# Assembly Line Balancing with Parallel Stations and Shared Resources: A Cycle-Relative Constraint Programming Approach

**Diego Olivier Fernandez Pons**
OptalCP, France

**Pierre Schaus**
UCLouvain/ICTEAM, Louvain-la-Neuve Belgium

## Abstract

This paper addresses an assembly line balancing problem that combines task assignment, scheduling, and allocation of parallel workstations to minimize cycle time, subject to precedence and shared resource constraints. A key complexity in periodic production environments is the presence of cyclic resource constraints which arise when workers (resources) move between workstations within the same cycle. This scenario is particularly prevalent in large-scale manufacturing industries with long cycle times, such as aerospace assembly. Existing constraint programming (CP) models for this problem schedule tasks on an absolute time horizon. However, this approach relies on modulo operators to project task intervals into a cycle window, which hinders constraint propagation, and makes extending the model beyond fixed factory layouts difficult. We propose a novel cycle-relative CP formulation that addresses both shortcomings. Our model dynamically determines the optimal placement of parallel workstations and defines task decision variables directly within the cycle window, eliminating the need for modulo operators. Experimental evaluations demonstrate the cycle-relative model's superiority, consistently finding equal or better solutions faster than the temporal approach. Furthermore, a direct engine comparison shows that OptalCP outperforms CP Optimizer on these models across the vast majority of instances.

**2012 ACM Subject Classification** Replace ccsdesc macro with valid one

**Keywords and phrases** Constraint programming, line-balancing, cyclic scheduling, rcpsp

**Digital Object Identifier** 10.4230/LIPIcs.CVIT.2016.23

## 1 Introduction

Assembly Line Balancing (ALB) [6] is a fundamental optimization problem in manufacturing. Although common in automotive and appliance industries, the aerospace sector presents unique characteristics that shift the optimization focus. In high-volume industries like automotive, cycle times are measured in minutes or even seconds, and workers are strictly confined to their specific workstations to maintain the "takt" time. In contrast, aerospace assembly involves massive products (e.g., aircraft) that remain at a single production stage for several days. This extended time horizon—often an order of magnitude larger than individual task durations—allows for a high degree of resource flexibility.

Specifically, the *Multi-manned Assembly Line Balancing problem with Walking workers and Parallel stations* (MALBWP), as introduced in [14], captures these nuances. A defining feature of this environment is the presence of *walking workers*: because the aircraft stays in the station for a long period, workers have the time to physically move between different stages or stations to perform tasks. While the concept of shared resources is not exclusive to aerospace, the practical feasibility of workers moving between stations within the same cycle is a direct consequence of the long cycle times characteristic of this industry. Efficiently assigning and scheduling tasks in this context is crucial for minimizing cycle time and maximizing throughput.


The key components of the problem are:

- **Tasks:** A set $\mathcal{T}$ of tasks. Each task $\alpha \in \mathcal{T}$ has a fixed duration $d_\alpha > 0$.
- **Stages and Parallel Workstations:** The assembly line is organized as a sequence of *stages* $\mathcal{S} = \{1, \ldots, n_S\}$. Each stage $s \in \mathcal{S}$ consists of $w_s \geq 1$ identical *parallel workstations* (also called sub-stages). The line operates in a *pulsated* manner with a cycle time $c$: at every interval of $c$, a new aircraft enters and another exits each stage. An aircraft spends $w_s$ cycles in stage $s$ and there are exactly $w_s$ aircraft in stage $s$ at any point in time. Crucially, all parallel workstations in a given stage $s$ execute the *exact same schedule* of tasks. However, because aircraft enter the stage at different cycles, these identical schedules are shifted by exactly one cycle relative to each other. This structure allows for tasks with a duration $d_\alpha$ up to $w_s \times c$, which can be longer than the cycle time itself. The cycle time $c$ defines the production rate: one completed aircraft is delivered every $c$ time units. While parallelism enables the execution of long tasks, the overall throughput is determined by $c$. If the cycle time needs to be increased to accommodate longer tasks (even with parallelism), the production rate effectively slows down.
- **Precedences:** A set of precedence relations $P \subseteq \mathcal{T} \times \mathcal{T}$. If $(\alpha, \beta) \in P$, then task $\alpha$ must be completed before task $\beta$ can start on the same aircraft.
- **Resources:** A set of shared resources $\mathcal{R}$ (e.g., workers, tools). Each resource $r \in \mathcal{R}$ has a global capacity $\text{capacity}_r$. A task $\alpha$ requires $\text{consumption}_r^\alpha$ units of resource $r$ throughout its execution. A unique feature of this problem is the *walking worker* assumption; resources are not restricted to a single workstation but can move freely between them. In this model, we assume that the time required for a worker to move between workstations is negligible compared to the task durations and the overall cycle time, and is therefore considered to be instantaneous. Consequently, the capacity constraints are global and must account for the periodic nature of the assembly line.

The notations used throughout this paper are summarized in Table 1.

**Table 1** Summary of notation used in the formal problem description.

| Symbol | Description |
|---|---|
| $\mathcal{T}$ | Set of tasks to be balanced |
| $\alpha, \beta$ | Individual tasks in $\mathcal{T}$ |
| $d_\alpha$ | Duration of task $\alpha$ |
| $P$ | Set of precedence constraints $(\alpha, \beta)$ |
| $\mathcal{S}$ | Set of production stages $\{1, \ldots, n_S\}$ |
| $w_s$ | Number of parallel workstations in stage $s$ |
| $\mathcal{R}$ | Set of shared resources |
| $\text{consumption}_r^\alpha$ | Consumption of resource $r$ by task $\alpha$ |
| $\text{capacity}_r$ | Maximum capacity of resource $r$ |
| $c$ | Cycle time (objective to minimize) |

The state-of-the-art CP model for this problem [14] relies on a temporal formulation where tasks are scheduled on an absolute time horizon. While effective, this approach has limitations, particularly in how it handles cyclic resource constraints using modulo operators, which can weaken constraint propagation.

In this paper, we aim to introduce stronger models than the existing ones relying on CP interval variables by proposing a novel *cycle-relative* formulation. This formulation defines task decision variables directly within the cycle window, thereby eliminating the need for modulo operators and enhancing the overall efficiency of the search.

Our main contributions are as follows:

- We propose a novel cycle-relative CP formulation for the MALBWP that avoids modulo operators and improves propagation by defining variables directly in the cycle window.
- Our model naturally supports dynamic layout optimization, allowing the solver to determine the optimal number of parallel workstations per stage rather than relying on a predetermined configuration.
- We provide an extensive experimental evaluation comparing our cycle-relative model with the state-of-the-art temporal model using two modern CP engines: OptalCP and CP Optimizer.

