Constraint Programming for Mixed-Model Assembly Line Scheduling with Complex Industrial Constraints

Guillaume Povéda
Airbus SAS, Toulouse, France

Javier Buil Tejero
ALTEN GmbH, Manching, Germany

Tamara Borreguero Sanchidrian
Airbus DS, Getafe, Spain

Abstract

The aeronautical industry transitioned in the 90s to takt-paced, product-specific assembly lines. The current trend of increased customization and demand variability is pushing for a transition to flexible mixed-model assembly lines. We address a mid-term planning problem for an airframe assembly plant, modeled as a Resource-Constrained Project Scheduling Problem (RCPSP) with complex industrial constraints. This formulation serves a dual purpose: facilitating high-level production planning and validating plant designs, particularly during ramp-up scenarios. We specifically tackle challenges involving calendar-based preemption, variable resource capacity, and resource blocking between task groups. We propose a Constraint Programming (CP) formulation that optimizes conflicting objectives, including Tardiness and Just-in-Time costs. To ensure scalability for large industrial instances, we introduce a sequential solving method based on topological decomposition. The proposed approach proves effective in handling complex scenarios, acting as a foundation for more realistic models.

2012 ACM Subject Classification Computing methodologies → Planning and scheduling; Theory of computation → Constraint and logic programming; Applied computing → Industry and manufacturing

Keywords and phrases modeling, scheduling, heuristics, multi-objective

Digital Object Identifier 10.4230/LIPIcs.CP.2026.46

Supplementary Material *Software*: `https://github.com/airbus/discrete-optimization/`

Funding Our work has benefitted from the AI Interdisciplinary Institute ANITI. ANITI is funded by the French "Investing for the Future – PIA3" program under the Grant agreement n°ANR-23-IACL-0002.

1 Introduction

1.1 Industrial context

This paper focuses on a mixed-model assembly line scheduling problem. Unlike paced lines where products move sequentially through a fixed series of stations, this use case involves sequencing multiple distinct product instances across a graph of production stations with an arbitrary directed acyclic graph layout. The flexibility of the system comes from its graph layout and its ability to handle heterogeneous product flows simultaneously.

In this environment without production pace (also called *pulse*); the optimal rhythm for each station must emerge from the scheduling decisions. Tasks are pre-allocated to stations, which can be seen as disjunctive resources with specific availability calendars. A critical constraint is that once a product enters a station, it uses that resource exclusively until all

assigned tasks of this product to this station are completed, preventing any interleaving of products. Execution requires a workforce with diverse skills, modeled as cumulative resource pools with independent availability calendars. Additionally, material resources such as jigs or transport equipment are required to move products between locations. The goal is to generate an efficient high-level production schedule for a mid-to-long-term horizon (typically 6 months to 1 year), managing approximately 50 products to output across ~30 stations and ~10 shared resource pools.

1.2 Operational challenges

This application presents a combination of scale and complexity. The problem involves scheduling hundreds of tasks over a long horizon, subject to calendar and resource blocking constraints that make it substantially harder than the classical RCPSP. The two main additional constraints coming from the use case is that the tasks can span over resource unavailability periods, and that some resources are blocked between group of tasks, representing physical usage of some parts of the factory.

Planners must satisfy strict release dates and soft deadlines to meet delivery targets (minimizing tardiness). At the same time, they must adhere to "Just-in-Time" (JIT) principles, leading to the minimization of early task completion to reduce Work-in-Progress (WIP). This creates a difficult trade-off where solutions must balance delivery reliability against efficient inventory management.

An optimization tool for this use case would have a dual purpose: in an operational context it requires agility to update schedules in response to supply chain delays or task overruns. In a design context, it acts as a validation engine: planners could use the solver to stress-test proposed factory layouts (e.g., station dependencies or capacities) under ramp-up scenarios. Consequently, the solver must be robust enough to identify bottlenecks in the facility design itself, and be fast enough to possibly test many different design and ramp-up scenarios.

2 Related Work

2.1 Line Balancing

Line balancing problems are focused in allocating set of tasks to stations. It maps precedence relation between tasks to precedence in station allocation. Each station has typically a cycle-time that limits the amount of tasks that could be assigned to it. The final goal of line balancing is to build a schedule that repeats itself for each product that will go through the production stations. The foundation works looking at assembly line problems Simple Assembly Line Balancing Problem (SALBP), classified extensively by Baybars [2] and later by Scholl and Becker [26]. SALBP-1 problem assumes fixed cycle-time and minimize the number of station whereas SALBP-2 minimizes cycle-time, given a number of stations. More advanced line balancing generalized the SALBP to increase its realism, Boysen et al. [4] highlight the shift towards Generalized Assembly Line Balancing (GALBP) to handle mixed-model sequencing and variable task times. Moreover, in modern assembly contexts, production line is no longer a rigid sequence of stations. This leads to more flexible line balancing problem with custom stations layout, for e.g in [5] in which there are different technological modes to accomplish the tasks, that can follow different alternative subpaths in an assembly layout.

2.2 Line Balancing and Scheduling

The detailed schedule of tasks in each station is not provided with a SALBP abstraction. Where detailed scheduling should be computed in each station, a scheduling approach is needed. Combining balancing and scheduling is done in [18] where task are scheduled taken into account precedence and equipment constraints. Recently, [23] solved a balancing/scheduling problem considering parallel stations with mobile workers using CP, demonstrating that resource mobility transforms the standard balancing problem into a complex routing and scheduling challenge. The presence of multiple worker resources, material or physical resources make the problem of balancing and scheduling best modeled with some variants of Resource-constrained project scheduling problem (RCPSP) [15].

2.3 RCPSP variants

A classical RCPSP consists in a list of tasks having precedence constraints, a list of resources with limited capacity, and resource consumption for each tasks. The goal is to assign start time to the tasks while respecting a set of constraints such as precedence and resource consumption. Our problem relates and includes several variants:

1. Multi-Mode RCPSP: [11] introduce different alternative/modes for tasks, as decision variable.
2. Generalized Precedence (RCPSP-GPR): [3], which significantly increases complexity by introducing the risk of cycles and infeasibility.
3. Preemptive RCPSP: [21] formalized the Partially Preemptive Multi-skill RCPSP, where tasks can pause and resume (specifically due to calendar constraints).
4. Resource blocking/zone-constrained: [7] [22] are interested in exclusion constraints linked to packed physical area, which add additional constraints on the assembly line scenario.

2.4 Optimization objectives

While academic benchmarks typically focus on makespan, industrial relevance requires multi-objective optimization. Minimizing Total Weighted Tardiness is paramount for delivery reliability [24]. Furthermore, Just-in-Time (JIT) principles introduce earliness costs to minimize Work-in-Progress (WIP), creating a non-regular objective function that penalizes both delays and premature production [28]. Additionally to JIT and WIP, we consider one more objective function related to resource-leveling objectives. [8], [12] are the fundamental work on minimizing resource usage peaks through project schedules. [25] optimized resource-level in a takt-based assembly line scheduling problem context.

