \tag{8}$$

$$t^{(\ell)}_2 \leq t^{(k)}_1 + W z_{k\ell} \quad k, \ell = 1, \ldots, m, \, k < \ell \tag{9}$$

$$x^1_{kj}, x^2_{kj} \in \{0, 1\} \quad k = 1, \ldots, m, \, j \in B_k \tag{10}$$

$$y_{kj} \in \{0, 1\} \quad k = 1, \ldots, m, \, j \in R_k \tag{11}$$

$$z_{k\ell} \in \{0,1\} \quad k,\ell = 1,\ldots,m,\ k < \ell \tag{12}$$

$$t_1^{(k)}, t_2^{(k)} \geq 0 \quad k = 1,\ldots,m. \tag{13}$$

In the model, the objective function (1) maximizes the total profit of the selected jobs. Constraint (2) states that for every machine k, the total processing time of the blue jobs scheduled before ρ_k is not superior to the starting time $t_1^{(k)}$. Constraint (3) states that for every machine k, the total processing time of the red jobs is equal to $t_2^{(k)} - t_1^{(k)}$. Constraint (4) states that for every machine k, the total processing time of the blue jobs scheduled after ρ_k is not superior to $W - t_2^{(k)}$ so that they can be operated within the time availability period W. Constraint (5) states that the total processing time of the red jobs is not superior to the time availability period W (we remark that this constraint is redundant but positively affects the performances of an ILP solver). Constraint (6) states that for every machine k the starting time of the red block is not superior to the ending time. Constraint (7) states that for machine k, the blue jobs, if scheduled, must be processed either before or after the red block (i.e., in β_1 or β_2). Constraints (8-9) are the so-called disjunctive constraints on the red blocks, that is, for each pair of machines k, ℓ either the red block of k precedes the red block of ℓ, or, vice versa, the opposite holds.

4 A dedicated exact solution algorithm

4.1 Justification of the approach

The difficulty in solving the SRORS problem comes mainly from the selection of blue/red jobs for each machine and the sequencing of all the blocks of red jobs through all the machines. We initially designed and experimented a Branch-and-Bound algorithm based on a dedicated branching scheme that is quite intuitive: for a given machine which schedule $\sigma = (\beta_1, \rho, \beta_2)$ is under construction, each blue job is either selected in β_1 or in β_2, or rejected. Similarly, each red job is either selected in ρ or rejected except for the first selected red job: its starting time $t_1(\sigma)$ has to be determined. So, branching on blue jobs generates 3 children nodes and on any, but the first, red job generates 2 children nodes. Branching on the first red job generates $O(W)$ children nodes. The consequence that we faced is that, even with strong bounds and dominances conditions, the resulting Branch-and-Bound algorithm gave worst results than solving the ILP formulation given in Sect. 3. One major explanation of this, is that in mathematical solvers are embedded techniques to remove symmetries which seem to handle pretty well the symmetries generated when scheduling the first red job on the machines.

Instead of branching on jobs, we propose to branch on machine schedules. Such approaches, known as column generation approaches, usually lead to iteratively generate columns (machine schedules) until no more improving columns exist. We found out that, to limit the symmetries induced by the blocks of red jobs, it is profitable to generate for each machine all columns and eliminate the dominated ones. Thus, we do not have to cope with the problem of generating improving columns that may not be simple to solve in practice.

All the details of the proposed approach are given in the remainder of this section.

4.2 A tailored column generation approach

In this section we aim to solve the SRORS problem as a very large MILP with an exponential number of variables by means of a non-standard column generation approach. For each machine k, let $\mathcal{S}_k$ be the set of all possible schedules $\sigma = (\beta_1 \rho \beta_2)$ for such machine. We aim at solving the following linear program, with binary control variables $x_\sigma = 1$ iff $\sigma \in \mathcal{S}_k$ is scheduled on machine k.