The remainder of this paper is organized as follows. Section 1.1 provides a brief background on the CP scheduling constructs used in our models. Section 2 presents a simplified line balancing model with local resources. Section 3 introduces shared resources and walking workers, comparing the temporal and cycle-relative formulations. Section 4 extends the problem to stages with parallel workstations and dynamic layout optimization. Section 5 reviews the related work. Section 6 presents the experimental results, and Section 7 concludes the paper and discusses future research directions.

## 1.1 Constraint Programming for Scheduling

Modern commercial CP-scheduling solvers such as [18, 10] provide specialized modeling constructs that enable the natural representation of scheduling problems through *optional interval variables* and *cumulative functions*, introduced in [9, 11]. We briefly recall here the main constructs used in the presented models.

An *interval variable* represents a time interval during which an activity occurs. Each interval variable $a$ is characterized by three attributes: a start time $\texttt{start}(a)$, an end time $\texttt{end}(a)$, and a duration $\texttt{duration}(a)$, The relationship $\texttt{start}(a) + \texttt{duration}(a) = \texttt{end}(a)$ always holds.

An *optional interval variables* enables the interval variable to be present or absent state. The constraint $\texttt{alternative}(a, \{a_1, \dots, a_n\})$ ensures that if interval $a$ is present, then exactly one interval from $\{a_1, \dots, a_n\}$ is present, and that $a$ starts and ends together with the chosen interval. If $a$ is absent, all intervals in $\{a_1, \dots, a_n\}$ are absent.

*Cumulative functions* provide a powerful mechanism for modeling resource usage over time (see [11, 17]). A cumulative function is a piecewise constant function $f : \mathbb{Z} \to \mathbb{Z}$ that represents the aggregated contribution of activities to a resource level. For any time point $t \in \mathbb{Z}$, $f(t)$ returns the total level of consumption or production of the resource at that time.

The most basic building block is the $\texttt{pulse}(a)$ function, which defines an elementary cumulative function that is zero everywhere except over the interval $a = [s, e)$, where it takes the value 1:

$$\texttt{pulse}(a)(t) = \begin{cases} 1 & \text{if } s \leq t < e \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Cumulative functions can be combined linearly to form more complex ones, $f(t) = \sum_i w_i \cdot f_i(t)$. When all interval variables involved in a cumulative function are fixed, $f$ becomes a fixed piecewise constant function. The constraint $0 \leq f \leq C$ ensures that the resource consumption $f(t)$ remains within the capacity $C$ at every time point $t \in \mathbb{Z}$.

## 2 Stages with Single Stations and Local Resources

The simpler case with $w_s = 1\ \forall s \in \mathcal{S}$, that is, a unique station at every stage, is considered first. Furthermore, in this simplified version of the problem, the resources are not shared. Each station has its own resources, which models the fact that, as in traditional line balancing, workers remain at their stations, and capacity constraints are applied station by station (Figure 1). A typical constraint programming model would create an optional task per stage, $T_s^\alpha$, and express the capacity constraint on these optional tasks.

$$\min c$$

$$\texttt{alternative}(T^\alpha, \{T_s^\alpha \mid s \in \mathcal{S}\}) \qquad \forall \alpha \in \mathcal{T} \tag{2}$$

$$\texttt{duration}(T^\alpha) = \texttt{duration}(T_s^\alpha) = d_\alpha \qquad \forall \alpha \in \mathcal{T}, \forall s \in \mathcal{S} \tag{3}$$

$$(s-1) \times c \leq \texttt{start}(T_s^\alpha) \wedge \texttt{end}(T_s^\alpha) \leq s \times c \qquad \forall \alpha \in \mathcal{T}, \forall s \in \mathcal{S} \tag{4}$$

$$\texttt{end}(T^\alpha) \leq \texttt{start}(T^\beta) \qquad \forall (\alpha, \beta) \in P \tag{5}$$

$$\sum_{\alpha \in \mathcal{T}} \text{consumption}_r^\alpha \times \texttt{pulse}(T_s^\alpha) \leq \text{capacity}_r \qquad \forall r \in \mathcal{R}, \forall s \in \mathcal{S} \tag{6}$$

$$T^\alpha, T_s^\alpha \subseteq [0, +\infty] \qquad \forall \alpha \in \mathcal{T}, \forall s \in \mathcal{S}$$

$$c \in [0, +\infty]$$

Equation (2) assigns each task to a station, (3) sets its duration, while (4) ensures it remains within the (variable) boundaries of the assigned station. Equation (5) enforces precedences between tasks. Finally (6) expresses worker capacity constraints station by station.

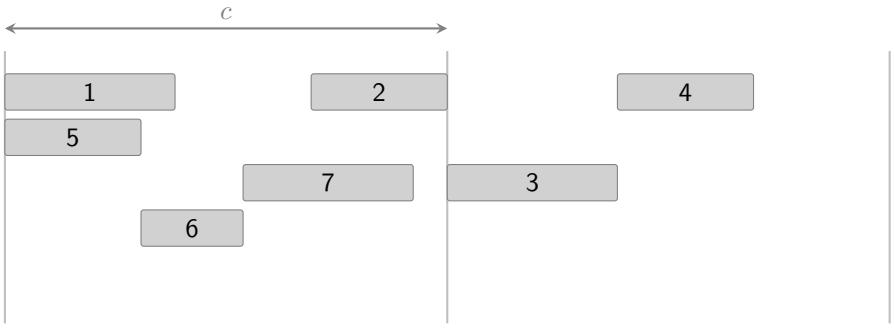


**Figure 1** Standard line-balancing schedule where tasks are assigned to individual workstations within a fixed cycle time $c$.

Capacity constraints for each station do not interfere with each other as they act on different time slices. Hence they can be merged into a single capacity constraint that spans the whole horizon, removing the need for optional tasks. This makes the model more efficient as capacity constraints propagate better when acting on mandatory intervals $T^\alpha$ rather than optional ones $T_s^\alpha$ (see [17, 2] for propagating cumulative functions over optional intervals).

As we remove optional tasks, we instead introduce an integer variable $s^\alpha \in \mathcal{S}$ to represent the station to which task $\alpha$ is assigned.

$$\min c$$

$$\texttt{duration}(T^\alpha) = d_\alpha \qquad \forall \alpha \in \mathcal{T} \tag{7}$$

$$(s^\alpha - 1) \times c \leq \texttt{start}(T^\alpha) \wedge \texttt{end}(T^\alpha) \leq s^\alpha \times c \qquad \forall \alpha \in \mathcal{T} \tag{8}$$

$$\texttt{end}(T^\alpha) \leq \texttt{start}(T^\beta) \qquad \forall (\alpha, \beta) \in P \tag{9}$$

$$\sum_{\alpha \in \mathcal{T}} \text{consumption}_r^\alpha \times \texttt{pulse}(T^\alpha) \leq \text{capacity}_r \qquad \forall r \in \mathcal{R} \tag{10}$$

$$s^\alpha \in \mathcal{S},\ T^\alpha \subseteq [0, +\infty] \qquad \forall \alpha \in \mathcal{T}$$

$$c \in [0, +\infty]$$

Equation (7) fixes the length of tasks, (8) via the decision variable $s^\alpha$, simultaneously handles the assignment of task $\alpha$ to a station and limits it to the time window of the corresponding station. Equation (9) enforces precedences between tasks, and (10) expresses worker capacity constraints as a single capacity spanning the entire horizon using intervals $T^\alpha$.

