Proceedings of the Thirty-Sixth International Conference on Automated Planning and Scheduling (ICAPS 2026)

GPMS: A Generalized Parallel Machine Scheduling Framework with Rich Temporal and Resource Constraints

Lukas Frühwirth[1], Christoph Einspieler[2], Nysret Musliu[1], Felix Winter[1]
[1]DBAI, TU Wien, Vienna, Austria
[2]MCP GmbH, Vienna, Austria
{lukas.fruehwirth, nysret.musliu, felix.winter}@tuwien.ac.at, christoph.einspieler@mcp-alfa.com

Abstract

Classical scheduling problems such as Unrelated Parallel Machine Scheduling (UPMSP), Flexible Job Shop Scheduling (FJSP), and Resource-Constrained Project Scheduling (RCPSP) each capture important aspects of industrial scheduling, but real-world applications often require a combination of constraints from several of these formulations and additional requirements that are typically not supported in standard models. We propose a Generalized Parallel Machine Scheduling (GPMS) framework that, to the best of our knowledge, is the most general machine scheduling formulation to date and unifies machine eligibility with machine-dependent processing times, sequence- and machine-dependent setup times, rich temporal constraints, and secondary-resource constraints in a single formulation. Temporal constraints include machine calendars and precedence relations with min/max time lags and conditional machine eligibilities. Secondary resources are modeled with capacity calendars and pulse or step demands.

To model and solve this generalized problem, we investigate a constraint programming approach and propose two CP models: a solver-agnostic high-level model and an interval-variable model. We also develop a randomized construction heuristic that finds feasible, high-quality schedules even for large and highly constrained instances; these schedules serve as warm starts for the interval-variable model. We evaluate 216 generated and 91 real-world instances, comparing the interval-variable model to the solver-agnostic model and to the construction heuristic. The warm-start interval-variable model attains the best objective on the majority of both generated and real-world instances, proves optimality for many small-to-medium instances, and finds near-optimal schedules for most industrial instances within a one-hour time limit. We release all models, code, and instances to support reproducible research.

Models, Code & Instances —
https://github.com/lukasfruehwirth/GPMS

1 Introduction

Although many industrial scheduling problems can be modeled and solved with existing methods, recent real-world applications often combine requirements that extend beyond the boundaries of classical formulations (Schlenkrich and Parragh 2023). For example, one plant may resemble a parallel-machine environment yet require project-style temporal logic with minimum and maximum time lags; another may look like a job shop with operation sequences but also depends on secondary resources and shift calendars. Often, even small changes in requirements, such as adding fixed start or end times or introducing step-like inventory demands, can significantly disrupt existing models and algorithms.

As a result, practitioners often face a choice between (i) simplifying the model at the risk of producing schedules that are no longer usable in real operations, or (ii) maintaining factory-specific formulations that are difficult to adapt when objectives or constraints change.

To address the difficulty of capturing all these requirements within a single, flexible constraint model, we propose a Generalized Parallel Machine Scheduling (GPMS) framework that unifies modeling features from UPMSP, FJSP, and RCPSP. GPMS supports unit-capacity and non-capacitated machines, machine eligibility with machine-dependent processing times, and sequence- and machine-dependent setup times. Temporal constraints include machine calendars, release dates, due dates, deadlines, fixed timings, and precedence relations with minimum and maximum time lags, transfer fractions, immediate-successor constraints, and conditional machine eligibilities. Secondary resources are modeled through capacity calendars and pulse or step demands. This combination closely reflects complex real industrial requirements, but, to the best of our knowledge, is not supported by any single existing parallel machine scheduling model in the literature.

For example, UPMSP research has extensively studied sequence-dependent setups, machine eligibility, and diverse objective functions (Allahverdi 2015; Vallada and Ruiz 2011; Durasević and Jakobović 2023; Moser et al. 2022). Variants with release dates, precedences, additional resources, or machine calendars have also been considered (Edis, Oguz, and Ozkarahan 2013; Afzalirad and Rezaeian 2016; Yunusoglu and Topaloglu Yildiz 2022; Santoro and Junqueira 2023; Gacias, Artigues, and Lopez 2010; Dang et al. 2021), with one of the most general variants to date combining precedences with minimum time lags, machine calendars, and calendar-based cumulative resource constraints (Einspieler et al. 2025). However, existing work does not integrate machine calendars, temporal constraints with time lags, and secondary-resource

calendars with step demands in a single model.

Previous work on the FJSP offers machine flexibility and routing alternatives, with a recent survey synthesizing three decades of research (Dauzère-Pérès et al. 2024). Extensions include sequence-dependent setups, arbitrary precedence graphs, and multiple resource types (Shen, Dauzère-Pérès, and Neufeld 2018; Kasapidis et al. 2021, 2025). Further, the RCPSP literature contributes expressive temporal and resource semantics. Foundational work (Brucker et al. 1999; Hartmann and Briskorn 2022) defines the core taxonomy; generalized precedence with time lags and resource calendars are handled in RCPSP/max and RCPSP/max-cal (De Reyck and Herroelen 1999; Kreter, Rieck, and Zimmermann 2016), and multi-mode RCPSP introduces alternatives analogous to machine-dependent processing times (Weglarz et al. 2011). However, existing FJSP and RCPSP formulations do not support the full range of machine-related constraints in our framework, such as sequence- and machine-dependent setups, machine calendars, and conditional machine eligibilities.

The following summarizes the contributions of this work:

- We formalize GPMS, a unified scheduling framework that integrates constraints from UPMSP, FJSP, and RCPSP, and additionally supports constraints like conditional machine eligibilities and secondary-resource calendars with pulse and step demands.
- We investigate two novel constraint programming models: a solver-independent MiniZinc model and an interval-variable model leveraging global scheduling constraints such as `NoOverlap`, `Circuit`, `Cumulative`, and `Reservoir`.
- We propose an innovative randomized construction heuristic (CH) tailored to handle calendars, max-lag precedences, conditional eligibilities, fixed timings, and step demands. Existing construction heuristics (Durasević and Jakobović 2023) cannot handle such constraints sufficiently; our CH reliably finds feasible, high-quality schedules for large, highly constrained instances and provides strong warm starts for the CP-SAT model.
- We provide an open benchmark suite comprising 91 real-world instances from multiple sectors and 216 generated instances, created using a generator that systematically varies calendar structures, precedence patterns, resource demand types, and setup logic. We extensively evaluate all methods on these instances.

2 GPMS Framework

2.1 Informal Description and Example

The goal of GPMS is to assign jobs to machines and determine their start times such that a weighted sum of objective terms (e.g., makespan, tardiness) is minimized and all hard constraints are satisfied. Figure 1 shows a visualization of a simple example schedule.