$$\text{minimize} \sum_{k=1}^{m} \sum_{\sigma \in \mathcal{S}_k} \pi(\sigma) x_\sigma \tag{14}$$

$$\text{subject to} \sum_{\sigma \in \mathcal{S}_k} x_\sigma = 1 \quad \text{for } k = 1, \ldots, m \tag{15}$$

$$\sum_{k=1}^{m} \sum_{\substack{\sigma \in \mathcal{S}_k : \\ t_1(\sigma) \le \tau \le t_2(\sigma)}} x_\sigma \le 1 \quad \text{for } \tau = 1, \ldots, W \tag{16}$$

$$x_\sigma \in \{0, 1\} \quad \text{for } \sigma \in \mathcal{S}_k,\ k = 1, \ldots, m. \tag{17}$$

The objective function represents the total profit for scheduling all the machines, constraints (15) require that only one schedule is selected for each machine and constraints (16) require that the selected schedules have non-overlapping windows for processing red jobs (W constraints are imposed for each possible value of τ).

This ILP model is in principle very large (as there are as many variables x_σ corresponding to schedules in sets $\mathcal{S}_k$) to be solved with a classic iterative procedure. Starting with reduced sets $\mathcal{S}_k$ a standard column generation procedure would be iteratively used to add *improving schedules* to these sets until an optimal solution is reached. But usually, such an approach can be very time consuming. We rather consider the generation, at once, of all machine schedules σ before eliminating those that are detected to be dominated. At the end, this leads to sets $\mathcal{S}_k$ of reduced sizes on which the model (14)–(17) can be explicitly formulated and solved. The complete exact algorithm is given in Algorithm 2 and we present hereafter its different components.

4.3 Elimination of dominated jobs

First of all, jobs that are never worth to be scheduled are identified and eliminated by applying the following

Property 1 Given a red job $j \in R_k$, define

$$\hat{B}_k(j) = \{i \in B_k : i \neq j,\ p_i \le p_j \text{ and } \pi_i \ge \pi_j\},$$
$$\hat{R}_k(j) = \{i \in R_k : i \neq j,\ p_i \le p_j \text{ and } \pi_i \ge \pi_j\},$$

with at least one strict inequality, in the definitions of $\hat{B}_k(j)$ and $\hat{R}_k(j)$. If

$$p[\hat{B}_k(j)] + p[\hat{R}_k(j)] + p_j > W \tag{18}$$

then there exists an optimal solution where job j is not scheduled.

Proof A simple interchange argument shows that in any optimal solution each of the jobs in $\hat{R}_i(j)$, $\hat{B}_i(j)$ must also be scheduled whenever job j is scheduled. But then inequality (18) rules out the possibility of scheduling j. □

4.4 Generation of schedules and elimination of dominated schedules

Schedules are generated by dynamic programming. On a given machine k with red window $[t_1, t_2]$, all possible red blocks are solutions for the knapsack problem

$$\max \left\{ \sum_{i \in R_k} \pi_i y_i : \sum_{i \in R_k} p_i y_i \leq t_2 - t_1,\ y_i \in \{0,1\}, i \in R_k \right\}, \tag{19}$$

while all the blue block pairs are solutions to the 2-knapsack problem

$$\begin{aligned} \max \Bigg\{ \sum_{i \in B_k} (\pi_i x_i^1 + \pi_i x_i^2) : & \sum_{i \in B_k} p_i x_i^1 \leq t_1, \\ & \sum_{i \in B_k} p_i x_i^2 \leq W - t_2, \\ & x_i^1 + x_i^2 \leq 1, \quad x_i^1, x_i^2 \in \{0,1\}, i \in B_k \Bigg\}, \end{aligned} \tag{20}$$

Filling such tables requires a pseudo-polynomial computation time of the order of $O(nW)$ and $O(nW^2)$ respectively. Schedules are obtained by composing solutions of (19) and (20). Given two schedules $\sigma_1, \sigma_2 \in \mathcal{S}_k$, σ_1 dominates σ_2 if $\pi_{\sigma_1} = \pi_{\sigma_2}$ and either $(t_1(\sigma_1) = t_1(\sigma_2) \wedge t_2(\sigma_1) < t_2(\sigma_2)$ or $t_2(\sigma_1) = t_2(\sigma_2) \wedge t_1(\sigma_1) > t_1(\sigma_2)$. The time complexity of eliminating the dominated schedules for a machine is of the order of $O(W^2)$: note that given a red window $[t_1, t_2]$ for a machine k the corresponding schedule can be generated by picking the appropriate solutions from the dynamic programming tables of (19) and (20). In practice the procedure is very quick, and eliminates a substantial amount of schedules. We denote by $\mathcal{S}'_k$ the set of non-dominated schedules for each machine k.