2.5 Solving Methodology

Manual methodology and procedural solution have been predominant in solving scheduling problems such as assembly line balancing [9] or RCPSP [14]. From a mathematical programming perspective, many exact formulations of scheduling has been done with MIP [2]. For fine-grained scheduling over long horizons, time-indexed MIP formulations suffer from an explosion in the number of variables. Disjunctive flow formulations, while smaller, often provide weak linear relaxation bounds for cumulative resources [1]. To circumvent these scaling issues, decomposition methods such as Logic-Based Benders Decomposition (LBBD), pioneered by Hooker [13], separate resource allocation from sequencing. Modern Constraint Programming (CP) is well-suited for scheduling due to powerful abstractions and efficient algorithms [17]. The interval variable abstraction introduced by Laborie et al. [16, 17] allows tasks to be treated as single objects with start, end, and size domains. Crucially, the

concept of optionality natively handles the Multi-Mode aspect: unselected modes are simply "absent" intervals that do not participate in resource propagation. The efficiency of CP also increased thanks to scheduling global constraints like `Cumulative`. Vilím [29, 30] improved the propagation of these constraints by introducing fast Edge-Finding algorithms using Θ-tree data structures. Furthermore, Lazy Clause Generation (LCG) described by Feydy and Stuckey [10] has been a major driver of CP success, allowing solvers to "learn" from conflicts. Modern solvers like CP-SAT [19] use a portfolio approach, applying efficient Large Neighborhood Search (LNS) for upper bound improvements. Finally, for multi-objective exploration, recent algorithms such as Slide-and-Drill [6] offer efficient ways to approximate the Pareto front using CP/SAT technologies.

3 Problem Description

3.1 Inputs Definition

We address our flexible assembly line scheduling problem as a Multi-Mode Resource-Constrained Project Scheduling Problem (MRCPSP) extended with generalized precedence relations, time-dependent resource capacities, calendar-based preemption and complex resource blocking constraints. The problem instance is defined by the tuple $\mathcal{M} = (\mathcal{P}, \mathcal{T}, \mathcal{R}, \mathcal{G}, \mathcal{BG}, \mathcal{B}, \mathcal{C}, \mathcal{O}, H)$.

- **Products ($\mathcal{P}$):** A set of p products to be manufactured.
- **Tasks ($\mathcal{T}$):** A set of n production activities. Each task $i \in \mathcal{T}$ is associated with a set of valid execution modes M_i. For each mode $m \in M_i$, we define:
  - A nominal duration $D_{i,m}^{nom} \in \mathbb{N}$.
  - A resource requirement $r_{i,m,k} \in \mathbb{N}$ for each resource $k \in \mathcal{R}$.

  Additionally, each task i may have a release date rel_i and a deadline due_i. These can be hard constraints (validity bounds) or soft constraints (penalized if violated).
- **Resources ($\mathcal{R}$):** A set of renewable resources (e.g., stations, operators, tooling). Unlike the standard RCPSP, the capacity of a resource k is time-dependent, defined by an availability profile $C_k(t)$. We let $Capa_k$ denote the nominal maximum capacity of resource k. A subset of $\mathcal{R}$ represents Stations, which are typically disjunctive resources (capacity 1) corresponding to physical locations.
- **Task groups ($\mathcal{G}$):** A set of logical groupings, where each group $g \in \mathcal{G}$ contains a subset of tasks $\mathcal{T}_g \subseteq \mathcal{T}$. These groups represent high-level assembly phases performed at a specific station for a specific product.
- **Blocking constraints ($\mathcal{BG}, \mathcal{B}$):**
  - $\mathcal{BG}$: A set of **group blocking** constraints where a whole group g blocks a resource k (see Section 3.2.2).
  - $\mathcal{B}$: A set of **generic blocking** (or "bridging") constraints where a resource k is blocked during the transition between two entities (see Section 3.2.3).
- **Horizon (H):** The discrete planning horizon $\{0, \dots, H\}$.

3.2 Industrial Specifications

3.2.1 Calendar-based preemption

Standard preemption allows tasks to be interrupted arbitrarily. In our context, preemption is strictly calendar-driven. A task is interrupted only when a required resource becomes unavailable (e.g., during a shift change, weekend, or maintenance window). If a task i

requires a resource k that is unavailable during an interval $[t_1, t_2]$, the task execution pauses at t_1 and resumes at t_2. This increases the effective duration (wall-clock time) of the task beyond its nominal duration. In this work, we focus on "all-or-nothing" calendars, where $C_k(t) \in \{0, Capa_k\}$.

3.2.2 Group-based resource blocking

In assembly operations, a station must often be reserved continuously for a product across a sequence of tasks, including the idle time between them. We define a set of group blocking constraints $\mathcal{BG} \subseteq (\mathcal{G} \times \mathcal{R} \times \mathbb{N})$. A constraint $(g, k, q) \in \mathcal{BG}$ implies that q units of resource k are blocked for the entire duration of group g, from the start of its first task to the end of its last task. For disjunctive resources ($q = 1$), this ensures that once a product enters a station (starts group g), no other product can use that station until group g is fully complete.

3.2.3 Generic resource blocking

To model complex material handling (e.g., holding a part in a jig while moving between stations), we define **Resource Release Constraints** $\mathcal{B} \subseteq (\mathcal{X} \times \mathcal{X} \times \mathcal{R} \times \mathbb{N})$, where $\mathcal{X} = \mathcal{T} \cup \mathcal{G}$ (representing either single tasks or groups). A constraint $(X_{src}, X_{dst}, k, q) \in \mathcal{B}$ specifies that q units of resource k must be reserved from the *end* of entity X_{src} until the *start* of entity X_{dst}. This effectively creates a "bridge" reservation, preventing the resource from being used by other tasks during the transition period.

3.2.4 Precedence Constraints

With, S_i, E_i are the start / end dates of task i, the set $\mathcal{C}$ defines temporal relations between tasks, including:

- **Generalized precedence:** Minimal or maximal time lags between tasks (e.g., $S_j \geq S_i + \delta$ or $S_j \geq E_i + \delta$).
- **Synchronization:** Hard constraints enforcing simultaneous execution or fixed offsets (e.g., $S_j = S_i$).

3.2.5 Objectives

The optimization goal minimizes a cost function $\mathcal{O}$, defined as a weighted sum or lexicographic hierarchy of the following components:

- **Tardiness:** Minimizes delays against committed delivery dates: $\sum_{i \in \mathcal{T}} w_i^{tard} \times \max(0, E_i - due_i)$
- **Just-in-Time (JIT) Cost:** Penalizes early completion to reduce Work-in-Progress (WIP): $\sum_{i \in \mathcal{T}} w_i^{jit} \times \max(0, due_i - E_i)$
- **Resource cost:** Minimizes the usage of flexible resources (e.g., temporary staff). The maximum capacity becomes an integer decision variable $C_k^{max} \in [0, Capa_k]$. We minimize $\sum_{k \in \mathcal{R}} c_k \times C_k^{max}$
- **Makespan:** Minimizes the overall completion time: $\max_{i \in \mathcal{T}}(E_i)$

3.2.6 Illustration of the problem

Figure 1 illustrates the manufacturing process for a single product $p \in \mathcal{P}$. The product moves through Station 1 to complete Task Group 1 (containing tasks $\{T1, T2, T3\}$). Upon completion, the workflow splits across Stations 2 and 3 before converging at a final station.

This diagram highlights the concept of *group blocking constraint*: Station 1 is reserved for the entire span of Group 1, preventing any other product from entering. The dotted lines indicate resource requirements (e.g., workforce pools) shared across different stations and products.