Machine schedules and setups. The upper part of the figure shows a Gantt-style schedule for two machines. Each white rectangle corresponds to the processing of a job J_y on a machine. Gray bars represent setup activities. GPMS supports sequence-dependent setups $s_{a\to b}$ (e.g., $s_{1\to 2}$ in Figure 1) as well as sequence-independent setups s_a (e.g., s_7) and initial setups s_a^{init} (e.g., s_3^{init}). Sequence-dependent setups are essential in many industries where tooling, cleaning, or changeovers depend on the exact job sequence (e.g., color changes or machine reconfigurations). Initial setups s_a^{init} model preparation time when J_a is scheduled as the first job on a machine.

Machine calendars specify when each machine is available. Hatched rectangles in the Gantt chart indicate downtimes: during these, any setup or processing is paused and resumes when the machine becomes available again. In our example, this is the case for setup $s_{2\to 5}$ and processing of J_7. GPMS can also limit how long processing may be paused, which is useful in settings where excessive interruptions would degrade quality.

Temporal constraints. In Figure 1, arrows between jobs represent precedence arcs; the corresponding precedence graph is shown in Figure 2. For each arc, we specify a transfer fraction that determines the point during the predecessor's processing at which the lag constraints apply. For the arc $J_1 \to J_2$ the transfer fraction is 0.5, so with a processing interval of $[0, 2]$ the transfer point is at $t = 1$. Given a minimum lag of 2 and a maximum lag of 5, the setup of J_2 must start in the interval $[3, 6]$. The flag *no job in between* requires that J_2 is scheduled directly after J_1 on the same machine (pauses allowed), which is useful when a part must stay on the same fixture. Some arcs also impose conditional machine eligibility: for example, for $J_1 \to J_3$, if J_1 runs on M_1 then J_3 must run on M_2, modeling that the machine used for the predecessor restricts where the successor can be processed.

Secondary resources. The bottom part of Figure 1 shows, for the secondary resources R_1 and R_2, the resource usage (blue) and the time-varying capacities (red) over the horizon $H = [0, 11]$. GPMS supports two demand types. *Pulse* demands consume (or produce) a fixed amount of the resource throughout a setup or processing and are paused during machine downtimes, as seen for the demand of J_7 on R_1. Pulse demands can, e.g., represent personnel needed during setups or while a job is processed. In contrast, *step* demands change the resource usage at specific events such as the start/end of a setup or processing. These are useful, e.g., for inventory-like resources that behave as reservoirs: one job may increase the level by producing material, while another job later decreases it by consuming that material. In our example, the start of J_3 and the start of the setup of J_2 each increase the usage by one unit, and the start of J_4, which is a successor of both, decreases it by two units. Time-varying capacities are needed, for example, due to shift boundaries or tool availability. In the example, the pause between J_4 and J_7 occurs because the capacity of R_1 drops to zero between $t = 6$ and $t = 7$ (e.g., due to a staff break), so J_7's pulse demand cannot be satisfied during this period and its start is shifted accordingly.

Beyond what is illustrated in the example schedule, GPMS also supports release dates, due dates, deadlines, fixed timing constraints, and non-capacitated machines, which we describe formally in Section 2.2.

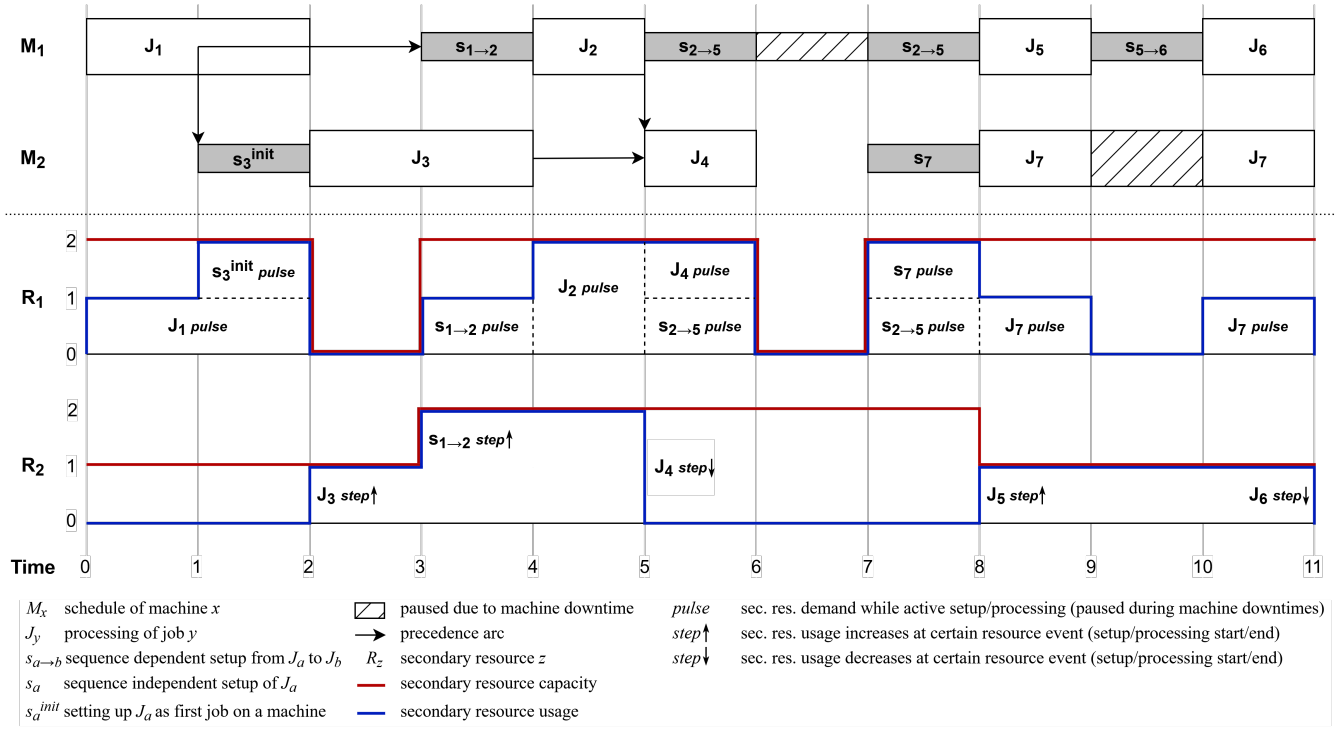

Figure 1: Example schedule and secondary resource profile chart.