4.5 Upper bounding, lower bounding and fixing

Aside from the continuous relaxation, a relaxed solution for the integer program (14)–(17) can be obtained by neglecting the non-overlapping constraints on the red windows in the non-dominated schedules, and solving a knapsack problem on the set of such schedules:

$$\text{minimize} \sum_{k=1}^{m} \sum_{\sigma \in \mathcal{S}_k} \pi(\sigma) x_\sigma \tag{21}$$

$$\text{subject to} \sum_{k=1}^{m} \sum_{\sigma_\in \mathcal{S}_k} [t_2(\sigma) - t_1(\sigma)] x_\sigma \leq W \tag{22}$$

$$x_\sigma \in \{0,1\} \qquad \sigma \in \mathcal{S}'_k,\ k = 1, \ldots, m. \tag{23}$$

Solving the knapsack problem (21)–(23) requires $O(W \sum_k |\mathcal{S}'_k|)$ time. Consider the relaxed solution computed by solving (21)–(23). Then, a feasible solution can be searched for by trying to eliminate the overlapping red windows in the relaxed solution. This can be done by keeping the red windows and considering all possible non-overlapping permutations of such windows. The brute-force enumeration is usually possible because the number of machines is small, in practical settings. A time-limit on this procedure is enforced anyway, as a fail-safe. In practice, these procedures are very effective: the resulting gap between LB and UB is usually below 0.01%.

4.6 Schedule elimination via variable fixing

For a given machine k let $\sigma \in \mathcal{S}_k$ and $\mathrm{UB}_k[\sigma; W - (t_2(\sigma) - t_1(\sigma)]$ be the upper bound computed neglecting machine k and allowing only $W - (t_2(\sigma) - t_1(\sigma))$ units of red time. A variable fixing procedure can then be applied on variables x_σ of each $\mathcal{S}_k$: schedule σ is removed from consideration (x_σ is fixed at 0) if $\mathrm{UB}_k[\sigma; W - (t_2(\sigma) - t_1(\sigma))] + \pi(\sigma) < \mathrm{LB}$. The overall procedure is then sketched in Algorithm 2.

```
Remove dominated jobs by Property 1;
Generate the schedules in 𝔖_1, ..., 𝔖_m;
for k ∈ {1, 2, ..., m} do
    Remove dominated schedules in 𝔖_k;
end for
Compute the upper bound UB;
Compute the lower bound LB;
for k ∈ {1, 2, ..., m} do
    for σ ∈ 𝔖_k do
        Eliminate σ if UB[W − (t_2(σ) − t_1(σ))] + π(σ) ≤ LB;
    end for
end for
Solve (14)–(17) on the remaining schedules;
```

Algorithm 2 Solution approach for SRORS based on column generation

5 Computational results

We tested MILP (1)–(13) and the column generation procedure on different types of instances. A first series of tests was performed on "realistic" random instances: such instances were generated according to the following specifications given by the involved hospital.

- Number *m* of machines (operating rooms) set to 5 or 10.
- Number of jobs per machine (surgeries per operating room) set to 10 or 20.
- Time horizon set to $W = 360$ or 720 minutes. $W = 360$ represents the most common case for the availability of operating rooms corresponding to 6 hours of operating room availability. $W = 720$ (corresponding to 12 hours) is used in order to create harder instances.
- Processing time of each job p_j between 30 and 240 (minutes of surgery).
- Number of red jobs randomly set from 10 to 30% of the jobs.
- Profits strongly correlated with job processing times, with $\pi_j = \alpha p_j$, $\alpha \in \{1, 2, 5\}$.

The tests were performed on an Intel i5-8500 CPU 3.00 GHz with 16GB RAM running Windows. For all the the $W = 360$ instances, as reported in Table 1, CPLEX was able to solve the MILP formulation (1)–(13) to optimality within a handful of seconds, up to the instances with $n = 20$ jobs per machine.