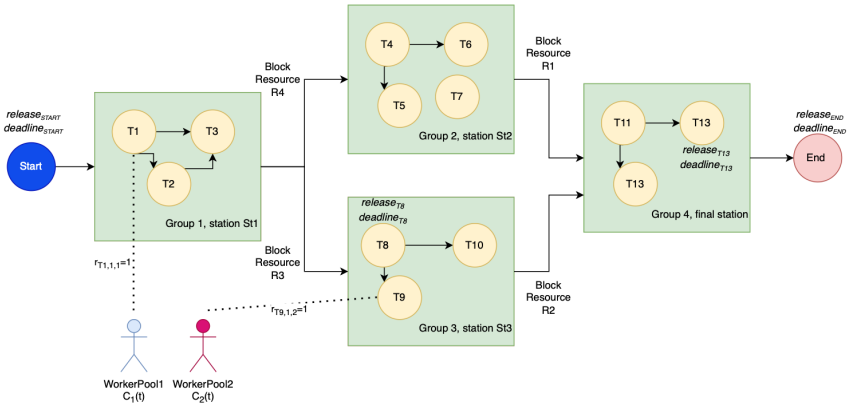

Figure 1 Example of the process of manufacturing one product.

Figure 2 depicts the physical layout of the stations. Different products follow different paths through the facility; for instance, Type A products traverse Stations $1 \to (2,3) \to 4$, while Type B products follow $3 \to (4,7) \to 5$. The shared usage of stations (e.g., Stations 3 and 4) could create a resource bottleneck in the system.

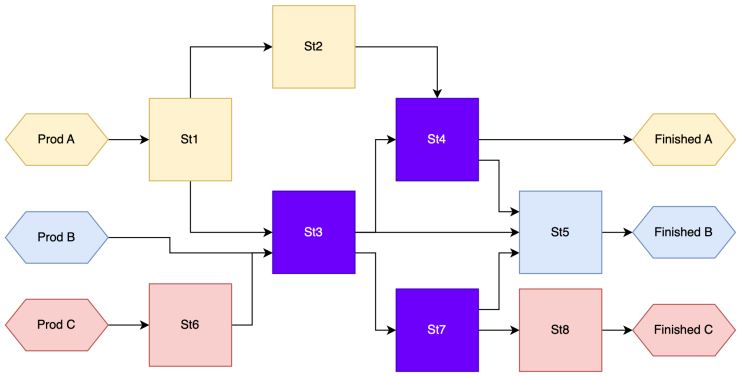

Figure 2 Stations layout view.

4 Solving methodology

4.1 Constraint Programming formulation

We propose a Constraint Programming model implemented using the CP-SAT solver, but is mostly generic and could be used with other solvers. The formulation relies on `Optional Interval` variables, which are fundamental to modern scheduling solvers. An optional interval variable I is defined by a start time s, a duration d, and a boolean literal p, and an end time e, such that $s + d = e$ if p is true. If p is false, the interval is absent and notably ignored by resource constraints.

4.1.1 Variables

For each task $i \in \mathcal{T}$ and each valid mode $m \in M_i$, we define:

- $x_{i,m} \in \{0, 1\}$: A Boolean variable indicating if mode m is selected.
- S_i, E_i, D_i: Integer variables for the global start time, end time, and actual duration of task i.
- $D_{i,m}$: Integer variable representing the duration of task i specifically in mode m.
- $I_{i,m}$: An optional interval variable representing the execution of task i in mode m. It is present only if $x_{i,m} = 1$ and is defined over $(S_i, D_{i,m}, E_i)$.

We enforce that exactly one mode is selected for each task:

$$\sum_{m \in M_i} x_{i,m} = 1 \quad \forall i \in \mathcal{T} \tag{1}$$

For every task group $g \in \mathcal{G}$, we define a master interval variable I_g spanning $[S_g, E_g]$, which covers the entire duration of the group's activities. Its bounds are linked to the tasks within the group $i \in \mathcal{T}_g$:

$$S_g = \min_{i \in \mathcal{T}_g}(S_i), \quad E_g = \max_{i \in \mathcal{T}_g}(E_i) \tag{2}$$

Precedence constraints and synchronizations described in 3.2.4 are directly written as linear constraints on start/end time of tasks.

4.1.2 Calendar preemption

A major challenge is that the effective duration of a task is not constant; it depends on the task's start time and the resource calendar. A task starting on Friday afternoon takes longer (wall-clock time) than one starting Monday morning due to the weekend break.

To avoid creating a lists of interval variables to define when a task is actually executed, we pre-compute the effective duration/span as a function of start time. Let $\Phi_{i,m}(t)$ be the time required to complete the nominal work $D^{nom}_{i,m}$ if the task starts at t, accounting for all resource pauses. If the resource capacity at t is insufficient to even start, we mark it as infeasible.

We collect all possible duration values into a set $\Delta_{i,m}$ by computing $\Phi_{i,m}(t)$ (a polynomial computation), over the horizon. For each possible duration $d \in \Delta_{i,m}$, there is a corresponding set of valid start times $\mathcal{T}_{i,m,d}$. In the constraint model, we introduce Boolean variables $\delta_{i,m,d}$ acting as indicators:

1. **Duration binding:** If the indicator is active, the mode-specific duration variable $D_{i,m}$ is fixed to the pre-calculated value d:

$$\forall d \in \Delta_{i,m}, \quad \delta_{i,m,d} = 1 \implies D_{i,m} = d \tag{3}$$

2. **Start time binding:** If $\delta_{i,m,d}$ is true, the start time S_i is part of the domain $\mathcal{T}_{i,m,d}$

$$\forall d \in \Delta_{i,m}, \ \delta_{i,m,d} = 1 \implies S_i \in \mathcal{T}_{i,m,d} \tag{4}$$

3. **Unique selection:** When mode m is selected, one duration indicator must be active and reversely.

$$x_{i,m} = 1 \iff \sum_{d \in \Delta_{i,m}} \delta_{i,m,d} = 1 \tag{5}$$

4.1.3 Resource constraints

We enforce resource limits using `NoOverlap` and `Cumulative` global constraints. The primary challenge lies in aggregating all potential consumers of a resource k, which includes tasks, and resource blocking constraint (group and generic).

For each resource $k \in \mathcal{R}$, we construct three sets of intervals:

1. **Task execution ($\mathcal{I}_k^{task}$):** Contains the intervals of all tasks requiring resource k.

$$\mathcal{I}_k^{task} = \{(I_{i,m}, r_{i,m,k}) \mid i \in \mathcal{T}, m \in M_i, r_{i,m,k} > 0\} \tag{6}$$

2. **Group blocking ($\mathcal{I}_k^{group}$):** Contains the span intervals of task groups that block resource k.

$$\mathcal{I}_k^{group} = \{(I_g, q) \mid (g, k, q) \in \mathcal{BG}\} \tag{7}$$

3. **Generic blocking ($\mathcal{I}_k^{bridge}$):** Contains intervals representing the transition periods defined in $\mathcal{B}$. For each constraint $(X_{src}, X_{dst}, k, q) \in \mathcal{B}$, we create a specific interval variable I_{bridge} with demand q, defined by:

$$\text{start}(I_{bridge}) = \text{end}(X_{src}), \quad \text{end}(I_{bridge}) = \text{start}(X_{dst}) \tag{8}$$

where $\text{start}(X)$ and $\text{end}(X)$ refer to the respective boundary variables of the entity X (whether it is a task or a group).

$$\mathcal{I}_k^{bridge} = \{(I_{bridge}, q) \mid (X_{src}, X_{dst}, k, q) \in \mathcal{B}\} \tag{9}$$

Finally, we define $\mathcal{U}_k = \mathcal{I}_k^{task} \cup \mathcal{I}_k^{group} \cup \mathcal{I}_k^{bridge}$ as the complete set of resource-consuming intervals. The constraint is applied based on the resource type:

- If resource k is of disjunctive type ($Capa_k = 1$):

$$\texttt{NoOverlap}(\{I \mid (I, q) \in \mathcal{U}_k\}) \tag{10}$$

- If resource k is of cumulative type ($Capa_k > 1$):

$$\texttt{Cumulative}(\mathcal{U}_k, Capa_k) \tag{11}$$

4.2 Objective functions

All but resource cost objective function are simply reuse of Section 3.2.5. For resources where capacity is flexible (e.g., hiring temporary staff), we introduce an integer decision variable $C_k^{max} \in [0, Capa_k]$ representing the peak capacity used.