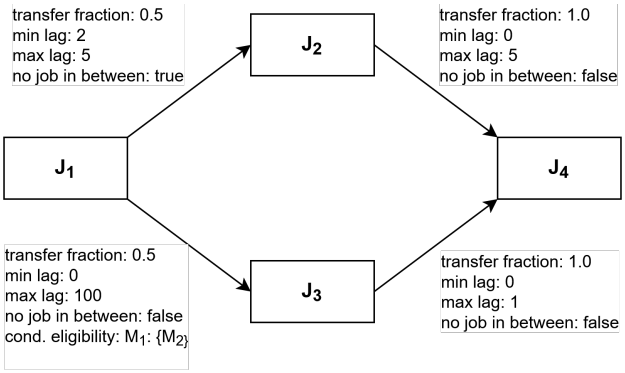

Figure 2: Example precedence graph.

2.2 Formal Problem Definition

Instance Data. We are given a set of machines $M = M_u \cup M_n$ partitioned into unit-capacity machines M_u (at most one job at a time) and non-capacitated machines M_n (allowing overlapping jobs). Each machine $m \in M$ is available over a finite scheduling horizon $H = \{0, \dots, H_{\max}\}$ and has a calendar I_m consisting of pairwise disjoint availability intervals $[s, e) \subseteq H$.

We consider a set of jobs J. Each job $j \in J$ has a non-empty set of eligible machines $E_j \subseteq M$, a release date r_j, a due date d_j, and optionally a deadline δ_j. For some jobs we may specify fixed setup- (F^{set}) and/or processing intervals (F^{proc}): for $j \in F^{\text{set}}$ we are given $[\bar{S}_j^{\text{set}}, \bar{C}_j^{\text{set}})$, and for $j \in F^{\text{proc}}$ an interval $[\bar{S}_j^{\text{proc}}, \bar{C}_j^{\text{proc}})$.

For each job–machine pair (j, m) with $m \in E_j$ we are given a processing time $p_{m,j} \geq 0$, initial and sequence-independent setup times $s^0_{m,j} \geq 0$ and $s^{\text{SI}}_{m,j} \geq 0$. J^{SD} denotes the subset of jobs with sequence-dependent setup times. For $j_2 \in J^{\text{SD}}$, we are given the sequence-dependent setup time $s^{\text{SD}}_{m,j_1,j_2} \geq 0$ when j_2 immediately follows j_1 on m. We also allow corresponding setup and processing costs $k^0_{m,j}$, $k^{\text{SI}}_{m,j}$, $k^{\text{SD}}_{m,j_1,j_2}$, and $k^{\text{proc}}_{m,j}$, and a maximum processing-span factor $\rho_{m,j} \geq 0$ limiting how much the processing of a job can be stretched due to downtime.

Secondary resources are given by a set R. For each resource $r \in R$, we consider a capacity profile on a discrete horizon $H_r = \{0, \dots, b_r\}$ with capacities $c_{r,t} \geq 0$ for $t \in H_r$, and an initial usage level u^{init}_r. For each (j, m, r) with $m \in E_j$, we may have initial setup-, setup-, and processing demands ($q^0_{m,j,r}$, $q^{\text{set}}_{m,j,r}$, and $q^{\text{proc}}_{m,j,r}$). Each demand has a type $z \in Z$, where $Z = \{\pm\text{pulse}, \pm\text{stepAtStart}, \pm\text{stepAtEnd}\}$. Pulse demands consume or produce a fixed amount while the corresponding activity is actively running and are paused during downtimes; step demands cause permanent step changes at the start or end of the activity. If a setup or processing duration is zero, pulse demands have no effect, whereas step demands still trigger at the event time.

Precedence relations form a directed acyclic graph over J. For each job j, we are given a set P_j of arcs. Each arc $p \in P_j$ is a tuple $(\text{pred}_p, \tau_p, \ell^{\min}_p, \ell^{\max}_p, \iota_p, (E^{(p)}_{m,j})_{\forall m \in E_{\text{pred}_p}})$, where $pred_p \in J$ is the predecessor, $\tau_p \in [0, 1]$ is a transfer fraction, $\ell^{\min}_p$ and $\ell^{\max}_p$ define minimum and maximum lags, $\iota_p \in \{0, 1\}$ indicates whether j must be processed imme-

diately after pred_p on the same unit-capacity machine, and $(E^{(p)}_{m,j})_{\forall m \in E_{\mathrm{pred}_p}}$ encodes conditional machine eligibilities for j depending on the machine chosen for pred_p.

Decision variables and induced timing. A schedule is defined by assigning to each job $j \in J$ a machine $a_j \in E_j$ and a setup start time $S^{\mathrm{set}}_j \in H$. Given these decisions, we order the jobs on each machine $m \in M_u$ by increasing setup start time. Assume variable first_j is true if $a_j \in M_u$ and j is first in the ordering of a_j; otherwise it is false. If first_j is false, let prev_j denote the job immediately preceding j on a_j. The actual setup time of job j on a_j is then

$$s^{\mathrm{set}}_j = \begin{cases} s^0_{a_j,j} & \mathrm{first}_j \wedge a_j \in M_u \\ s^{\mathrm{SI}}_{a_j,j} & (\neg \mathrm{first}_j \wedge j \notin J^{\mathrm{SD}}) \vee a_j \in M_n \\ s^{\mathrm{SD}}_{a_j,\mathrm{prev}_j,j} & \neg \mathrm{first}_j \wedge j \in J^{\mathrm{SD}} \wedge a_j \in M_u \end{cases}$$

and the corresponding setup cost $k^{\mathrm{set}}_{a_j,j}$ is defined analogously from $k^0_{m,j}$, $k^{\mathrm{SI}}_{m,j}$, and $k^{\mathrm{SD}}_{m,j_1,j_2}$. Starting at time S^{set}_j, the setup of job j on a_j accumulates active time only while a_j is available (i.e., inside the intervals of I_{a_j}) and is paused during downtimes; we denote by C^{set}_j the time at which the accumulated active setup time reaches s^{set}_j. Processing then starts at time $S^{\mathrm{proc}}_j = C^{\mathrm{set}}_j$ if C^{set}_j lies inside an availability interval (i.e., does not coincide with the end of an availability interval). Otherwise S^{proc}_j equals the start of the next availability interval of a_j. Processing evolves in the same way: starting at S^{proc}_j, it accumulates active time only while a_j is available, is paused during downtimes, and completes at the time C_j when the accumulated active processing time reaches $p_{a_j,j}$.

Constraints. A schedule $(a_j, S^{\mathrm{set}}_j)_{j \in J}$ is feasible if the following hold:

(C1) Calendars and machine capacity. Setup and processing of each job j on machine a_j may only occur inside availability intervals of I_{a_j}; activities are paused during downtimes as described above. When we say that setup or processing of job j on machine a_j occurs inside the availability intervals of a_j, we mean that whenever j is being set up or processed on a_j at time t, there exists an interval $[s, e) \in I_{a_j}$ with $s \le t < e$. All non-zero setup and processing durations must satisfy this condition. On unit-capacity machines $m \in M_u$, setup and processing intervals of different jobs must not overlap; on $m \in M_n$, overlaps are allowed. The effective processing span of j, i.e., the elapsed time from S^{proc}_j to C_j including pauses, must not exceed $(1 + \rho_{a_j,j})p_{a_j,j}$.