At $n = 20$ the difficulty of the instances seems to increase considerably. The average CPU time is approximately 90 seconds with a worst case of approximately half an hour. Applying the column generation approach to the same instances crashes the CPU times to a fraction of a second on all instances. We report in the table, instead of the node count that is usually low on the final problem, the figures relative to the initial number of generated columns an the final number of columns, after all the pruning procedures have been applied.

For the instances with a larger time horizon $W = 720$, the performances of CPLEX on the MILP formulation (1)–(13) gradually deteriorate, with the solver being able to only partially handle the $m = 5$, $n = 15$ instances within a time limit of 15,000 seconds (the relevant results available from the authors are therefore omitted). We note, however, that for the unsolved instances the delivered optimality gaps were quite small.

The remaining tests were performed in order to test the column generation approach. Table 2 shows the related performances on instances with $W = 720, 1080, 1440$ minutes.

The $n = 20$ cases are (quite expectedly) still the hardest instances; it is interesting to note that, as *W* increases, the procedures for pruning columns become even more effective if the percentage of pruned columns is taken into account. Apparently more and more dominated machine schedules appear as *W* increases. The CPU times obviously increase due to the larger problems that have to be build, but apparently such problems are not very difficult, despite the number of columns involved.

Indeed the CPU times are still very low — below half a minute in the worst case — for the largest instances ($W = 1440$, $m = 10$, $n = 20$).

Given these results, we tried to generate harder instances setting a large time horizon $W = 1000$ and pushing the number of jobs per machine to very large values $n = 40, 60, 80, 100$ and *m* up to 20 machines.

Table 1 Tests with MILP and Column generation on realistic large instances

Size			MILP				Column Gener.					
			CPU		NODES		CPU		#InitCols		#FinCols	
m	*n*	*W*	avg	max	avg	max	avg	max	avg	max	avg	max
5	10	360	0.06	0.23	522	3446	0.06	0.26	636	2187	415	1070
5	15	360	0.17	0.36	1597	4859	0.06	0.07	315	1310	224	992
5	20	360	0.87	7.66	11,658	132,432	0.07	0.09	435	1979	250	1045
10	10	360	0.36	0.54	1093	2372	0.09	0.14	1250	2343	930	1890
10	15	360	1.28	6.92	4957	2966	0.12	0.16	744	3579	444	1214
10	20	360	87.49	1877.10	506,543	10,787,646	0.14	0.20	638	2585	329	1322

The relevant results are depicted in Table 3. The column generation approach, despite of the number of columns handled and the size of such problems was able to solve such instances with a worst case CPU time not exceeding 1800 seconds.

6 Conclusions

We have considered a dedicated parallel machines scheduling problem originated from a practical operating rooms scheduling problem faced by a public hospital located in Northern Italy. The problem involves the presence of a shared resource (a brilliance amplifier) required by conflicting surgeries on different operating rooms.

The related computational complexity has been fully established and a pair of exact solution approaches have been proposed. The first approach exploits a knapsack-based formulation of the problem and is capable of solving, in limited time, instances with up to 200 surgeries and 10 operating rooms. An alternative improved solution approach based on column generation dominates the knapsack-based approach and solves to optimality instances with up to 800 surgeries and 10 operating rooms in a few seconds. This approach, differently from standard column generation procedures, generates in one shot all necessary columns after having eliminated all dominated ones so that a unique ILP model with a relatively small subset of variables needs to be solved.

Several further research issues are conceivable. First, the generalization of the SRORS problem to the case of m identical shared resources is worthy of investigation. Second, the case where several distinct shared resources are required (for instance, always relating to the operating rooms setting, a set of brilliance amplifiers plus a team of anesthesiologists shared by more operating rooms) may possibly occur. Finally, the tailored column generation approach applied here may be generalized to other problems where the solution of the pricing problem is computationally expensive.