To model objective $\sum c_k \cdot C_k^{max}$ we add another cumulative constraint for these resources to use the variable capacity:

$$\texttt{Cumulative}(\mathcal{U}_k, C_k^{max}) \tag{12}$$

5 Sequential meta-algorithm

To address the scalability challenges posed by industrial instances, we introduce a sequential solving strategy. This meta-algorithm decomposes the monolithic problem into a sequence of smaller sub-problems, which are solved iteratively using the base CP formulation described in Section 4.1.

5.1 Atom-based decomposition

We will first group tasks into clusters called *atoms*. To build atoms, we construct a dependency graph $G_{dep} = (\mathcal{T}, E_{dep})$ where an edge (i, j) exists if tasks i and j are strongly coupled and must be scheduled jointly to guarantee feasibility. Specifically, we add edges for tasks linked by:

- Resource blocking constraints: Either group-based blocking or generic blocking (bridging) constraints.
- Synchronization constraints: Hard equality or rigid offset constraints between start and end of some tasks.

An atom is then defined as a connected component of G_{dep}. This guarantees that no rigid constraint spans across different atoms, allowing to schedule them sequentially without immediately breaking feasibility. This decomposition seems particularly effective for our problem structure: atoms typically correspond to distinct production phases of a single product $p \in \mathcal{P}$, resulting in connected components that are largely independent but interact through standard precedence relations and shared resource capacity.

5.2 Ordering and batching

We define a directed graph G_{atom}, where vertices represent the atoms. An edge exists from atom A to atom B if any task in A is a predecessor of any task in B (via standard or generalized precedence). For our industrial instances, this graph is acyclic. We compute a topological sort to obtain a linear ordering of m atoms $\mathcal{A}_1, \ldots, \mathcal{A}_m$.

We then aggregate this sequence into N batches, balancing the batch size in a greedy manner to include for e.g approximately 50 tasks per batch. Note that if the atom graph contained cycles, this linear ordering would not be possible; however, the nature of our assembly process ensures an acyclic flow. Figure 3 illustrates both G_{dep} (red edges connecting task in each atoms, and G_{atom} linking atoms with precedence constraint. In this example, a linear ordering of atoms (non unique) is : $[(Atom)1, 3, 2, 4, 5, 7, 6]$.

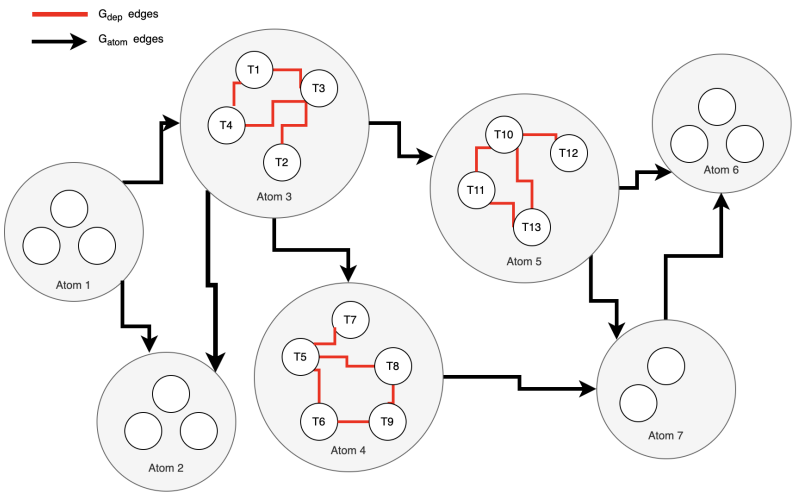

Figure 3 Atom decomposition.

5.3 Sequential solve algorithm

Once batches are determined, the algorithm solves the sub-problems in sequence, freezing the results of previous steps. The complete process of finding a feasible schedule is defined as follows:

1. **Initialize** an empty solution $\mathcal{S}_0$.
2. **Iterate** for $k = 1$ to N:
   a. **Build** model $\mathcal{M}_k$ containing all tasks from batches $1 \dots k$.
   b. **Fix past optimizations:** For all tasks i belonging to previous batches $(1 \dots k-1)$, add constraints fixing their start times, end times, and modes to the values found in $\mathcal{S}_{k-1}$.
   c. **Optimize** $\mathcal{M}_k$ with a limited time budget to produce the current solution $\mathcal{S}_k$.
3. **Return** the final schedule $\mathcal{S}_N$.

The fixing step reduces the search space of iteration k to only the variables of the new batch, while ensuring that new tasks respect the residual resource availability left by the frozen tasks. It is important to note that this approach is a constructive heuristic. It does not guarantee completeness or global optimality. In practice, early decisions might consume resource capacity in a way that makes later batches infeasible (e.g., due to calendar constraints or hard deadlines). The full sequential solving algorithms consist in : decomposing the problem into atoms and batches, building a feasible schedule S_N and solve the full CP model warm-started with S_N.

6 Computational Evaluation

6.1 Experimental setup

The model was implemented using the Python interface of OR-Tools CP-SAT. Our evaluations rely on a set of real industrial instances ranging from individual product lines to full-scale factory scenarios involving multiple interacting products.

6.1.1 Instance topology analysis

To characterize the complexity of these datasets beyond simple task counts, we analyzed both their constraint topology and resource characteristics. Table 1 details these metrics for three categories:

1. **Individual products (S1–S3):** Small-scale validation instances isolating single product flows.
2. **Baseline scenarios (H1–H3):** Full-scale production targets with three different station layouts.
3. **Ramp-up scenarios (H4–H6):** Stress-test instances requiring higher production quantity within the same horizon, significantly increasing resource contention.

We consider S1-S3 as the simple and H1-H6 as the hard scenarios. Key complexity drivers include the number of *disjunctive* resources (capacity=1), which are the primary bottlenecks, the volume of *generalized precedence* constraints. and the number of *blocking constraints* (both group-based and generic) imposing couplings between tasks.

Table 1 Topological characteristics of industrial instances. "Disj." indicates Disjunctive resources (Capacity=1).

Scenario	Tasks	Horizon	Resources		Precedence Constraints		Blocking Constraints (Generic)		
			Total	Disj.	Standard	Generalized	Total	$T \to T$	$T \to G/G \to T$
Simple Scenarios									
S1	25	2236	35	29	31	0	18	8	10
S2	11	2239	35	29	13	0	6	2	4
S3	8	2239	35	29	8	0	4	2	2
Hard Scenarios									
H1	396	3220	35	29	542	356	284	120	164
H2	396	3220	35	31	542	356	284	120	164
H3	396	3220	35	28	542	356	284	120	164
H4	534	3220	35	29	728	494	392	168	224
H5	450	3223	35	29	620	410	320	132	188
H6	432	3223	35	29	590	392	308	132	176

6.2 Experimental protocol

We conducted a comprehensive benchmark to assess three key aspects: (1) the scalability of our sequential approach, (2) the sensitivity to computational resources, and (3) the trade-offs between conflicting industrial objectives.

We compared two solving strategies:

- **Direct CP:** The monolithic model is instantiated for the entire horizon and solved directly.
- **Sequential CP (Seq-CP):** The meta-algorithm described in Section 5 generates an initial solution (using 10 batches), which is then injected as a warm-start hint into the monolithic model.