(C2) Release dates, due dates, deadlines, and fixed timings. For all $j \in J$, $S^{\mathrm{set}}_j \ge r_j$. If j has a deadline δ_j, then $C_j \le \delta_j$. For $j \in F^{\mathrm{set}}$, $[S^{\mathrm{set}}_j, C^{\mathrm{set}}_j) = [\bar{S}^{\mathrm{set}}_j, \bar{C}^{\mathrm{set}}_j)$, and for $j \in F^{\mathrm{proc}}$, $[S^{\mathrm{proc}}_j, C_j) = [\bar{S}^{\mathrm{proc}}_j, \bar{C}^{\mathrm{proc}}_j)$.

(C3) Precedences. For each $p \in P_j$ with predecessor pred_p, let $T_{j,p}$ be the time at which the accumulated active processing time of pred_p on its assigned machine reaches $\tau_p\,(p_{a_{\mathrm{pred}_p},\mathrm{pred}_p})$. Then $T_{j,p} + \ell^{\min}_p \le S^{\mathrm{set}}_j \le T_{j,p} + \ell^{\max}_p$. If $\iota_p = 1$, then $a_j = a_{\mathrm{pred}_p}$; moreover, if $a_j \in M_u$, then $\mathrm{pred}_p = \mathrm{prev}_j$ (i.e., no job in between). Conditional eligibility requires $a_j \in E^{(p)}_{a_{\mathrm{pred}_p},j}$.

(C4) Secondary resources. For each resource $r \in R$ and time $t \in H_r$, we denote by $u_{r,t}$ the usage induced by all setup and processing activities according to their demands and types. For a job j, the initial-setup demand $q^0_{a_j,j,r}$ is only relevant if j is first on $a_j \in M_u$; otherwise the setup demand $q^{\mathrm{set}}_{a_j,j,r}$ is used. For all $r \in R$ and $t \in H_r$, $0 \le u_{r,t} \le c_{r,t}$. Section 3 specifies the event-based construction of $u_{r,t}$.

Objectives. GPMS supports a finite set O of objective components. Each $o \in O$ is a function $f_o(\sigma)$ of a schedule σ combined with a nonnegative weight $w_o \ge 0$; some components aggregate over jobs and use additional per-job weights $w_{o,j} \ge 0$. In this paper we use the following components: Total weighted tardiness (TWT) is $f_{\mathrm{TWT}}(\sigma) = \sum_{j \in J} w_{\mathrm{TWT},j} \max\{0, C_j - d_j\}$. The makespan (Cmax) is $f_{\mathrm{Cmax}}(\sigma) = \max_{j \in J} C_j$. The sum of machines' makespans (MSsum) is $f_{\mathrm{MSsum}}(\sigma) = \sum_{m \in M} \max_{j \in J: a_j = m} C_j$. Total setup time (ST) is $f_{\mathrm{ST}}(\sigma) = \sum_{j \in J} s^{\mathrm{set}}_j$. Total setup and processing cost (K) is $f_{\mathrm{K}}(\sigma) = \sum_{j \in J}(k^{\mathrm{set}}_{a_j,j} + k^{\mathrm{proc}}_{a_j,j})$. The overall objective is to minimize $\sum_{o \in O} w_o f_o(\sigma)$, where the weights w_o trade off different components and the per-job weights $w_{o,j}$ capture the relative importance of individual jobs within job-aggregating components such as total weighted tardiness.

3 Modeling and Solution Approaches

3.1 Solver-Agnostic MiniZinc Model

The MiniZinc (Nethercote et al. 2007) model we propose follows the formal GPMS semantics, but pushes parts of the calendar and resource logic into preprocessing. We represent each machine calendar as a sequence of availability periods; the gaps between periods are the downtimes. For each machine m and period i, we precompute the cumulative downtime $D_{m,i}$, i.e., the total length of all downtimes of m strictly before period i. For each job–machine pair, we use membership indicators (e.g., $ssIn_{j,m,i}$, $seIn_{j,m,i}$) with exactly-one constraints that indicate in which periods setup and processing start and end lie. If i^{ss}_j and i^{se}_j are the selected periods, the setup completion time is

$$C^{\mathrm{set}}_j = S^{\mathrm{set}}_j + s^{\mathrm{set}}_j + \big(D_{a_j,i^{\mathrm{se}}_j} - D_{a_j,i^{\mathrm{ss}}_j}\big),$$

and the processing completion time C_j is defined analogously from the corresponding memberships. Instead of introducing Booleans for each downtime gap and summing their lengths, we express the downtime as the difference of two cumulative values, yielding a compact and efficient calendar encoding.

Precedences with transfer fractions reuse the same mechanism as completion times. For each arc $p \in P_j$, we introduce a variable $T_{j,p}$ and membership indicators $tpIn_{j,p,m,i}$ that select the period i of the predecessor's machine m containing the transfer point. We then compute $T_{j,p}$ like a completion time on that machine, but with processing time scaled by τ_p, i.e., from $S^{\mathrm{proc}}_{\mathrm{pred}_p}$ plus $\tau_p\, p_{a_{\mathrm{pred}_p},\mathrm{pred}_p}$ and plus the corresponding calendar downtime. The lag constraints are then: $T_{j,p} + \ell^{\min}_p \le S^{\mathrm{set}}_j \le T_{j,p} + \ell^{\max}_p$.

On unit-capacity machines we encode the sequence via a predecessor index variable $prevJob_j$ and a Boolean $isFirst_j$.

An `alldifferent` constraint on $prevJob_j$ per machine plus $prevJob_j = 0$ for first jobs yields a predecessor chain; non-capacitated machines use $prevJob_j = 0$ and ignore this structure. These variables are then used for determining sequence-dependent setup times and for the "no job in between" precedence constraint.