Table 2 Tests with column generation on large size instances (I)

Size			CPU (secs)		#InitCols		#FinCols	
m	*n*	*W*	avg	max	avg	max	avg	max
5	10	720	0.63	5.58	5358	21,365	2135	8561
5	15	720	0.48	0.81	4171	23,930	1128	8934
5	20	720	0.56	0.81	4534	19,701	400	3244
10	10	720	0.77	1.30	12,339	27,291	5660	18,142
10	15	720	0.97	1.41	7668	30,579	2106	9405
10	20	720	1.09	1.21	5488	15,165	329	1697
Size			CPU (secs)		#InitCols		#FinCols	
m	*n*	*W*	avg	max	avg	max	avg	max
5	30	1080	1.65	5.42	23,124	87,602	8878	36,309
5	45	1080	1.81	7.29	20,623	105,352	1299	8622
5	60	1080	1.96	4.02	51,028	87,242	133	1719
10	30	1080	3.85	12.52	54,075	169,229	18,577	86,910
10	45	1080	3.78	7.16	51,382	116,488	3311	11,176
10	60	1080	3.80	4.23	99,186	176,347	102	1823
Size			CPU (secs)		#InitCols		#FinCols	
m	*n*	*W*	avg	max	avg	max	avg	max
5	40	1440	5.99	21.03	69,632	203,692	18,128	84,892
5	60	1440	4.78	13.20	114,214	253,045	1476	13,174
5	80	1440	4.89	5.31	241,335	419,610	48	1289
10	40	1440	13.36	45.31	145,469	257,572	42,735	123,567
10	60	1440	10.71	43.75	237,348	469,280	1838	19,041
10	80	1440	11.39	26.23	401,199	672,804	148	2281

Table 3 Tests with column generation on large size instances (II)

Size			CPU (secs)		#InitCols		#FinCols	
m	*n*	*W*	avg	max	avg	max	avg	max
5	40	1000	14.05	70.10	171,847	302,241	153,202	287,090
5	60	1000	46.66	151.79	354,542	649,705	284,198	564,520
5	80	1000	160.68	443.41	667,891	1,159,685	584,183	1,106,046
5	100	1000	252.28	521.80	882,588	1,285,003	813,492	1,256,176
10	40	1000	23.42	73.79	321,930	483,379	282,613	452,810
10	60	1000	101.84	259.58	711,425	1,194,306	585,448	1,011,131
10	80	1000	253.42	571.47	1,174,093	1,788,908	1,017,958	1,701,383
10	100	1000	529.62	952.83	1,778,479	2,427,057	1,636,146	2,305,818
Size			CPU (secs)		#InitCols		#FinCols	
m	*n*	*W*	avg	max	avg	max	avg	max
15	40	1000	57.40	301.27	509,498	941,342	450,645	869,827
15	60	1000	162.96	378.22	1,065,351	1,793,567	865,605	1,582,575
15	80	1000	413.65	724.16	1,870,755	2,483,647	1,615,745	2,251,469
15	100	1000	857.56	1,562.82	2,790,576	3,894,450	2,586,009	3,727,577
20	40	1000	58.83	307.52	640,795	1,004,320	557,641	936,186
20	60	1000	225.43	436.30	1,471,310	2,158,904	1,200,291	1,836,656
20	80	1000	587.96	943.81	2,540,903	3,326,854	2,235,932	3,079,934
20	100	1000	1,257.19	1,765.30	3,852,021	4,604,790	3,592,303	4,353,281

Funding Open access funding provided by Politecnico di Torino within the CRUI-CARE Agreement. Funding was provided by Italian Ministry of Education, Grant No. TESUN-83486178370409.

Declarations

Conflict of interest All authors declare that they have no conflict of interest.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

Agnetis, A., Kellerer, H., Nicosia, G., & Pacifici, A. (2012). Parallel dedicated machines scheduling with chain precedence constraints. *European Journal of Operational Research, 221*(2), 296–305. https://doi.org/10.1016/j.ejor.2012.03.040

Al Amin, M., Baldacci, R., Kayvanfar, V., A comprehensive review on operating room scheduling and optimization. Operational Research (2025), *25* (3), https://doi.org/10.1007/s12351-024-00884-z.