## 3 Stages with Single Stations and Shared Resources

We now consider the case where resources are shared between all stations, enabling workers to move between stages within the same cycle. This "walking worker" assumption implies that capacity constraints must be enforced globally across the entire line. Consistent with the long cycle times discussed in the introduction, we assume that worker transition times between workstations are negligible and can be modeled as instantaneous.

In the presence of walking workers, stations schedules interfere with each other and the capacity constraint must be enforced for all stations simultaneously. The problem being periodic, capacity constraints can be enforced on a single time slice of length c. Similarly to [14] we choose to enforce capacity constraints on the time slice $[0, c]$ and introduce modular representatives $M^\alpha$ for the tasks $T^\alpha$ defined by $M^\alpha = [\texttt{start}(T^\alpha) \mod c,\ \texttt{end}(T^\alpha) \mod c]$.

▶ **Example 1.** Consider Figure 2 where each line represents a worker: at the beginning of the cycle worker 1 performs task 1 at station 1, walks to station 2, performs task 4, walks back to station 1 and performs task 2. Regardless of their location, tasks 1, 4 and 2 follow each other and cannot overlap in time.

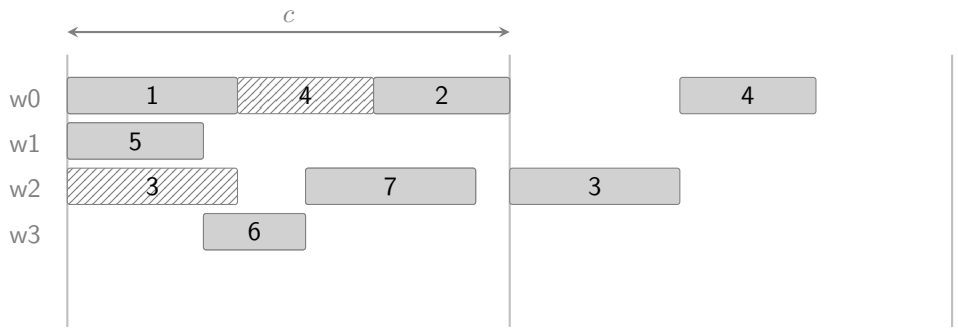


**Figure 2** Periodic schedule showing tasks $T^\alpha$ (solid gray) and their corresponding modular representatives $M^\alpha$ (hatched, white background) within the cycle time $c$.

## 3.1 Temporal Formulation

The temporal formulation is a simplified version of the model introduced in [14]. The model presented here is the line-balancing with workers model where the capacity constraints are simultaneously applied on all stations on the time slice $[0, c]$ using modular tasks.

$$\min c$$

$$0 \leq \texttt{start}(M^\alpha) \wedge \texttt{end}(M^\alpha) \leq c \qquad \forall \alpha \in \mathcal{T} \qquad (11)$$

$$\texttt{duration}(M^\alpha) = \texttt{duration}(T^\alpha) = d_\alpha \qquad \forall \alpha \in \mathcal{T} \qquad (12)$$

$$\texttt{start}(M^\alpha) + (s^\alpha - 1) \times c = \texttt{start}(T^\alpha) \qquad \forall \alpha \in \mathcal{T} \qquad (13)$$

$$\texttt{end}(M^\alpha) + (s^\alpha - 1) \times c = \texttt{end}(T^\alpha) \qquad \forall \alpha \in \mathcal{T} \qquad (14)$$

$$\texttt{end}(T^\alpha) \leq \texttt{start}(T^\beta) \qquad \forall (\alpha, \beta) \in P \qquad (15)$$

$$\sum_{\alpha \in \mathcal{T}} \text{consumption}_r^\alpha \times \texttt{pulse}(M^\alpha) \leq \text{capacity}_r \qquad \forall r \in \mathcal{R} \qquad (16)$$

$$s^\alpha \in \mathcal{S},\ T^\alpha, M^\alpha \subseteq [0, +\infty] \qquad \forall \alpha \in \mathcal{T}$$

$$c \in [0, +\infty]$$

Equations (11 - 14) define modular representatives $M^\alpha$ using arithmetic expressions (rather than a modulo operator that may not be available in all CP systems), confining simultaneously $M^\alpha$ to $[0, c]$ and $T^\alpha$ to station $s^\alpha$ $[(s^\alpha - 1) \times c, s^\alpha \times c]$. Equation (15) enforces precedences and (16) enforces global capacity constraints expressed with $M^\alpha$ variables.

## 3.2 Cycle-Relative Formulation

The temporal model uses absolute time tasks $T^\alpha$ to express precedences (15), and their modular representatives $M^\alpha$ to express capacity constraints (16). The cycle-relative model is obtained by expressing the precedences on the modular representatives (19 - 20) and dropping the absolute time tasks.

$$\min c$$

$$0 \leq \texttt{start}(M^\alpha) \wedge \texttt{end}(M^\alpha) \leq c \qquad \forall \alpha \in \mathcal{T} \qquad (17)$$

$$\texttt{duration}(M^\alpha) = d_\alpha \qquad \forall \alpha \in \mathcal{T} \qquad (18)$$

$$s^\alpha \leq s^\beta \qquad \forall (\alpha, \beta) \in P \qquad (19)$$

$$s^\alpha = s^\beta \Rightarrow \texttt{end}(M^\alpha) \leq \texttt{start}(M^\beta) \qquad \forall (\alpha, \beta) \in P \qquad (20)$$

$$\sum_{\alpha \in \mathcal{T}} \text{consumption}_r^\alpha \times \texttt{pulse}(M^\alpha) \leq \text{capacity}_r \qquad \forall r \in \mathcal{R} \qquad (21)$$

$$s^\alpha \in \mathcal{S},\ M^\alpha \subseteq [0, +\infty] \qquad \forall \alpha \in \mathcal{T}$$

$$c \in [0, +\infty]$$

Equations (17) and (18) are the only equations left from the definition of $M^\alpha$ when the link with intervals $T^\alpha$ is removed. Equations (19 - 20) state the predecessor of a task either has to be in a previous station, or if it is in the same station, has to finish before its successor starts. Equation (21) is the global capacity constraint.