Secondary resources. Secondary resources are handled by two MiniZinc encodings, depending on whether there exist only demands with positive demand types ($\{+\text{pulse}, +\text{stepAtStart}, +\text{stepAtEnd}\}$) or also negative ones ($\{-\text{pulse}, -\text{stepAtStart}, -\text{stepAtEnd}\}$). For each resource r, we compute $B_r = \max_{t \in H_r} c_{r,t}$ and emulate time-varying capacities with dummy "calendar" events that block $B_r - c_{r,t}$ units at each $t \in H_r$; initial usage u_r^{init} is a single blocking task over H_r of height u_r^{init}. For the positive-only case, all resource events (pulse or step demands induced by setup and processing), calendar blocks, and the initial block are concatenated into arrays (S, D, R) of event start times S, durations D, and resource demands R, and are enforced by a single `cumulative(S,D,R,`B_r`)`. Pulse demands are intersected with machine-availability periods, so that resource usage is applied only while the machine is available and paused during downtimes. In contrast, step demands are modeled as events starting at the setup or processing start or end (depending on the type) and running to the end of H_r, so that the cumulative load at any time reflects all past step changes without introducing explicit time-indexed resource variables. When negative demand types are present, we instead use two `cumulatives` with signed resource demands. The first ("upper bound") constraint contains all resource events plus calendar and initial-usage blocks and enforces that the resulting profile never exceeds B_r. The second ("lower bound") constraint contains only the real resource events (no blocks, since these represent capacity reductions rather than actual usage increases) and enforces that the profile never drops below 0. Together, these two constraints bound the implicit resource level between 0 and the time-varying capacity without introducing time-indexed variables $u_{r,t}$.

3.2 Interval-Variable (CP-SAT) Model

The interval-variable model, implemented with the CP-SAT solver from OR-Tools (Perron, Didier, and Gay 2023), replaces the predecessor-chain encoding by setup and processing interval variables combined with global constraints. Each job has setup and processing intervals, and on unit-capacity machines we post a single `NoOverlap` per machine instead of a per-machine `alldifferent` over predecessor indices, so ordering and non-overlap are handled directly by one global constraint. The order on each unit-capacity machine is captured by a `Circuit` constraint (Vismara and Briot 2018) over a dummy node and the job nodes, where an arc $i \to j$ is active iff job i is immediately followed by j on that machine (the dummy node represents the start/end of the sequence or an unused machine). Sequence-dependent setup times and costs are attached to these arcs by setting the setup duration of j to $s^{\text{SD}}_{i,j,m}$ only when arc $i \to j$ is chosen. This removes the `prevJob` variables and the associated constraints of the MiniZinc model, and lets CP-SAT reason on a routing-style graph together with the `NoOverlap` constraint.

Secondary resources. For secondary resources the interval-variable model mirrors the MiniZinc encoding. With positive demand types only, we post one `Cumulative` (Aggoun and Beldiceanu 1993) per resource, using dummy intervals to represent time-varying capacities and initial usage, and intersect pulse demands with machine-availability periods while modeling step demands as intervals from the event to the end of H_r. With negative demand types, we switch to two `Reservoir` constraints (Schaus, Thomas, and Kameugne 2025) per resource and add signed events (consumption and production) at setup/processing start/end, enforcing both an upper capacity and a nonnegativity bound. The resulting resource profiles are therefore semantically identical to those in the MiniZinc model.

Algorithm 1: Construction heuristic (high-level sketch)

```
initialize schedule state S and machine tails T_m
while unscheduled jobs remain do
    determine sliding window [t_min, t_min + W)
    J ← eligible jobs (all predecessors done, release in window)
    C ← all feasible (j ∈ J, m ∈ E_j) pairs via dispatcher
    if C = ∅ then
        advance t_min to next release date; continue
    end if
    score candidates in C;
    randomly select a (j, m) pair from top-k
    schedule job j on machine m; update S and T_m
end while
```

3.3 Construction Heuristic

We propose a randomized construction heuristic (CH) that incrementally builds a feasible GPMS schedule by repeatedly selecting a machine-job pair based on a scoring function. The chosen job is then appended to the tail of the chosen machine. Feasibility of a job assignment is determined by a procedural dispatcher that computes the earliest feasible setup and processing interval by first determining the earliest setup start from release dates, machine availability, and precedence constraints, then calculating end times accounting for machine downtimes, and finally verifying max-lag deadlines, span limits, and secondary-resource capacities. When a constraint is violated, a jump-ahead strategy advances to the next feasible time point (e.g., a capacity period boundary) rather than incrementing by one.

Main loop. Algorithm 1 sketches the procedure. At each iteration, the CH (i) identifies jobs whose predecessors are completed; (ii) restricts them to those released within a sliding time window of size W; (iii) evaluates all feasible machine–job pairs using the dispatcher; (iv) scores the candidates and randomly selects one from the top-k. If no feasible pair exists within the window, we advance the window to start at the next release date outside its current range.

Scoring. The scoring function combines five normalized, weighted components: (i) earliest completion time relative

to other candidates, (ii) sequence-dependent setup efficiency given the current predecessor, (iii) a tardiness-urgency term based on Lee and Pinedo (1997) penalizing jobs close to or past their due date, (iv) precedence urgency prioritizing predecessors of jobs with max-lag or no-job-in-between constraints, and (v) the slack to each job's *latest-setup-start (LSS)* time, i.e., the latest time at which its setup may begin without violating fixed timings, precedence constraints, or resource capacities.

Randomized selection from the top-k candidates not only yields diverse schedules but is also crucial for finding feasible solutions on highly constrained instances. Max lag constraints with multiple predecessors and successors, fixed timings, deadlines, and step demands effectively impose LSS times for several jobs. Since the CH does not backtrack, a single run can result in infeasibility due to conflicting LSS times; multiple randomized runs are therefore often needed to explore alternative choices and obtain a feasible schedule, which can then serve as a warm start for the CP model.

4 Experimental Evaluation

4.1 Experimental Setup

All experiments were conducted on a machine with two Intel Xeon Silver 4314 CPUs (16 cores each at 2.40 GHz) and Ubuntu 22.04. We set the memory limit to 144 GB, and each solver was allowed to use 8 threads. The CPU frequency was fixed to 2.40 GHz, and the initial random seed was set to 42. All experiments were conducted using MiniZinc version 2.9.4 and the OR-Tools CP-SAT solver version 9.14.

Real-world Instances. We evaluate our methods on 91 industrial instances collected from three sectors, provided by our industrial partner MCP GmbH. The first set comprises 77 instances from the packaging industry, each with 4 parallel machines and between 6 and 2142 jobs. These instances combine parallel machine scheduling with cumulative resource constraints (up to 27 secondary resources with pulse demands), simple precedences without min/max lags or conditional eligibilities, machine calendars, sequence-dependent setup times, release and due dates, and machine-dependent processing times. The second set contains 11 instances from food and beverage production with 21 to 1167 jobs and 2 to 10 machines. They feature machine calendars, sequence-dependent setup times, release and due dates, and machine-dependent processing times, but no precedences or secondary resources. The remaining 3 instances stem from chemical and electrical manufacturing, with 125 to 3061 jobs and 2 to 13 machines, and include the same constraint types as set 2. All real-world instances use unit-capacity machines only. While these datasets cover only a subset of the constraints supported by GPMS, the framework is derived from a broader set of industrial use cases provided by our industrial partner. In particular, all constraint types supported by GPMS arise in practice in at least one application domain; the real-world instances used here cover those parts of the framework for which we can currently obtain and publish data, and our generator systematically explores the remaining combinations.