We also tested four lexicographic optimization hierarchies to reflect different planning priorities:

- **Order A (A-Tardiness):** [Tardiness]. Minimizes delays only.
- **Order B (B-JIT):** [Just-in-Time (JIT)]. Minimizes earliness only.
- **Order C (C-Tardiness 1st):** [Tardiness $\to$ JIT $\to$ Cost $\to$ Makespan]. The expert-preferred strategy, prioritizing on-time delivery.
- **Order D (D-JIT 1st):** [JIT $\to$ Tardiness $\to$ Cost $\to$ Makespan]. Prioritizes WIP reduction.

We use strict lexicographic optimization, optimizing independently each objective and set an hard constraint for the next iteration : no degradation of previous found objective is allowed.

We kept a time limit of 1000 seconds on each runs and vary the number of workers used per the solver ($\in \{1, 8, 16\}$). This leads to 24 different settings for each scenarios, so 72 experiments overall easy instances and 144 on hards. When using a number of workers > 1, the search is done in several threads but more importantly it uses different search strategy used in the CP-SAT solver. Experiments were run on an ARM-based machine (2.42 GHz, 32GB RAM).

6.3 Scalability of direct solving and sequential approach

Table 2 compares the success of the Direct CP and Seq-CP strategies. The sequential strategy demonstrates superiority, achieving a 100% success rate on all instance types. In contrast, the monolithic Direct CP consistently fails on the hard scenarios (H1-H6), even with maximum timeouts. We consider an experiment to be optimal if all lexicographic objectives have been proven optimal. We observe that for simple scenario, both direct and

Table 2 Percentage of solved and optimal runs by class of instance.

Method	Type	Success Rate	Optimal
Direct CP	Simple	100.0%	36/36
Direct CP	Hard	0.0%	0/72
Seq-CP	Simple	100.0%	36/36
Seq-CP	Hard	100.0%	16/72

sequential achieved 100% optimal experiments, validating that Seq-CP is not worse than direct method on simple scenarios. On hard instances, Seq-CP found fully optimal results on 16 out of 144 experiments, all of them for either orders **A** and **B**. No complete optimality proof is found for actual lexicographic orders C and D. The success of Seq-CP validates the decomposition strategy: the atom dependency graph is acyclic, and the sequential fixing of variables provides a valid feasible path that the monolithic solver doesn't discover on its own.

6.4 Solver performance and parallelism

We analyzed the impact of parallel workers parameter on convergence speed for the hardest instances. Figure 4 illustrates the evolution of the primal and dual bounds for Tardiness and JIT objectives.

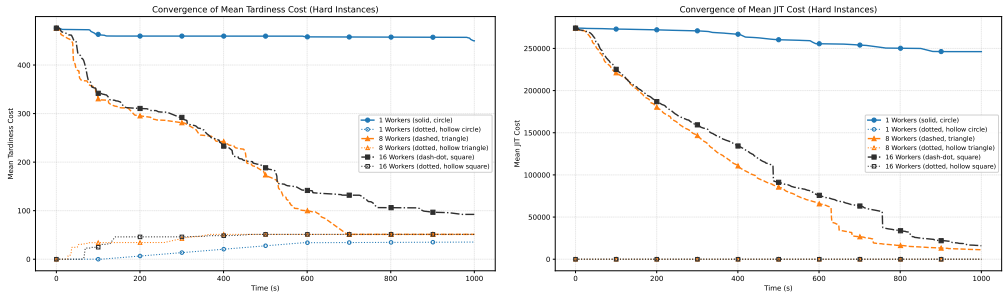

Figure 4 Evolution of Mean Tardiness and JIT cost on hard instances (1000s limit experiment). Solid lines represent the best feasible solution found (Primal), while dotted lines indicate the proven Lower Bound (Dual).

We observe that the best considered number of workers in terms of compromise between optimality and computation time is 8. It reaches near optimal proof for tardiness, as well as the best final objective value. For JIT cost, the dual bound remains identical for all settings (to 0) but best objective value is again found for 8 workers.

6.5 Objective orders effect

The choice of objective order dramatically shapes the final schedule. Table 3 details the KPIs for the hard instances. As expected, Order A (Tardiness only) and Order C (Tardiness first) successfully minimize delays but result in higher JIT costs. Conversely, Orders B and D aggressively reduce JIT cost but incur significant lateness penalties.

Table 3 Final KPI per objective order, **bold** numbers correspond to best value found for the corresponding objective over the 4 objective orders. The * corresponds to objective values where optimality proof have been provided by the solver (in its own lexicographical optimization perspective).

Instance	Order	Tardiness	JIT	Resource	Makespan
H1	A-Tardiness	**0***	274037	39	3220
-	B-JIT	22478	**0***	35	3220
-	C-Tardiness 1st	**0***	80569	35	**2114**
-	D-JIT 1st	9200	**0***	**34**	2626
H2	A-Tardiness	**0***	348605	39	3220
-	B-JIT	27829	**0***	37	3220
-	C-Tardiness 1st	**0***	80605	35	**2113**
-	D-JIT 1st	9261	**0***	**33**	2656
H3	A-Tardiness	**0***	281044	39	3220
-	B-JIT	22271	**0***	37	3220
-	C-Tardiness 1st	**0***	80937	37	**2113**
-	D-JIT 1st	9546	**0***	**36**	2626
H4	A-Tardiness	**307***	312601	37	3220
-	B-JIT	32164	**0***	35	3220
-	C-Tardiness 1st	**307***	211453	37	**2299**
-	D-JIT 1st	18392	**0***	**34**	3217
H5	A-Tardiness	**0***	283899	39	3223
-	B-JIT	7792	44630	37	3223
-	C-Tardiness 1st	**0***	116598	37	**2113**
-	D-JIT 1st	12976	**0***	**34**	2632
H6	A-Tardiness	**0***	400136	35	3223
-	B-JIT	7661	22727	37	3223
-	C-Tardiness 1st	**0***	109039	**32**	**2125**
-	D-JIT 1st	5698	**16743**	34	2551

Figure 5 visualizes this behavior on a specific subset of tasks. Order A places tasks well before their deadlines (large gap on the right), while Order B pushes them past the deadline. The lexicographic Orders C and D attempt to balance these extremes, though Order D often fails to recover from the initial lateness induced by the JIT minimization.

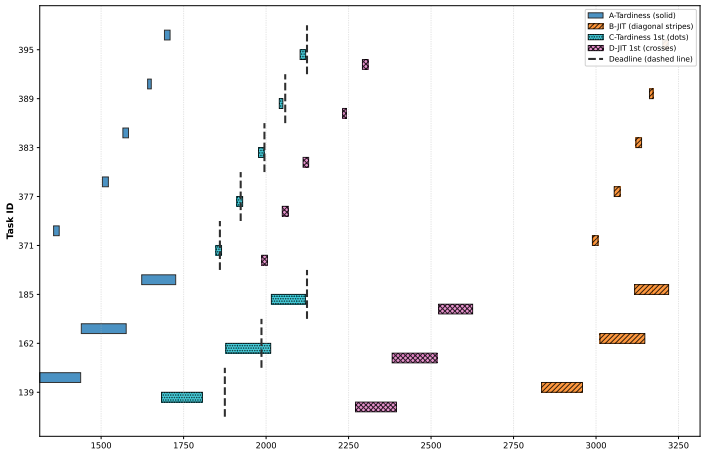

Figure 5 Schedules difference for some task between Order A, B, C, D on the H1 scenario.