For performance reasons due to (20) being a constraint on the right side of a logical operator (also known as a meta-constraint), constraints (19) and (20) are better written with a lexicographic operator instead

$$[s^\alpha, \texttt{end}(M^\alpha)] \ \leq_{lex} \ [s^\beta, \texttt{start}(M^\beta)]$$

# 4 Stages with Parallel Stations and Shared Resources

Parallel stations are a convenient way to handle very long tasks in line-balancing because they circumvent the limitation that tasks are shorter than the cycle time.

A parallel station has two identical substations of length $2c$ running in parallel. When an aircraft enters the station, it occupies the substation that is available and stays $2c$ units of time before moving to the next station.

▶ **Example 2.** Consider Figure 3 which shows the schedule of three aircraft, distinguished by fill pattern: solid, dotted, and hatched. Tasks 1 and 2 are scheduled on station 1, tasks 3, 4 and 5 on parallel station 2, task 6 on station 3. Because parallel station 2 is divided into two substations a and b, it can fit task 4 despite its length being larger than the cycle time. Notice how aircraft alternate on the substations of parallel station 2.

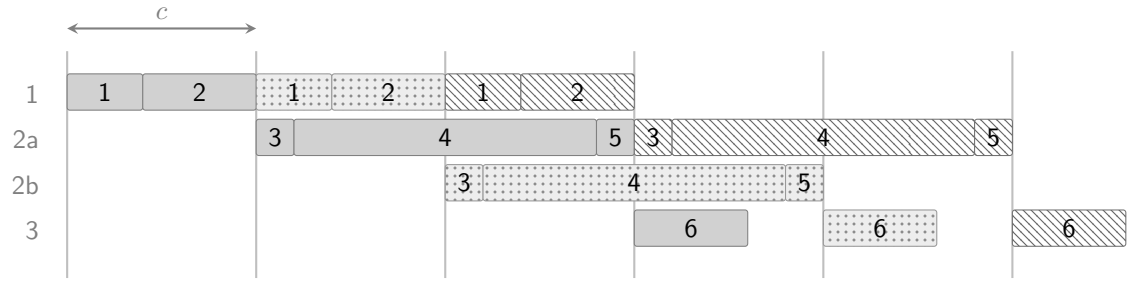


**Figure 3** Schedule across multiple stages including a parallel workstation (Station 2), illustrating how three aircraft (solid, dotted, hatched) move through the line.

## 4.1 Handling Parallelism with Modular Tasks

Introducing tasks modulo $c$ is more complex in the presence of parallel stations because tasks may "curl around" and their modular representatives may need to be cut into pieces. A modular representative of task 4 in Figure 3 would look like one of the cases in Figure 4 depending on its start time modulo c.

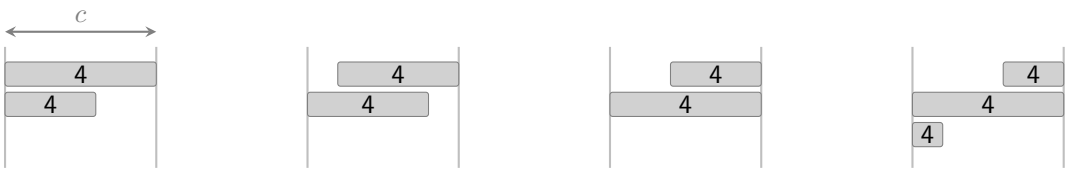


**Figure 4** Modular representation of a long task assigned to a parallel workstation (task 4 from Figure 3), illustrating how it 'curls around' the cycle boundary. In each sub-figure the task is scheduled one more time-unit to the right.

We avoid the segmented modular representative issue by locally unfolding the schedule.

## 4.2 Handling Parallelism by Unfolding the Cyclic Schedule

To avoid the segmented modular representative representation in the slice $[0, c]$, we consider the slice $[-c, c]$ of the original cyclic schedule and keep the tasks that start in $[-c, 0]$ and

extend into $[0, c]$. The cumulative profile of the schedule on $[0, c]$ is exactly the same as in the cyclic schedule because we retain all tasks that intersect the slice $[0, c]$ (Figure 5).

▶ **Example 3.** Figure 5 shows the schedule of Figure 3 unfolded over the time slice $[-c, 2c]$ with one line per task type. All tasks repeat every $c$ units of time. We want to compute the resource profile over the interval $[0, c]$. Task 4 intersects $[0, c]$ twice because it is assigned to a parallel station and lies on both sides of $\{c\}$. All other tasks, assigned to parallel stations or not, are fully included in $[0, c]$.

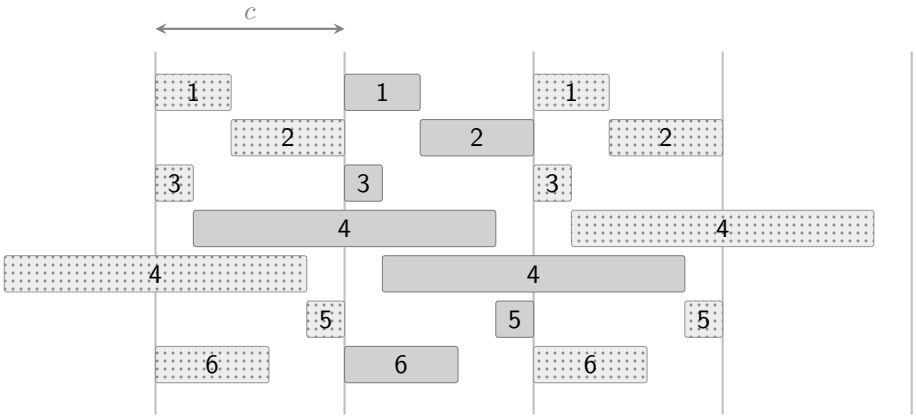


**Figure 5** Unfolded schedule over the interval $[-c, 2c]$: dotted tasks lie outside the target window $[0, c]$; solid-filled tasks are the ones that contribute to the resource profile in $[0, c]$.

Only the solid-filled tasks in Figure 5 need to be represented for the resource profile to be correct in the time slice $[0, c]$. A simple option is to keep all tasks that start in $[-c, c]$, another is to make the dotted tasks in $[-c, 0]$ optional and force them to be absent when not needed (e.g. not assigned to a parallel station or not crossing into $[0, c]$).

## 4.3 The Unfolded Model Up to two Parallel Stations per Stage

The cycle-relative model introduced in Section 3.2 is now generalized to accommodate up to two parallel workstations at any stage, such that $w_s \in \{1, 2\}$ for all $s \in \mathcal{S}$.