For a few real-world instances, some published machine calendars were too short to accommodate all jobs; in these cases we extended the scheduling horizon by appending an additional availability interval starting at H_{max}.

Generated instances. We complement the industrial data with 216 synthetic instances created by an instance generator specifically designed for the GPMS framework. We vary job counts $|J| \in \{10, 20, 50, 100\}$, machine counts $|M| \in \{1, 5, 10\}$, precedence mode (none, simple chains without max lag and conditional eligibilities, or complex with max lags, "no job in between" and conditional eligibilities), secondary resource mode (none, simple pulse demands, or complex mixes of pulse, step, and negative-step demands), and machine period lengths (short vs. long calendars). For each combination of these factors we generate one instance, yielding 216 instances.

The generator samples machine calendars (synchronized, random, and no-pause patterns), machine eligibility, setup and processing times, and marks a fraction of jobs as sequence-dependent with machine- and sequence-dependent setup times. In three-quarters of instances all machines are unit-capacity; in the rest, one machine is non-capacitated. Precedence chains grow with $|J|$, with time lags drawn from size-dependent ranges. A reference schedule is constructed and used to derive release dates, due dates, fixed timings, and deadlines for a subset of jobs. Secondary resources and capacity calendars are generated from induced usage profiles, with step and negative-step demands added in the complex mode. Per-job objective weights are sampled for each component.

Objectives. All objective components and weights were selected together with our industrial partner to reflect operational priorities in the respective production environments. We do not normalize components, as this would distort these priorities.

For real-world instance sets 1 and 3, we use the following objective function:

$$10\, f_{\mathrm{TWT}}(\sigma) + 5\, f_{\mathrm{ST}}(\sigma) + f_{\mathrm{MSsum}}(\sigma) + f_{\mathrm{Cmax}}(\sigma).$$

For real-world instance set 2, we use

$$100\, f_{\mathrm{TWT}}(\sigma) + f_{\mathrm{ST}}(\sigma) + f_{\mathrm{K}}(\sigma) + 50\, f_{\mathrm{MSsum}}(\sigma) + f_{\mathrm{Cmax}}(\sigma).$$

For the generated instances, the objective is

$$10\, f_{\mathrm{TWT}}(\sigma) + 5\, f_{\mathrm{ST}}(\sigma) + f_{\mathrm{K}}(\sigma) + f_{\mathrm{MSsum}}(\sigma) + f_{\mathrm{Cmax}}(\sigma).$$

Solver Configurations. We benchmark four configurations. (1) A randomized construction heuristic (CH) that runs for up to 3600 s or 10,000 schedules, whichever occurs first. It uses a two-phase strategy: a deterministic sweep over predefined weight combinations for the scoring function (weights drawn from $\{0, 0.2, \ldots, 1.0\}$ summing to 1) with $k = 1$, followed by randomized sampling of weights, k, and window size W. (2) The interval-variable CP-SAT model with 8 threads, using the CH for at most 60 s to obtain a feasible schedule for horizon trimming. CP-SAT then solves for 3600 s. We trim all machine and resource calendars to a horizon of 1.2 times the schedule's makespan to reduce the model size, using the factor 1.2 as a modest safety margin, although this may

still exclude some optimal solutions. (3) The same interval-variable model with a warm start: CH runs for 180 s, the best schedule is used as warm start (i.e., solution hints for all decision variables) and for trimming, and CP-SAT solves for 3480 s. (4) The MiniZinc model solved by CP-SAT with identical resource limits and trimming strategy as configuration 2. For all configurations, model creation time is included in the reported solving time. All solutions are validated by a dedicated solution validator.

Preliminary experiments with Gurobi and Chuffed as MiniZinc backends showed consistently weaker performance across all metrics; we therefore omit them. This is consistent with prior findings that MILP-based backends are less competitive on rich scheduling models with calendars and sequence-dependent setups (Oujana et al. 2023; Laborie 2018).

4.2 Results and Discussion

We compare four solving approaches: the construction heuristic (CH), the interval-variable model without warm start (IV_c) and with warm start (IV_w), as well as the MiniZinc model (MZn). In all tables, #Solv is the number of instances for which a configuration finds a feasible schedule, #Best is the number of instances where its best objective equals the best objective found by any configuration (ties included), and #Opt is the number of instances where the configuration proves optimality (i.e., best objective = best lower bound). In the anytime plots (Figs. 3–4), the metric "Avg. gap to best known (%)" denotes the average, over all instances, of the relative gap between the best objective value obtained at time t and the best known objective for that instance across all solvers. Formally, for instance i with best-known objective BK_i and current objective $\mathrm{OV}_{i,t}$, the gap is $|\mathrm{OV}_{i,t} - \mathrm{BK}_i| / \mathrm{OV}_{i,t}$; if no objective is available at time t, we assign a gap of 100%. In Fig. 5, the x-axis shows the final optimality gap at the time limit, computed as $(\mathrm{OV}_i - \mathrm{LB}_i)/\mathrm{OV}_i$ from the best objective OV_i and best lower bound LB_i of the respective configuration.

Solver	#Solv	#Best	#Opt
CH	**91**	5	0
IV_c	83	59	**14**
IV_w	89	**67**	**14**
MZn	64	31	9

Table 1: Overall results on 91 real-world GPMS instances.

$\lvert J\rvert$	#Inst	CH	IV_c	IV_w	MZn
		#Solv / #Best	#Solv / #Best / #Opt		
≤ 50	15	15 / 2	15 / 15 / 11	15 / 15 / 10	15 / 15 / 6
51–100	33	33 / 0	33 / 25 / 1	33 / 25 / 2	30 / 14 / 1
101–200	26	26 / 0	25 / 13 / 2	26 / 18 / 2	17 / 2 / 2
> 200	17	17 / 3	10 / 6 / 0	15 / 9 / 0	2 / 0 / 0

Table 2: Results on real instances grouped by job-count bins.

Table 1 shows that the CH finds feasible schedules for all real-world instances, IV_w solves 89 and attains 67 best-known objectives with 14 proven optima. IV_c and MZn attain fewer best solutions, but still prove 14 and 9 optima, respectively. The real-world job-count breakdown (Table 2) shows that IV_w is also able to find solutions for larger instances.

Solver	#Solv	#Best	#Opt
CH	**209**	41	0
IV_c	178	137	110
IV_w	198	**162**	**113**
MZn	170	121	103

Table 3: Overall results on 216 generated GPMS instances.