Table 4 Tardiness and JIT values when optimizing Tardiness+JIT, and comparison with best sum values found in Table 3.

Instance	Tardiness	JIT	Sum	BestSum (A, B, C, D)
H1	8397	497	**8993**	9200 (+2.3%)
H2	8598	466	**9064**	9261 (+2.2%)
H3	9050	657	9707	**9546** (−2.2%)
H4	4388	78908	83296	**18392** (−78%)
H5	13783	14	13797	**12976** (−6%)
H6	4359	25922	30281	**22441** (−26%)

Observing that lexicographic optimization seems to fail to balance JIT and tardiness cost together, we attempted to optimize a weighted sum of both objectives (tardiness + JIT) to find a middle ground (Table 4), but the results were mixed. We infer an hypothesis: the LNS search used in the solver struggles with the combined objective, often failing to match the quality of the pure lexicographic strategies, indeed local optimization can help improve one component and degrade the other one, complicating the overall improvements.

6.6 Complexity Analysis

To understand the root cause of the computational difficulty on hard instances, we performed an ablation study on problem features to include in the CP model. We hypothesized that the resource blocking constraints (group-based and generic) were the primary bottleneck, as they create many intermediate interval variables with unknown duration, and that could degrade the propagation performance of global constraints.

To test this, we solved the scenarios using the monolithic model but with all blocking constraints removed using the optimization order **C-Tardiness 1st** with 1000s per objective and 8 workers for the solver. The difference in performance is significant. As shown in Table 5, the solver finds feasible solution for all hard scenarios. It also proves optimality for all instance in the first optimization step on Tardiness, and for 6 out of 9 instance on its second objective. This side experiments clearly highlight that the model/solver performance highly depends on the non-blocking resource, motivating further works to tackle this problem feature differently.

Table 5 Success run percentage, when removing resource blocking constraints. On each criteria column, is stored the number of runs where the optimality proof has been done.

Method	Type	Success Solve	Tardiness	JIT	Resource	Makespan
Direct CP	Simple	3/3	3/3	3/3	3/3	3/3
Direct CP	Hard	6/6	6/6	3/6	0/6	1/6

6.7 Artificial benchmark

To share to broader community and test other solving approaches, a generator of instances has been implemented to replicate the features of the industrial use case. Notably the parameters of the generator include: the number of products to produce, the number of types of similar products, number and layout of stations, number of cumulative resources. We ran additional experiments to validate our previous findings in explaining the complexity

of the problem. Another scheduling solver OptalCP [20] is also tested on this artificial instances. We successfully replicate the same pattern as the industrial use case. Details on the instance topology is detailed in Appendix E. 10 instances have been generated for simple, medium and hard instances. For simple to medium generated cases, a direct CP solving is efficient, whereas for larger problem, the Sequential approach is the preferred approach to get a feasible and efficient solution in our current time limit settings (1000 s). Table 6 lists the number of instance that are solved and which solvers obtained the best objective function per complexity class of instances. The use of the Optal solver increases the number of feasible solution on the hard instances (4 out of 10 are solved against only 3 for CPSAT), but the sequential solver is still the best ranked solver in terms of objective function.

Table 6 Detailed performance comparison of solver configurations across three problem complexity levels.

Complexity	Config	# 1	# 2	# 3	# failed
Simple	Direct CP (CPSAT)	**8**	0	0	2 (Unsat)
	Direct CP (Optal)	**8**	0	0	2
	Seq-CP	**8**	0	0	2
Medium	Direct CP (CPSAT)	6	2	2	0
	Direct CP (Optal)	3	1	6	0
	Seq-CP	**8**	2	0	0
Hard	Direct CP (CPSAT)	1	2	0	7
	Direct CP (Optal)	0	1	3	6
	Seq-CP	**7**	2	0	**1**

7 Conclusion and future work

To facilitate further research and reproducibility, the problem formulation and a set of representative artificial instances are released in [27], a library designed to enable fair and efficient comparison between different solvers and resolution techniques. The benchmark suite will serve as a playground for tackling the specific complexities studied in this paper, particularly the blocking constraints for which we believe more efficient approach could be tried.

In conclusion, we presented a Constraint Programming formulation for the mid-term planning of a flexible assembly line. This problem introduces distinct challenges beyond the classical RCPSP, specifically calendar-based preemption and complex resource blocking constraints.

Our computational evaluation highlighted that these blocking constraints are the primary driver of complexity, often preventing monolithic solvers from finding feasible solutions for high-load scenarios. To overcome this scalability barrier, we introduced a sequential solving meta-algorithm based on topological decomposition. This method proved robust, achieving a 100% success rate on large industrial instances where the direct approach failed. Furthermore, we analyzed the trade-offs between conflicting industrial objectives, with standard hierarchical optimization strategies.

This work represents a step towards bridging the gap between academic scheduling models and the complexities of industrial assembly lines. The proposed model offers a promising framework for both high-level planning and the validation of future factory designs in a mixed model product setup.

References

1 Philippe Baptiste, Claude Le Pape, and Wim Nuijten. *Constraint-based scheduling: applying constraint programming to scheduling problems*. Springer, 2001.

2 Ilker Baybars. A survey of exact algorithms for the simple assembly line balancing problem. *Management Science*, 32(8):909–932, 1986.

3 Lucio Bianco, Massimiliano Caramia, and Stefano Giordani. Project scheduling with generalized precedence relations: A new method to analyze criticalities and flexibilities. *European Journal of Operational Research*, 298(2):451–462, none 2022. doi:10.1016/j.ejor.2021.07.022.

4 Nils Boysen, Malte Fliedner, and Armin Scholl. A classification of assembly line balancing problems. *European Journal of Operational Research*, 183(2):674–693, 2007. doi:10.1016/J.EJOR.2006.10.010.

5 Liliana Capacho and Rafael Pastor. The ASALB problem with processing alternatives involving different tasks: Definition, formalization and resolution. In Marina L. Gavrilova, Osvaldo Gervasi, Vipin Kumar, Chih Jeng Kenneth Tan, David Taniar, Antonio Laganà, Youngsong Mun, and Hyunseung Choo, editors, *Computational Science and Its Applications - ICCSA 2006, International Conference, Glasgow, UK, May 8-11, 2006, Proceedings, Part III*, volume 3982 of *Lecture Notes in Computer Science*, pages 554–563. Springer, 2006. doi:10.1007/11751595_59.

6 João Cortes, Inês Lynce, and Vasco Manquinho. Slide&drill, a new approach for multi-objective combinatorial optimization. In *30th International Conference on Principles and Practice of Constraint Programming (CP 2024)*, volume 307 of *LIPIcs*, pages 8:1–8:17, 2024. doi:10.4230/LIPIcs.CP.2024.8.

7 Bruno Jensen Virginio da Silva, Reinaldo Morabito, Denise Sato Yamashita, and Horacio Hideki Yanasse. Production scheduling of assembly fixtures in the aeronautical industry. *Computers & Industrial Engineering*, 67:195–203, 2014. doi:10.1016/j.cie.2013.11.009.

8 Erik Demeulemeester. Minimizing resource availability costs in time-limited project networks. *Management Science*, 41(10):1590–1598, 1995. URL: https://EconPapers.repec.org/RePEc:inm:ormnsc:v:41:y:1995:i:10:p:1590-1598.

9 Erdal Erel and Subhash C. Sarin. A survey of the assembly line balancing procedures. *Production Planning & Control*, 9(5):414–434, 1998. doi:10.1080/095372898233902.