To handle tasks that may span across the cycle boundary when assigned to a parallel workstation, we extend the considered time slice backward to $[-c, c]$. For each task $\alpha \in \mathcal{T}$, we introduce two interval variables: a historical copy $M_0^\alpha$, constrained to start in $[-c, 0]$, and a base copy $M_1^\alpha$, constrained to start in $[0, c]$.

The upper bounds of these intervals depend dynamically on the number of parallel workstations at the assigned stage, denoted by $w_{s^\alpha}$. Here, $w_s \in \{1, 2\}$ indicates whether stage $s$ is a simple (single) or parallel (double) workstation. The expression $w_{s^\alpha}$ represents the capacity of the specific stage to which task $\alpha$ is assigned. This relationship is concisely captured in the CP model using an *element* constraint (a variable-indexed array lookup). If a task is assigned to a single-station stage ($w_{s^\alpha} = 1$), its duration cannot exceed $c$. Thus, the end of $M_0^\alpha$ is strictly bounded by 0, and the end of $M_1^\alpha$ is bounded by $c$. Conversely, if the task is assigned to a parallel-station stage ($w_{s^\alpha} = 2$), it may last up to $2c$. In this case, the bound for $M_0^\alpha$ extends to $c$, and the bound for $M_1^\alpha$ extends to $2c$. Furthermore, we ensure the second interval is an exact cyclic copy of the first by enforcing a strict temporal offset of $c$.

$$\min c$$

$$\texttt{start}(M_k^\alpha) \geq (k-1) \times c \qquad \forall \alpha \in \mathcal{T}, \forall k \in \{0,1\} \tag{22}$$

$$\texttt{end}(M_k^\alpha) \leq (k + w_{s^\alpha} - 1) \times c \qquad \forall \alpha \in \mathcal{T}, \forall k \in \{0,1\} \tag{23}$$

$$\texttt{start}(M_0^\alpha) + c = \texttt{start}(M_1^\alpha) \qquad \forall \alpha \in \mathcal{T} \tag{24}$$

$$[s^\alpha, \texttt{end}(M_k^\alpha)] \ \leq_{lex} \ [s^\beta, \texttt{start}(M_k^\beta)] \qquad \forall (\alpha, \beta) \in P, \forall k \in \{0,1\} \tag{25}$$

$$\sum_{\alpha \in \mathcal{T}} \sum_{k \in \{0,1\}} \text{consumption}_r^\alpha \times \texttt{pulse}(M_k^\alpha) \leq \text{capacity}_r \qquad \forall r \in \mathcal{R} \tag{26}$$

$$s^\alpha \in \mathcal{S}, \ M_k^\alpha \subseteq [-\infty, +\infty] \qquad \forall \alpha \in \mathcal{T}, \forall k \in \{0,1\}$$

$$\texttt{duration}(M_k^\alpha) = d_\alpha \qquad \forall \alpha \in \mathcal{T}, \forall k \in \{0,1\}$$

$$c \in [0, +\infty]$$

Equations (22) and (23) establish the dynamic time windows for the interval copies based on the assigned stage's capacity. Equation (24) ties the two intervals together with a fixed spacing of $c$. Precedence relations are enforced across both stages and intervals by equation (25). Finally, equation (26) computes the global resource consumption over the expanded horizon $[-c, 2c]$.

The primary decision variables in this formulation are the intervals $M_1^\alpha$, which represent the task execution within $[0, 2c]$. The historical copies, $M_0^\alpha$, only extend into the active resource evaluation window $[0, c]$ when a task is assigned to a parallel station and $\texttt{end}(M_0^\alpha) > 0$. In all other scenarios, $M_0^\alpha$ remains strictly within the negative time slice $[-c, 0]$, does not contribute to the resource profile within $[0, c]$, and is for all practical purposes efficiently ignored by the solver's propagation engine.

## 4.4 Comparative Analysis of Unfolded and Pucel-Roussel Formulations

Both models work by computing the resource profile for the capacity constraints over the time slice $[0, c]$, and both models need to handle tasks that are assigned to parallel stations in a special way because these tasks lie in $[0, 2c]$ and may extend into the time slice $[c, 2c]$ therefore "curling around" and coming back into the time slice $[0, c]$ "by the left". An example of such tasks is task 4 in Figure 4 and Figure 5.

Pucel and Roussel [14] handle the task $T = [\texttt{start}(T), \ \texttt{end}(T)]$ that extends into $[c, 2c]$ by cutting it into 2 pieces $S_1 = [\texttt{start}(T), \ c]$ and $S_2 = [c, \ \texttt{end}(T)]$. Then $S_2$ is translated back into $[0, c]$ by $S_2' = [0, \ \texttt{end}(T) - c]$ and the capacity constraint is applied to $S_1$ and $S_2'$.

We instead introduce task $T' = [\texttt{start}(T) - c, \ \texttt{end}(T) - c]$ which is the translation of task $T$ per $-c$ and apply the capacity constraint to $T$ and $T'$ over the horizon $[-c, 2c]$ noticing that over the interval $[0, c]$ the resource profile is correct, thus the capacity constraint is properly enforced. We avoid conditional transformations, at the expense of task duplication and extended horizon for the capacity constraint.

Variants of the unfolded model that limit task duplication can be designed as only tasks assigned to parallel stations that extend into the time interval $[c, 2c]$ need to be duplicated. A simple approach is to make all $T'$ tasks optional and only make them present when $T$ is assigned to a parallel station. This would make the unfolded model isomorphic to the cycle-relative model after root propagation whenever the layout doesn't have parallel stations.

Moreover, because the only thing that distinguishes a task assigned to a parallel station from a single station is $\texttt{end}(T) \leq 2c$ instead of $\texttt{end}(T) \leq c$, our model can easily accommodate dynamic layouts.

## 4.5 Arbitrary Number of Parallel Stations per Stage

We now generalize the cycle-relative formulation to accommodate an arbitrary factory layout, where any number of stages can have multiple parallel workstations. Let $w_s \geq 1$ denote the number of parallel workstations available at stage $s \in \mathcal{S}$.

When a stage possesses $w_s$ workstations, a task assigned to it can span up to $w_s \times c$ units of time. Consequently, its execution can "curl around" and overlap the base time slice $[0, c]$ up to $w_s$ times. To model this using cycle-relative variables without relying on modulo operators, we must unfold the schedule backward far enough to capture all possible historical overlaps.