On the 216 generated instances, IV_w again produced the best results with 198 feasible, 162 best-known, and 113 optimal solutions (Table 3), followed by IV_c and MZn. CH solves 209 instances and reaches the best objective in 41 cases. The generated-instance breakdown by job count (Table 4) shows that performance degrades considerably with $|J| \geq 50$: all configurations handle 10–20 jobs well, but feasibility and optimality rates drop sharply for 50 and 100 jobs. For $|J| = 10$, IV_w proves optimality on all 54 instances but is counted as best on only 51, because horizon trimming with the warm-start schedule may exclude solutions with a better objective but longer makespan. The secondary-resource breakdown (Table 5) shows that feasibility becomes harder once secondary-resource calendars are used and particularly difficult in the complex mode with (negative) step demands, where two reservoir constraints per resource and many step-like resource events lead to very large models.

$\lvert J\rvert$	#Inst	CH	IV_c	IV_w	MZn
		#Solv / #Best	#Solv / #Best / #Opt		
10	54	54 / 14	54 / 54 / 54	54 / 51 / 54	54 / 54 / 54
20	54	54 / 4	54 / 52 / 49	54 / 51 / 51	52 / 51 / 45
50	54	51 / 9	41 / 20 / 4	48 / 30 / 5	39 / 13 / 2
100	54	50 / 14	29 / 11 / 3	42 / 30 / 3	25 / 3 / 2

Table 4: Results on generated instances grouped by job count.

Sec. Res.	#Inst	CH	IV_c	IV_w	MZn
		#Solv / #Best	#Solv / #Best / #Opt		
none	72	70 / 6	70 / 53 / 36	70 / 56 / 36	65 / 40 / 36
simple	72	70 / 11	60 / 43 / 40	70 / 59 / 40	61 / 45 / 39
complex	72	69 / 24	48 / 41 / 34	58 / 47 / 37	44 / 36 / 28

Table 5: Results on generated instances grouped by secondary-resource mode.

The anytime behavior visualized in Fig. 3 shows that IV_w and IV_c rapidly reduce their gaps on the real-world instances, and IV_w drops even below 5%. The remaining distance between the two curves is mainly due to unresolved instances in IV_c, which contribute 100% gaps. For the CH, the plots show the gap of its overall best objective as a constant reference

line rather than a true anytime curve, achieving an average gap below 10%. MZn remains above a 30% average gap. On the generated instances (Fig. 4), the same pattern holds with slightly higher gaps: IV_w converges fastest, followed by IV_c and MZn. MZn's weaker performance on generated instances is partly due to MiniZinc flattening, which for complex instances with secondary resources can exhaust the time limit before solving starts. CH achieves an average gap below 20%.

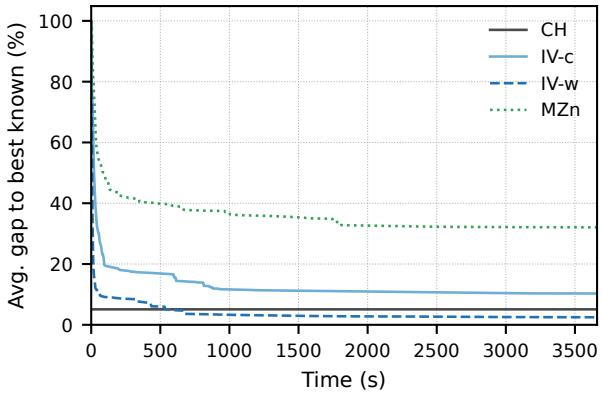

Figure 3: Anytime gap to best known objective on real-world instances.

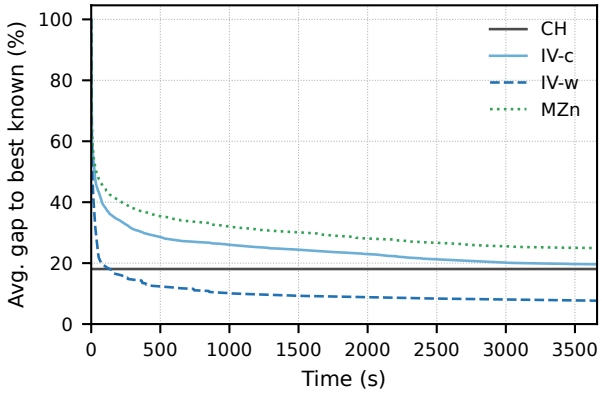

Figure 4: Anytime gap to best known objective on generated instances.

Figure 5 reports the final optimality-gap distributions. On the real-world instances, IV_w and IV_c achieve near-zero gaps for around 60% of the cases and gaps below 10% for roughly 80%; the remaining 20% correspond mainly to the largest instances. IV_w produces improved results compared to IV_c and MZn in our experiments. On the generated instances, gap quality decreases with instance size and constraint complexity: about half of the instances (mostly with 10 or 20 jobs) are solved to proven optimality, another ∼30% reach final gaps below 50%, and the remainder exhibit larger gaps. IV_w yields better average optimality gaps than IV_c and MZn.

Several patterns emerge from the experiments. The CH finds feasible, high-quality schedules efficiently on instances without hard timing constraints (max lags, deadlines, fixed

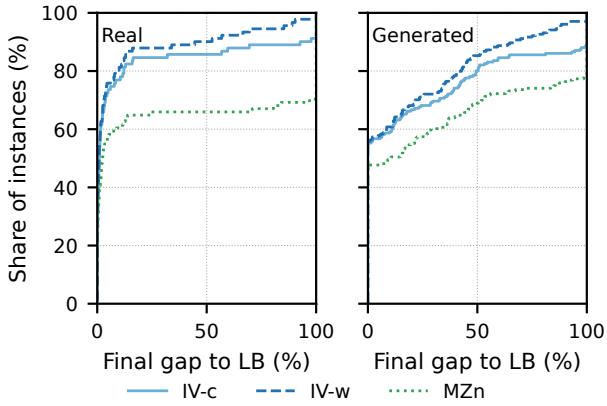

Figure 5: Final optimality gap distribution after 3600 s. Left: real instances; right: generated instances.

timings, step demands), but once such constraints are present, multiple randomized runs are needed to find a feasible solution. For the exact methods, job count is the strongest predictor of difficulty, but complex constraints like step demands quickly increase model creation time and size. Among objective components, tardiness is the hardest to bound tightly, whereas makespan and setup time yield tighter lower bounds.

The CH is lightweight in memory (under 2.5 GB on average), whereas the CP models require considerably more memory (around 8 GB on average for real, around 12 GB for generated instances). Across all CP configurations, 2 out-of-memory cases (exceeding the 144 GB limit) occurred on real-world and 26 on generated instances. On average, IV_w is the most effective exact approach. However, no configuration dominates universally: IV_c and MZn occasionally achieve faster optimality proofs or smaller gaps, and IV_w's 180 s warm-start investment only pays off for instances with at least 20 jobs. CH is essential for feasibility and warm starts, while MZn serves as a useful reference formulation.