10 Thibaut Feydy and Peter J Stuckey. Lazy clause generation reengineered. In *Principles and Practice of Constraint Programming (CP 2009)*, pages 352–366. Springer, 2009. doi:10.1007/978-3-642-04244-7_29.

11 Sönke Hartmann. Project scheduling with multiple modes: A genetic algorithm. *Annals of Operations Research*, 102:111–135, 2001. doi:10.1023/A:1010902015091.

12 T. Hegazy. Optimization of resource allocation and leveling using genetic algorithms. *Journal of Construction Engineering and Management*, 125(3):167–175, 1999. cited By 243. doi:10.1061/(ASCE)0733-9364(1999)125:3(167).

13 John N Hooker. *Logic-based methods for optimization: combining optimization and constraint satisfaction*. Wiley, 2000.

14 Rainer Kolisch. Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. *European Journal of Operational Research*, 90(2):320–333, 1996. doi:10.1016/0377-2217(95)00357-6.

15 Rainer Kolisch and Sönke Hartmann. Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European Journal of Operational Research*, 174(1):23–37, 2006. doi:10.1016/J.EJOR.2005.01.065.

16 Philippe Laborie. Ibm ilog cp optimizer for detailed scheduling illustrated on three problems. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2009)*, pages 148–162. Springer, 2009. doi:10.1007/978-3-642-01929-6_12.

17 Philippe Laborie, Jérôme Rogerie, Paul Shaw, and Petr Vilím. Ibm ilog cp optimizer for scheduling. *Constraints*, 23(2):210–250, 2018.

18 Cemalettin Öztürk, Semra Tunalı, Brahim Hnich, and M. Arslan Örnek. Balancing and scheduling of flexible mixed model assembly lines. *Constraints*, 18(3):434–469, 2013. doi:10.1007/s10601-013-9142-6.

19 Laurent Perron and Frédéric Didier. CP-SAT. URL: https://developers.google.com/optimization/cp/cp_solver/.

20 Diego Olivier Fernandez Pons and Petr Vilím. https://optalcp.com, 2023.

21 Guillaume Povéda, Nahum Alvarez, and Christian Artigues. Partially Preemptive Multi Skill/Mode Resource-Constrained Project Scheduling with Generalized Precedence Relations and Calendars. In Roland H. C. Yap, editor, *29th International Conference on Principles and Practice of Constraint Programming (CP 2023)*, volume 280 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 31:1–31:21, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CP.2023.31.

22 Cédric Pralet, Stéphanie Roussel, Thomas Polacsek, François Bouissière, Claude Cuiller, Pierre-Eric Dereux, Stéphane Kersuzan, and Marc Lelay. A scheduling tool for bridging the gap between aircraft design and aircraft manufacturing. *Proceedings of the International Conference on Automated Planning and Scheduling*, 28(1):347–355, June 2018. doi:10.1609/icaps.v28i1.13910.

23 Xavier Pucel and Stéphanie Roussel. Constraint programming model for assembly line balancing and scheduling with walking workers and parallel stations. In *30th International Conference on Principles and Practice of Constraint Programming (CP 2024)*, volume 307 of *LIPIcs*, pages 23:1–23:21. Schloss Dagstuhl, 2024. doi:10.4230/LIPIcs.CP.2024.23.

24 Mohammad Ranjbar, Mohammad Khalilzadeh, Fereydoon Kianfar, and Kobra Etminani. An optimal procedure for minimizing total weighted resource tardiness penalty costs in the resource-constrained project scheduling problem. *Computers & Industrial Engineering*, 62(1):264–270, 2012. doi:10.1016/j.cie.2011.09.013.

25 Tamara Borreguero Sanchidrian, Tom Portoleau, Christian Artigues, Alvaro Garcia Sanchez, Miguel Ortega Mier, and Pierre Lopez. Large neighborhood search for an aeronautical assembly line time-constrained scheduling problem with multiple modes and a resource leveling objective. *Annals of Operations Research*, 338(1):13–40, July 2024. doi:10.1007/s10479-023-05629-3.

26 Armin Scholl and Christian Becker. State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. *European Journal of Operational Research*, 168(3):666–693, 2006. doi:10.1016/J.EJOR.2004.07.022.

27 AIR team. airbus/discrete-optimization, 2023. URL: https://github.com/airbus/discrete-optimization.

28 Mario Vanhoucke, Erik Demeulemeester, and Willy Herroelen. On maximizing the net present value of a project under renewable resource constraints. *Management Science*, 47(8):1113–1121, 2001. doi:10.1287/MNSC.47.8.1113.10226.

29 Petr Vilím. Edge finding filtering algorithm for discrete cumulative resources in $o(kn \log n)$. In *Principles and Practice of Constraint Programming (CP 2009)*, pages 802–816. Springer, 2009.

30 Petr Vilím. Timetable edge finding filtering algorithm for discrete cumulative resources. In *Integration of AI and OR Techniques in Constraint Programming (CPAIOR 2011)*, pages 230–245. Springer, 2011. doi:10.1007/978-3-642-21311-3_22.

A Direct versus Sequential, details

Table 7 Detailed comparison of Pure CP and Sequential CP on simple instances across all lexicographic orders. Both solvers reliably converge to the same optimal values.

Instance	Order	Method	Tardiness	WIP	Resource	Makespan
S1	A	Pure CP	**0**	**21242**	**10**	**577**
		Seq-CP	**0**	**21242**	21	**577**
	B	Pure CP	**1006**	**0**	**5**	**2146**
		Seq-CP	**1006**	**0**	6	2215
	C	Pure CP	**0**	**6248**	**10**	**1138**
		Seq-CP	**0**	**6248**	**10**	**1138**
	D	Pure CP	**502**	**0**	**7**	**1642**
		Seq-CP	**502**	**0**	**7**	**1642**
S2	A	Pure CP	**0**	**9298**	**10**	238
		Seq-CP	**0**	**9343**	18	**220**
	B	Pure CP	**187**	**0**	**4**	**1330**
		Seq-CP	**187**	**0**	18	**1330**
	C	Pure CP	**0**	**1431**	**4**	**1142**
		Seq-CP	**0**	**1431**	**4**	**1142**
	D	Pure CP	**152**	**0**	**4**	**1295**
		Seq-CP	**152**	**0**	**4**	**1295**
S3	A	Pure CP	**0**	**6063**	**2**	200
		Seq-CP	**0**	**6063**	18	**179**
	B	Pure CP	**107**	**0**	**2**	**1250**
		Seq-CP	**107**	**0**	**2**	**1247**
	C	Pure CP	**0**	**568**	**2**	**1142**
		Seq-CP	**0**	**568**	**2**	**1142**
	D	Pure CP	**104**	**0**	**2**	**1247**
		Seq-CP	**104**	**0**	**2**	**1247**

B Pareto view

By analyzing results in Table 3 we can plot the mean tardiness and JIT cost found by the 4 orders we tested and see the shape of the Pareto front in Figure 6.