Let $W = \max_{s \in \mathcal{S}} w_s$ be the maximum number of parallel workstations available across the entire assembly line. We introduce $W$ intervals for each task $\alpha \in \mathcal{T}$, indexed by $k \in \mathcal{K}$ with $\mathcal{K} = \{0, \dots, W-1\}$. The generalized model is formulated as follows:

$$\min c$$

$$\texttt{start}(M_k^\alpha) \geq (k - W + 1) \times c \qquad \forall \alpha \in \mathcal{T}, \forall k \in \mathcal{K} \tag{27}$$

$$\texttt{end}(M_k^\alpha) \leq (k - W + 1 + w_{s^\alpha}) \times c \qquad \forall \alpha \in \mathcal{T}, \forall k \in \mathcal{K} \tag{28}$$

$$\texttt{start}(M_k^\alpha) + c = \texttt{start}(M_{k+1}^\alpha) \qquad \forall \alpha \in \mathcal{T}, \forall k \in \{0, \dots, W-2\} \tag{29}$$

$$[s^\alpha, \texttt{end}(M_k^\alpha)] \ \leq_{lex} \ [s^\beta, \texttt{start}(M_k^\beta)] \qquad \forall (\alpha, \beta) \in P, \forall k \in \mathcal{K} \tag{30}$$

$$\sum_{\alpha \in \mathcal{T}} \sum_{k=0}^{W-1} \text{consumption}_r^\alpha \times \texttt{pulse}(M_k^\alpha) \leq \text{capacity}_r \qquad \forall r \in \mathcal{R} \tag{31}$$

$$s^\alpha \in \mathcal{S}, \ M_k^\alpha \subseteq [-\infty, +\infty] \qquad \forall \alpha \in \mathcal{T}, \forall k \in \mathcal{K}$$

$$\texttt{duration}(M_k^\alpha) = d_\alpha \qquad \forall \alpha \in \mathcal{T}, \forall k \in \mathcal{K}$$

$$c \in [0, +\infty]$$

Equations (27) and (28) dynamically adjust the domain of the modular representatives based on the assigned stage's capacity, $w_{s^\alpha}$. The "base" interval is $M_{W-1}^\alpha$, which is constrained to start in $[0, c]$. The intervals $M_0^\alpha$ through $M_{W-2}^\alpha$ represent the unfolded historical copies.

For instance, if a task is assigned to a standard stage ($w_{s^\alpha} = 1$), the constraints naturally confine its historical copies strictly to negative time domains (e.g., $[-c, 0]$, $[-2c, -c]$), preventing them from affecting the resource profile within the $[0, c]$ window. Conversely, if assigned to a highly parallel stage, the upper bound in Equation (28) expands, allowing older intervals to cross into the evaluation window.

Equation (29) ensures all copies maintain an exact distance of $c$. Equation (30) enforce precedences across all generated copies, while Equation (31) accumulates the resource consumption over the entire unfolded horizon. By applying this systematic unfolding bounded by $W$, the cycle-relative model guarantees that even the longest possible task is accurately captured within the cycle window, eliminating fragmentation issues while preserving strong constraint propagation.

## 4.6 Layout Optimization

We have assumed that the layout is fixed, meaning that the number of parallel stations at each stage is predetermined. However, the unfolded model is flexible enough to consider the number of parallel stations at each stage as a decision variable of the problem, $w_s, \forall s \in \mathcal{S}$. The expression $w_{s^\alpha}$ is then modeled as a variable-indexed lookup (called an *element* constraint): the number of workstations assigned to task $\alpha$ depends on the station $s^\alpha$ it belongs to.

This is a fundamental advantage of this model over the one in [14], which required generating and testing all possible layout combinations to decide the stage where a single parallel station should be placed. With this modification, the model can simultaneously compute the cycle time and the corresponding layout.

## 5 Related Work

The Simple Assembly Line Balancing Problem (SALBP) has been extensively studied, with Constraint Programming (CP) emerging as a powerful technique for its various extensions due to its flexibility in handling complex constraints. Early contributions by Schaus et al. [15, 16] introduced a model and a global constraint specifically tailored for ALBP. Bukchin and Raviv [7] provided a comprehensive evaluation of CP for several ALBP variants, demonstrating its effectiveness compared to traditional Mixed-Integer Programming (MIP), particularly when complex assignment restrictions are present. Another interesting variant solved with CP was introduced in [8] that not only assigns the tasks to the workstations but also the workers to the stations. Further research has extended CP models to more specialized scenarios. Pinarbasi et al. [13] addressed the Type-2 ALBP, which aims to minimize cycle time for a fixed number of stations, while incorporating additional assignment restrictions. The resource-constrained assembly line balancing problem (RCALBP), where tasks require multiple types of resources beyond just time, was modeled using CP by Alakaş et al. [1]. More recently, Zhang and Beck [19] explored the combination of CP with domain-independent dynamic programming for ALBP with sequence-dependent setup times, highlighting CP's competitiveness in handling temporal dependencies. Additionally, CP has been applied to modern manufacturing environments such as human-robot collaborative lines; Nourmohammadi et al. [12] developed a CP approach that accounts for different capabilities and collaborative modes between human workers and robots. Finally, the specific problem of cyclic line balancing with walking workers, which forms the basis of this work, was recently addressed by Pucel and Roussel [14] using a temporal CP formulation. The works [19, 12, 14] have in common a scheduling component in addition to the more standard bin-packing component.

Another important body of work is due to Bonfietti, Lombardi, Benini and Milano. In [3] they study cyclic RCPSP problems, proposing a CP approach based on modular arithmetic, where each activity is defined by a modular start time and an iteration (compare to our modular start time and station). A filtering algorithm for modular precedences and a dedicated search are provided. The authors view one of their main innovations to be linking the schedule period $c$ with the activity positions and capacity, while traditional approaches first fix the value of the period and then solve a non-periodic scheduling problem. Their work nevertheless makes the hypothesis that start and end of activities lie within one period $[0, c]$ (similar to our section 3) which allows them to use a "traditional" resource constraint filtering algorithm. In [4] they propose a cyclic cumulative propagator that relaxes that hypothesis (similar to our section 4), with activities that can extend over multiple periods of time. Finally [5] shows an application to instruction scheduling in chips.

# 6 Experimental Evaluation

## 6.1 Comparison of fixed layout models

In this section, we evaluate the performance of three different CP models. Table 2 provides a summary of the models, their key characteristics, and the sections where they are formally introduced.

**Table 2** Summary of the CP models evaluated in the experiments.

| Model Name | Section | Type | Key Characteristics |
|---|---|---|---|
| *Temporal* | 3.1 | Absolute + Modular | Uses absolute time and modulo-based arithmetic for periodic constraints. |
| *Cycle-Relative* | 3.2 | Cycle-Relative | Defines variables directly within the cycle window; avoids modulo operators. |
| *Unfolded* | 4.3 | Cycle-Relative | Extends the cycle-relative model to support stages with multiple parallel stations. |

A pair $(model, solver)$ is said to find a *best solution* for a instance if the solution it returns is less than or equal to the solutions obtained by all other pairs for that instance. This measure is therefore relative to the performance of the other pairs. A pair finds an *optimal solution* if it proves that the best possible solution has been reached; this measure is independent of the performance of the other pairs.