Bianchessi, N., & Tresoldi, E. (2021). A stand-alone branch-and-price algorithm for identical parallel machine scheduling with conflicts. *Computers & Operations Research, 136*, 1054645. https://doi.org/10.1016/j.cor.2021.105464

Cardoen, B., Demeulemeester, E., & Belien, J. (2010). Operating room planning and scheduling: A literature review. *European Journal of Operational Research, 201*(3), 921–932. https://doi.org/10.1016/j.ejor.2009.04.011

Cheng, T. C. E., Kravchenko, S. A., & Lin, B. M. (2019). Server scheduling on parallel dedicated machines with fixed job sequences. *Naval Research Logistics, 66*(4), 321–332. https://doi.org/10.1002/nav.21846

Cheng, T. C. E., Kravchenko, S. A., & Lin, B. M. (2021). Complexity of server scheduling on parallel dedicated machines subject to fixed job sequences. *Journal of the Operational Research Society, 72*(10), 2286–2289. https://doi.org/10.1080/01605682.2020.1779625

Druetto, A., Grosso, A., Jeunet, J., Salassa, F., & Chiaramello, E. (2025). Exact approaches for parallel machine scheduling with loading and unloading servers. *Computers & Industrial Engineering, 210*, Article 111550. https://doi.org/10.1016/j.cie.2025.111550

Engels, D. W., Karger, D. R., Kolliopoulos, S. G., Sengupta, S., Uma, R. N., & Wein, J. (2003). Techniques for scheduling with rejection. *Journal of Algorithms, 49*(1), 175–191. https://doi.org/10.1016/S0196-6774(03)00078-6

Glass, C. A., Shafransky, Y. M., & Strusevich, V. A. (2000). Scheduling for parallel dedicated machines with a single server. *Naval Research Logistics,47*, 304–328. https://doi.org/10.1002/(SICI)1520-6750(200006)47:4%3C304::AID-NAV3%3E3.0.CO;2-1

Guerriero, F., & Guido, R. (2011). Operational research in the management of the operating theatre: A survey. *Health Care Management Science, 14*(1), 89–114. https://doi.org/10.1007/s10729-010-9143-6

Harris, S., & Claudio, D. (2022). Current trends in operating room scheduling 2015 to 2020: A literature review. *Operations Research Forum, 3*(21), 42. https://doi.org/10.1007/s43069-022-00134-y

Hong, H. C., & Lin, B. M. T. (2018). Parallel dedicated machine scheduling with conflict graphs. *Computers and Industrial Engineering, 124*, 316–321. https://doi.org/10.1016/j.cie.2018.07.035

Huang, S., Cai, L., & Zhang, X. (2010). Parallel dedicated machine scheduling problem with sequence-dependent setups and a single server. *Computers & Operations Research, 58*(1), 165–174. https://doi.org/10.1016/j.cie.2009.10.003

Kellerer, H., & Strusevich, V. A. (2003a). Scheduling problems for parallel dedicated machines under multiple resource constraints. *Discrete Applied Mathematics, 133*(1–3), 45–68. https://doi.org/10.1016/S0166-218X(03)00433-5

Kellerer, H., & Strusevich, V. A. (2003b). Scheduling parallel dedicated machines under a single non-shared resource. *European Journal of Operational Research, 147*(2), 345–364. https://doi.org/10.1016/S0377-2217(02)00246-1

Kellerer, H., & Strusevich, V. A. (2008). Scheduling parallel dedicated machines with the speeding up resource. *Naval Research Logistics, 55*(5), 377–389. https://doi.org/10.1002/nav.20292

Kim, H. J., & Lee, J. H. (2021). Scheduling uniform parallel dedicated machines with job splitting, sequence-dependent setup times, and multiple servers. *Computers & Operations Research, 126*, Article 105115. https://doi.org/10.1016/j.cor.2020.105115

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

Mor, B., & Mosheiov, G. (2024). Scheduling on parallel dedicated machines with job rejection. *International Journal of Production Research, 62*(19), 6933–6940. https://doi.org/10.1080/00207543.2024.2314714

Pferschy, U., Pisinger, D., Kellerer, H. (2021). *Knapsack problems*. Springer-Verlag.

Samudra, M., Van Riet, C., Demeulemeester, E., Cardoen, B., Vansteenkiste, N., & Rademakers, F. E. (2016). Scheduling operating rooms: achievements, challenges and pitfalls. *Journal of Scheduling, 19*, 493–525. https://doi.org/10.1007/s10951-016-0489-6

Zhang, A., Zhang, L., Chen, Y., Chen, G., & Wang, X. (2023). Complexity and approximation algorithms for two parallel dedicated machine scheduling with conflict constraints. *Theoretical Computer Science, 941*(4), 167–179. https://doi.org/10.1016/j.tcs.2022.11.012

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.