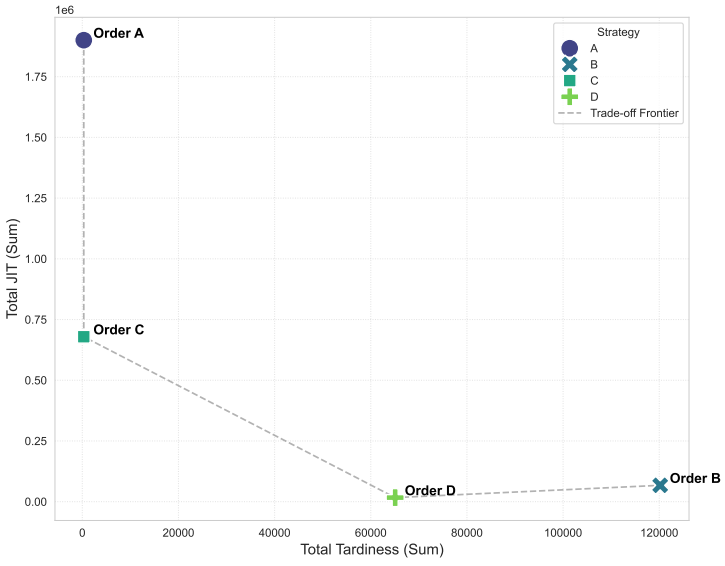

Figure 6 Pareto view (JIT/Tardiness) aggregating values for each order.

C Extension to general calendar profiles

Standard `Cumulative` constraints in CP solvers typically enforce a constant maximum capacity. However, our resources have time-dependent availability defined by a calendar $Av_k(t) \in [0, Capa_k]$. We model this by introducing virtual consumption (constant) intervals that represent the capacity lost during specific periods. Crucially, we distinguish between two types of unavailability to ensure consistency with the duration pre-calculation described above:

- **Total unavailability ($Av_k(t) = 0$):** Periods where the resource is completely unavailable are **excluded** from the cumulative formulation. Since the pre-calculated duration function $\Phi_{i,m}(t)$ already forces tasks to "jump over" these zero-capacity windows (effectively pausing progress), adding them to the cumulative constraint is redundant and would only increase solver overhead. This part is covered by the model presented in the paper.
- **Partial unavailability ($0 < Av_k(t) < Capa_k$):** Periods where capacity is reduced but non-zero require careful handling. A naive cumulative constraint including all tasks and all reduction intervals would fail. For instance, if a task requires 5 units and the capacity drops to 2 (there is virtual interval consuming of $Capa_k - 2$), the sum of the task requirement and the virtual consumption exceeds $Capa_k$. The solver would incorrectly mark this as infeasible, whereas in reality, the task should simply be "paused" or extended via the duration logic, rather than forbidden from overlapping the virtual interval.

To handle the issue on partial unavailability, we employ a decomposition strategy. We identify the set of unique capacity loss values $\mathcal{L}_k = \{\text{loss} \mid \exists t, Capa_k - Av_k(t) = \text{loss} > 0\}$. For each distinct loss level $v \in \mathcal{L}_k$, we generate a specific cumulative constraint that validates only the compatible combinations of tasks and calendar drops:

$$\forall v \in \mathcal{L}_k, \quad \texttt{Cumulative}(\mathcal{T}_{k,v} \cup \mathcal{F}_{k,v}, \text{demands} \cup \text{losses}, Capa_k) \tag{13}$$

Where $\mathcal{F}_{k,v}$ is the subset of virtual intervals where the capacity loss is $\leq v$, and $\mathcal{T}_{k,v}$ is the subset of tasks/modes (i, m) such that $r_{i,m,k} + v \leq C_k^{max}$. This ensures that for any given capacity drop, we only enforce the cumulative limit for tasks that are theoretically capable of running alongside that drop.

C.1 Limitation of the proposed model

A corner case happens with the proposed approach described above. Let's consider a resource k that has a availability profile of e.g : $Av_k = [0, 2, 8, 8, 0, 2, 8, 8, 0...]$, if 2 tasks t_1, t_2 need both 2 resources k and has nominal duration of 4 unit of time, then with the current model of partial unavailability, t_1 and t_2 won't be able to span together over time when the availability is 2. This actually leads to unexpected results and the schedule might be longer than with a calendar $Av_k = [0, 0, 8, 8, 0, 0, 8, 8, 0...]$. where the tasks could actually overlap during their execution. Having more resource available degrades the efficiency of the schedule as can be seen in Figure 7 The desired solution motivates to design a new model, illustrated in the third schedule of 7, can't be found using the calendar-based preemption model we currently modeled. Indeed, $T2$ is not running at a period of time when it could be potentially, as this resource is already taken by another task. $T2$ is really paused at this period of time, and the way to model such preemption is still under study, however a full preemptive model considering list of interval vars that will describe finely the task seems necessary.

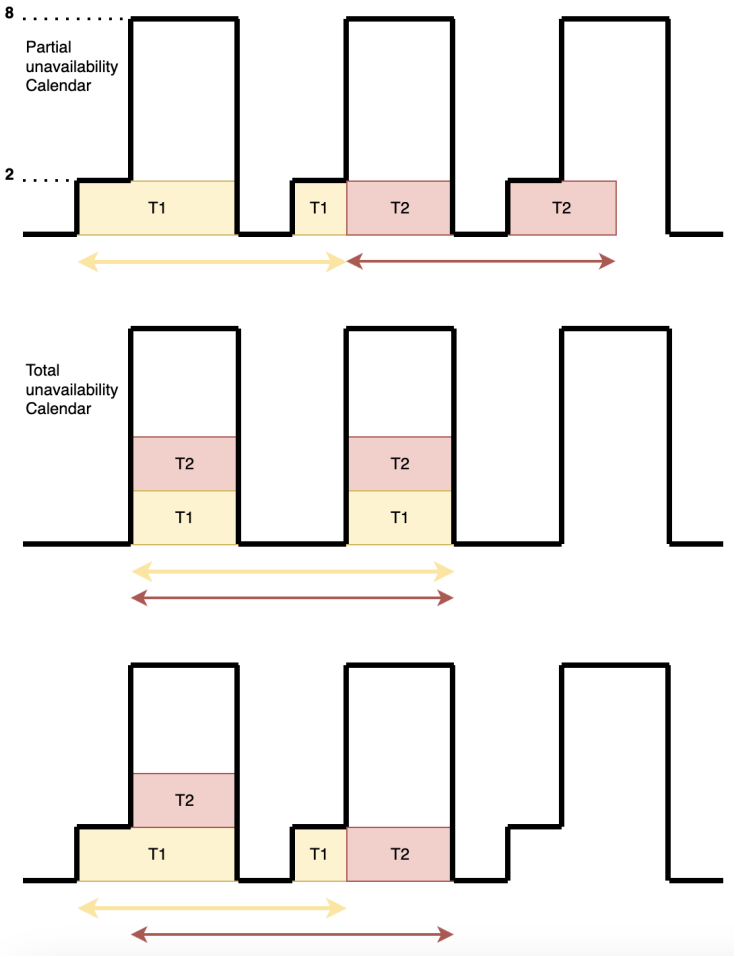

Figure 7 First 2 illustrations show examples of scheduling T1 and T2 under partial and total unavailability calendars using the proposed model based on calendar preemption. The 3rd one proposed a more efficient schedule, but where *calendar-based-preemption* is not longer well defined.

D CP model complexity analysis

Detailed results of the ablation study removing the resource blocking constraints are presented in Table 8. We see that the metrics are constantly better than results of order C presented in Table 3.

Table 8 Results found when running **Order C** optimisation, 1000s per optimisation, 8 workers, without adding the resource-blocking constraints.

Instance	Order	Tardiness	WIP	Resource	Makespan
H1	C	0	75976	35	2114
H2	C	0	77791	35	2114
H3	C	0	76867	37	2114
H4	C	276	182831	37	2293
H5	C	0	90318	37	2126
H6	C	0	82179	37	2126

E Artificial instances

Table 9 Aggregated statistics for generated instances.

Class	Tasks	Stations	Tools	Horizon	Blocking Constraints
Simple	30	32	12	5000	38
Medium	189	32	12	5000	236
Hard	516	32	12	7000	636