Table 3 reports, for each pair $(model, solver)$, the number of instances for which it found a best solution and the number of instances for which it proved optimality. The comparison is conducted on the first 150 instances of the J30 RCPSP benchmark from PSPlib and considers six different layout configurations.

Our experiments were conducted using OptalCP 2026.2.0.46 and IBM ILOG CP Optimizer 22.1.1.0 on an i7 4-core @3.3GHz with 32GB of RAM, with a time-limit of 300 seconds per instance. As a baseline, we include the results reported by Pucel and Roussel [14], which were obtained on a 20-core Intel Xeon CPU @2.60GHz with 62GB of RAM, also with a 300-second time-limit.

### 6.1.1 Analysis of the results

The fundamental distinction between the formulations lies in the trade-off between the complexity of modeling precedence relations and resource constraints, and the resulting impact on constraint propagation.

Without parallel stations, the cycle-relative model has a clear advantage. This is likely due to the learning branching heuristics implemented in engines such as OptalCP and CP Optimizer, as well as to the fact that no branching is required to determine the status of optional interval variables in the cycle-relative model. Despite being more generic, the unfolded model performs reasonably well, being only a few instances away from the most efficient cycle-relative model without parallel stations. It also outperforms the results reported in [14]. Unlike the model used by [14], none of the models have issues in finding feasible solutions, and all instances with the layout [2] are solved.

With parallel stations, the *unfolded* model finds better solutions than [14] in all layout configurations, and is able to prove more often that the solutions found are optimal.

**Table 3** Comparison of models and solvers over 150 J30 instances across six layout configurations. The columns report the number of instances where each model/engine pair found the best upper bound among all tested configurations ("Best solution") and the number of instances where it proved optimality within 300 seconds ("Optimal solution"). The results from Pucel and Roussel [14] are included for comparison.

| Layout | Model | Best solution | | Optimal solution | |
|---|---|---|---|---|---|
| | | CPO | OptalCP | CPO | OptalCP |
| [1] | temporal | **150** | **150** | **150** | **150** |
| | cycle-relative | **150** | **150** | **150** | **150** |
| | unfolded | **150** | **150** | **150** | **150** |
| | [14] | **150** | - | **150** | - |
| [1, 1] | temporal | 144 | **146** | 131 | **132** |
| | cycle-relative | **146** | 144 | 127 | **132** |
| | unfolded | 143 | 143 | 124 | 130 |
| | [14] | 126 | - | 132 | - |
| [1, 1, 1] | temporal | 131 | 133 | 115 | 115 |
| | cycle-relative | 141 | **144** | 111 | **118** |
| | unfolded | 130 | 139 | 104 | 115 |
| | [14] | 98 | - | 98 | - |
| [2] | unfolded | 124 | **132** | 104 | **110** |
| | [14] | 119 | - | 99 | - |
| [1, 2] | unfolded | 112 | **147** | 71 | **78** |
| | [14] | 94 | - | 64 | - |
| [2, 1] | unfolded | 113 | **149** | 71 | **77** |
| | [14] | 82 | - | 59 | - |

## 6.2 Comparison of best solution times on model from Pucel and Roussel

Pucel and Roussel report in [14] that "the solver finds its best solution in a few seconds, but fails to prove that the solution is optimal in the rest of its computation time".

To compare the best solution times, for each model and engine we record the time and value of the last solution reported within the 300 seconds limit.

- if an engine reports a strictly better solution, it is the winner
- if engines report equal solutions, the engine that found it faster is the winner

Table 3 shows that models *cycle-relative* and *unfolded* tend to find better solutions than other models, and that OptalCP tends to find better solutions than CPO using the same model. Solution times (for solutions of equal value) follow the same pattern.

In order to extend this comparison to model [14], Pucel and Roussel provided us with the 2,880 models they generated for the full set of `j30` instances (480) and the six layouts in `.cpo` format. For each model and engine, we imported the `.cpo` file, ran with a 300 seconds time limit, and recorded the value and time of the last solution reported by each engine. Among the 2880 models, 124 were removed from the comparison as neither engine could find a feasible solution, which is a known weakness of model [14] - in Pucel and Roussel tests it failed to solve 16 instances of layout [2] and 1 instance of layout [2, 1].

Table 4 shows that for model [14] OptalCP more often obtains a better solution that CP Optimizer and, in the case of ties, it most often finds the solution faster.

**Table 4** Comparison of time to find best solution on model [14]

| | OptalCP | CPO |
|---|---|---|
| strictly better solution | 776 | 7 |
| better time with same solution | 1869 | 105 |

## 6.3 Layout Optimization

A key advantage of our cycle-relative formulation is its ability to handle dynamic layouts, where the number of stations $w_s$ is itself a decision variable. In this last experiment, we compare two approaches: (i) an *exhaustive search* where each possible layout configuration is solved independently, and (ii) an *integrated optimization* approach where the layout is a decision variable within a single CP model. We allowed the model to allocate one extra parallel station to any of three stages ($\sum w_s = 4, w_s \in \{1, 2\}$), resulting in three possible layouts: $[2, 1, 1]$, $[1, 2, 1]$, and $[1, 1, 2]$.

Table 5 shows different types of behaviors that are visible in the results

- All fixed layout terminate and the integrated approach is faster
- All fixed layout terminate and the integrated approach is slower
- One fixed layout timeouts but the integrated approach is unaffected
- One fixed layout timeouts and the integrated approach timeouts too

**Table 5** Performance comparison between exhaustive layout evaluation and integrated layout optimization across three instances. For each configuration, we report the lower bound (LB), upper bound (UB), and execution time.

| Instance | Layout | LB | UB | Time (s) |
|---|---|---|---|---|
| j30 13 10 | $[2, 1, 1]$ | 27 | 27 | 5 |
| | $[1, 2, 1]$ | 27 | 27 | 8 |
| | $[1, 1, 2]$ | 30 | 30 | 26 |
| | $[*, *, *]$ | 27 | 27 | **20** |
| j30 9 8 | $[2, 1, 1]$ | 40 | 40 | **7** |
| | $[1, 2, 1]$ | 40 | 40 | **7** |
| | $[1, 1, 2]$ | 41 | 41 | **1** |
| | $[*, *, *]$ | 40 | 40 | 18 |
| j30 15 5 | $[2, 1, 1]$ | 26 | 26 | 12 |
| | $[1, 2, 1]$ | 26 | 26 | 12 |
| | $[1, 1, 2]$ | 27 | 28 | 300 |
| | $[*, *, *]$ | 26 | 26 | **49** |
| j30 14 7 | $[2, 1, 1]$ | 22 | 22 | 151 |
| | $[1, 2, 1]$ | 21 | 22 | 300 |
| | $[1, 1, 2]$ | 23 | 23 | 6 |
| | $[*, *, *]$ | 21 | 22 | **300** |

In 5 instances (`9 1`, `9 8`, `14 1`, `14 8`, `15 1`) the integrated approach is slower than the exhaustive search, the worst case difference being 3s slower. In all other cases the integrated approach is faster, the best case difference being 600s faster. In all instances the best solution of the integrated approach is equal to the best solution of the exhaustive search. However in

4 instances the lower bound of the integrated approach is worse than the lower bound of the exhaustive search.