5 Conclusion

We introduced GPMS, a generalized parallel machine scheduling framework that unifies rich temporal and secondary-resource constraints in a single model. We formalized the problem, proposed a solver-independent MiniZinc model and an interval-variable CP-SAT model, and introduced a novel, randomized construction heuristic that can handle large, highly constrained instances. Our experiments on 91 industrial and 216 generated instances show that the interval-variable model with warm starts is the most effective exact approach on the evaluated instances, while the construction heuristic is very robust for finding feasible schedules and MiniZinc serves as a clear reference formulation. Our results show that the proposed approaches provide high-quality results for all of the evaluated real-world instances.

Future work could investigate the hybridization of the proposed exact techniques with heuristic approaches in the framework of large neighborhood search to further improve results on large-scale instances.

Acknowledgements

This work was supported by MCP GmbH.

References

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

Aggoun, A.; and Beldiceanu, N. 1993. Extending chip in order to solve complex scheduling and placement problems. *Mathematical and Computer Modelling*, 17(7): 57–73.

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

Brucker, P.; Drexl, A.; Möhring, R.; Neumann, K.; and Pesch, E. 1999. Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, 112(1): 3–41.

Dang, Q.-V.; van Diessen, T.; Martagan, T.; and Adan, I. 2021. A matheuristic for parallel machine scheduling with tool replacements. *European Journal of Operational Research*, 291(2): 640–660.

Dauzère-Pérès, S.; Ding, J.; Shen, L.; and Tamssaouet, K. 2024. The flexible job shop scheduling problem: A review. *European Journal of Operational Research*, 314(2): 409–432.

De Reyck, B.; and Herroelen, W. 1999. The multi-mode resource-constrained project scheduling problem with generalized precedence relations. *European Journal of Operational Research*, 119(2): 538–556.

Durasević, M.; and Jakobović, D. 2023. Heuristic and metaheuristic methods for the parallel unrelated machines scheduling problem: a survey. *Artificial Intelligence Review*, 56: 3181–3289.

Edis, E. B.; Oguz, C.; and Ozkarahan, I. 2013. Parallel machine scheduling with additional resources: Notation, classification, models and solution methods. *European Journal of Operational Research*, 230(3): 449–463.

Einspieler, C.; Horn, M.; Lackner, M.-L.; Malik, P.; Musliu, N.; and Winter, F. 2025. Solving Parallel Machine Scheduling With Precedences and Cumulative Resource Constraints With Calendars. *arXiv preprint arXiv:2512.11864*.

Gacias, B.; Artigues, C.; and Lopez, P. 2010. Parallel machine scheduling with precedence constraints and setup times. *Computers & Operations Research*, 37(12): 2141–2151.

Hartmann, S.; and Briskorn, D. 2022. An updated survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research*, 297(1): 1–14.

Kasapidis, G. A.; Paraskevopoulos, D. C.; Mourtos, I.; and Repoussis, P. P. 2025. A unified solution framework for flexible job shop scheduling problems with multiple resource constraints. *European Journal of Operational Research*, 320(3): 479–495.

Kasapidis, G. A.; Paraskevopoulos, D. C.; Repoussis, P. P.; and Tarantilis, C. D. 2021. Flexible Job Shop Scheduling Problems with Arbitrary Precedence Graphs. *Production and Operations Management*, 30(11): 4044–4068.

Kreter, S.; Rieck, J.; and Zimmermann, J. 2016. Models and solution procedures for the resource-constrained project scheduling problem with general temporal constraints and calendars. *European Journal of Operational Research*, 251(2): 387–403.

Laborie, P. 2018. An Update on the Comparison of MIP, CP and Hybrid Approaches for Mixed Resource Allocation and Scheduling. In van Hoeve, W., ed., *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, volume 10848 of *Lecture Notes in Computer Science*, 403–411. Springer.

Lee, Y. H.; and Pinedo, M. 1997. Scheduling jobs on parallel machines with sequence-dependent setup times. *European Journal of Operational Research*, 100(3): 464–474.

Moser, M.; Musliu, N.; Schaerf, A.; and Winter, F. 2022. Exact and metaheuristic approaches for unrelated parallel machine scheduling. *Journal of Scheduling*, 25(5): 507–534.

Nethercote, N.; Stuckey, P. J.; Becket, R.; Brand, S.; Duck, G. J.; and Tack, G. 2007. MiniZinc: Towards a Standard CP Modelling Language. In Bessière, C., ed., *Principles and Practice of Constraint Programming – CP 2007*, volume 4741 of *Lecture Notes in Computer Science*, 529–543. Berlin, Heidelberg: Springer Berlin Heidelberg.

Oujana, S.; Amodeo, L.; Yalaoui, F.; and Brodart, D. 2023. Mixed-Integer Linear Programming, Constraint Programming and a Novel Dedicated Heuristic for Production Scheduling in a Packaging Plant. *Applied Sciences*, 13(10).

Perron, L.; Didier, F.; and Gay, S. 2023. The CP-SAT-LP Solver. In Yap, R. H. C., ed., *29th International Conference on Principles and Practice of Constraint Programming (CP 2023)*, volume 280 of *Leibniz International Proceedings in Informatics (LIPIcs)*, 3:1–3:2. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

Santoro, M. C.; and Junqueira, L. 2023. Unrelated parallel machine scheduling models with machine availability and eligibility constraints. *Computers & Industrial Engineering*, 179: 109219.

Schaus, P.; Thomas, C.; and Kameugne, R. 2025. Implementing Cumulative Functions with Generalized Cumulative Constraints. *arXiv preprint arXiv:2508.01751*.

Schlenkrich, M.; and Parragh, S. N. 2023. Solving large scale industrial production scheduling problems with complex constraints: an overview of the state-of-the-art. *Procedia Computer Science*, 217: 1028–1037.

Shen, L.; Dauzère-Pérès, S.; and Neufeld, J. S. 2018. Solving the flexible job shop scheduling problem with sequence-dependent setup times. *European Journal of Operational Research*, 265(2): 503–516.

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

Vismara, P.; and Briot, N. 2018. A Circuit Constraint for Multiple Tours Problems. In Hooker, J., ed., *Principles and Practice of Constraint Programming*, volume 11008 of *Lecture Notes in Computer Science*, 389–402. Cham: Springer International Publishing.

Weglarz, J.; Józefowska, J.; Mika, M.; and Waligóra, G. 2011. Project scheduling with finite or infinite number of activity processing modes – A survey. *European Journal of Operational Research*, 208(3): 177–205.

Yunusoglu, P.; and Topaloglu Yildiz, S. 2022. Constraint programming approach for multi-resource-constrained unrelated parallel machine scheduling problem with sequence-dependent setup times. *International Journal of Production Research*, 60(7): 2212–2229.