The results highlight the efficiency of the integrated approach. This performance gain stems from the CP engine's ability to prune sub-optimal layout configurations early in the search by sharing bounds and learned heuristics across the unified search space, which is visible in instance `j30 15 5` and `j301310`.

As the number of stages and potential layouts grows, the combinatorial explosion of configurations may render the exhaustive approach intractable, making the integrated cycle-relative model the only viable path for simultaneous balancing and layout design.

## 7 Conclusion and Future Works

In this paper, we addressed the Multi-manned Assembly Line Balancing problem with Walking workers and Parallel stations (MALBWP), a complex optimization problem arising in aerospace manufacturing. We introduced a novel cycle-relative Constraint Programming (CP) formulation that defines task decision variables directly within the cycle window. This approach eliminates the reliance on modulo operators used in existing temporal models, significantly improving constraint propagation and search efficiency.

Our experimental results demonstrate that the cycle-relative model, together with its extension to parallel stations (the unfolded model), consistently outperforms the state of the art temporal formulation. Notably, the unfolded model not only provides superior solution quality and faster optimality proofs, but also enables the simultaneous optimization of the assembly line layout. This capability previously required exhaustive manual testing of configurations.

As future work, we aim to investigate the complex interactions between different CP modeling choices and engine-specific features, such as the impact of redundant constraints on lower bound computations and search speed. A deeper understanding of how capacity constraints and precedences propagate in cyclic contexts could lead to even more robust formulations. Second, we plan to benchmark our decomposition-based approach against an approach using specialized filtering algorithms, such as the global cyclic cumulative constraint [4]. This will help quantify the trade-offs between general-purpose modeling flexibility and the raw performance of dedicated global constraints. Third, the model itself could be generalized to address more varied industrial requirements. One such generalization is to relax the "identical schedule" constraint for parallel stations. In many real-world scenarios, parallel units may possess slight variations in equipment or specialized tooling, and allowing for heterogeneous task assignments could yield more efficient line configurations. Another relevant extension involves restricting workers from performing tasks across different stations. Such constraints would minimize worker travel time and cognitive load, ensuring that each operator remains within a localized work environment, which is often a requirement for safety in large-scale assembly.

## References

1 Hacı Mehmet Alakaş, Mehmet Pınarbaşı, and Mustafa Yüzükırmızı. Constraint programming model for resource-constrained assembly line balancing problem: Hm alakaş et al. *Soft Computing*, 24(7):5367–5375, 2020.

2 Nicolas Beldiceanu and Mats Carlsson. A new multi-resource cumulatives constraint with negative heights. In *International Conference on Principles and Practice of Constraint Programming*, pages 63–79. Springer, 2002.

3 Alessio Bonfietti, Michele Lombardi, Luca Benini, and Michela Milano. A constraint based approach to cyclic RCPSP. In *International Conference on Principles and Practice of Constraint Programming*, pages 130–144. Springer, 2011.

4 Alessio Bonfietti, Michele Lombardi, Luca Benini, and Michela Milano. Global cyclic cumulative constraint. In *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*, pages 81–96, Berlin, Heidelberg, May 2012. Springer Berlin Heidelberg.

5 Alessio Bonfietti, Michele Lombardi, Luca Benini, and Michela Milano. Cross cyclic resource-constrained scheduling solver. *Artificial Intelligence*, 206:25–52, 2014.

6 Nils Boysen, Philipp Schulze, and Armin Scholl. Assembly line balancing: What happened in the last fifteen years? *European Journal of Operational Research*, 301(3):797–814, 2022.

7 Yossi Bukchin and Tal Raviv. Constraint programming for solving various assembly line balancing problems. *Omega*, 78:57–68, 2018.

8 Zeynel Abidin Çil and Damla Kizilay. Constraint programming model for multi-manned assembly line balancing problem. *Computers & Operations Research*, 124:105069, 2020.

9 Philippe Laborie and Jerome Rogerie. Reasoning with conditional time-intervals. In *FLAIRS*, pages 555–560, 2008.

10 Philippe Laborie, Jérôme Rogerie, Paul Shaw, and Petr Vilím. Ibm ilog cp optimizer for scheduling: 20+ years of scheduling with constraints at ibm/ilog. *Constraints*, 23:210–250, 2018.

11 Philippe Laborie, Jérôme Rogerie, Paul Shaw, Petr Vilím, and Ferenc Katai. Interval-based language for modeling scheduling problems: An extension to constraint programming. *Algebraic Modeling Systems: Modeling and Solving Real World Optimization Problems*, pages 111–143, 2012.

12 Amir Nourmohammadi, Taha Arbaoui, Masood Fathi, and Alexandre Dolgui. Balancing human–robot collaborative assembly lines: A constraint programming approach. *Computers & industrial engineering*, 205:111154, 2025.

13 Mehmet Pinarbasi, Hacı Mehmet Alakas, and Mustafa Yuzukirmizi. A constraint programming approach to type-2 assembly line balancing problem with assignment restrictions. *Assembly Automation*, 39(5):813–826, 2019.

14 Xavier Pucel and Stéphanie Roussel. Constraint programming model for assembly line balancing and scheduling with walking workers and parallel stations. In *International Conference on Principles and Practice of Constraint Programming*, September 2024.

15 Pierre Schaus, Yves Deville, et al. A global constraint for bin-packing with precedences: Application to the assembly line balancing problem. In *AAAI*, pages 369–374, 2008.

16 Pierre Schaus et al. *Solving balancing and bin-packing problems with constraint programming.* PhD thesis, Catholic University of Louvain, Louvain-la-Neuve, Belgium, 2009.

17 Pierre Schaus, Charles Thomas, and Roger Kameugne. Implementing cumulative functions with generalized cumulative constraints. *arXiv preprint arXiv:2508.01751*, 2025.

18 Petr Vilím and Diego Olivier Fernandez Pons. Optalcp. OptalCP Scheduling Solver, 2023. Available online. URL: `https://optalcp.com`.

19 Jiachen Zhang and J Christopher Beck. Domain-independent dynamic programming and constraint programming approaches for assembly line balancing problems with setups. *INFORMS Journal on Computing*, 37(4):977–997, 2025.