A Scheduling Tool for Bridging the Gap Between Aircraft Design and Aircraft Manufacturing

Cédric Pralet, Stéphanie Roussel, Thomas Polacsek
ONERA – The French Aerospace Lab, F-31055, Toulouse, France

François Bouissière, Claude Cuiller, Pierre-Éric Dereux, Stéphane Kersuzan, Marc Lelay
AIRBUS, 1 Rond Point Maurice Bellonte, 31707 Blagnac, France

Abstract

For aircraft manufacturers, the market demand in the nowadays aeronautical industry requires a high reactivity between teams in charge of the design of the aircraft and teams in charge of its production system. One way to increase this reactivity is to help the design architects understand the way the aircraft is produced together with the bottlenecks in the manufacturing process, and to help them evaluate the impact of a design modification on the production system. This paper addresses these two needs. We formally describe the scheduling problem considered, the algorithmic approaches developed, the implemented tool, and results obtained on data from a real production line of the Airbus A320 aircraft family.

1 Context

The development process of a new aircraft program is nowadays mainly sequential: the architectural design is carried out, first from the high level definition of major components then gradually towards a very detailed design, and only then the industrial system, i.e. all the means of production of the aircraft, is considered. The development process is taking 7 to 10 years for a brand new aircraft, and can take several years for an incremental development of an existing one. This leaves a rather small latitude for adapting the production system, while the architectural design sometimes leads to rather complex operations on the production line.

Moreover, with the arrival of new airline companies on the market, there is a high demand for customized aircraft. Each customization requires an adaptation of the architectural design and consequently of the production system. With a sequential process, evaluating whether the current production system can be adapted along with the adapted design is long compared to the evolution of the market. The arrival of these new companies also puts pressure on aircraft manufacturers for increasing the production rate in order to meet the demand. As the production line represents very heavy investments, the manufacturers do not wish to modify it. A common solution is then to locally adapt the design of the aircraft to simplify some operations on the production line. To propose relevant design adaptations, the design architects must understand and analyze correctly the current production process. All these considerations highlight the need for the design and manufacturing architects to work together in a closer cooperation (Bouissière et al. 2018).

In this paper, we present a scheduling tool that pursues two objectives. The first one is to provide to the aircraft architects a better view of the current *line balancing*, *i.e.* the current organization of manufacturing tasks on the production line[1]. Such a representation is not fully available, mainly because the Airbus A320 program is not recent and relevant documents are not digitalized yet. A clearer representation can help the architects to identify manufacturing bottlenecks and propose new relevant designs. For instance, if a technical precedence between two manufacturing tasks appears on a critical path, design architects might switch to new physical systems that remove this precedence. Similarly, if an aircraft zone is saturated at some steps of the manufacturing process, it might be useful to move some physical parts to less saturated zones, *e.g.* to replace a pipe currently installed in a low-capacity zone by two pipes installed in zones where two human operators can work in parallel.

The second objective is to help the architects to evaluate the impact of design modifications on the line balancing. In fact, the current scheduling tool at the manufacturing level requires a completely specified design to compute a line balancing (for instance, how many attaches and of which type should be used for fixing a part). When the architects come up with a new design, this refinement process is long and they must wait several weeks before having a feedback from the manufacturing. On the opposite, the tool presented in this paper allows them to quickly compare several design adaptations and to focus on the most promising ones.

The paper is organized as follows. In Sect. 2, we formally model the problem to be addressed. In Sect. 3 and 4, we present the algorithmic approaches developed for helping the architects analyze and optimize an existing line balancing. We next describe a modeling based on CpOptimizer for getting optimization bounds on the line balancing (Sect. 5). Sect. 6 describes the implemented tool and the results obtained on real benchmarks coming from the production line equipping the nose section of the Airbus A320 aircraft family. We describe related works in Sect. 7, and we finally discuss some perspectives in Sect. 8.

Copyright © 2018, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

[1]*Line balancing* is used in aeronautics to designate a scheduling of the tasks and does not relate to any balancing of this scheduling.

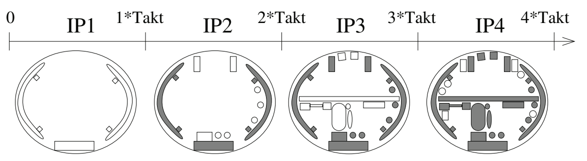


Figure 1: A 2D-view of the way an aircraft section is equipped by traversing each IP of the production line. The parts added in each IP are depicted in white.

2 Problem Modeling

The production line of the front section of the Airbus A320 is divided into several workstations, here called IPs (*Intervals of Production*) and numbered from 1 to N. Each IP is a physical area in which the aircraft section is equipped along with relevant tools and parts. In order for the section to be completely equipped, it has to go through all the IPs of the production line in a fixed order $(1, \dots, N)$. The section stays in each IP of the line for a given constant duration, denoted $\mathbf{Takt}$. When this duration is over, the section goes to the next IP of the line or is shipped to another plant if it was in the last IP. See Fig. 1 for an illustration.

The A320 front section is partitioned into a set of zones $\mathcal{Z}$. Each zone z in $\mathcal{Z}$ has a capacity K_z corresponding to the maximum number of operators that can work in z at the same time. Small zones such as the forward cargo compartment can contain at most one operator at a time ($K_z = 1$), while large zones such as the cabin can contain up to five human operators simultaneously ($K_z = 5$).

To equip the aircraft section, a set of elementary tasks $\mathcal{T}$ must be performed. Each task t in $\mathcal{T}$ is defined by a duration d_t, a zone z_t, and the number of operators q_t required by the task (with $q_t \le K_{z_t}$). We denote by $\mathcal{T}_z$ the set of tasks t such that $z_t = z$. Tasks are also constrained by an acyclic precedence relation $\mathcal{P} \subset \mathcal{T} \times \mathcal{T}$.

Tasks are partitioned into a set of elements $\mathcal{R}$ called the *routings* of the manufacturing problem. A task t belongs to a unique routing r_t, and we denote by $\mathcal{T}_r$ the set of tasks contained in routing $r \in \mathcal{R}$. A routing often groups tasks that focus on one physical part (for instance, the installation of a fan). When a human operator starts a routing in a zone, the associated physical part might be exposed and are therefore more subject to damages until the routing is finished. In order to forbid unnecessary expositions of parts, a *non-interleaving constraint* is imposed between tasks belonging to different routings (more details later on this point).

We assume that each routing r always contains two special tasks $start_r$ and end_r which must precede and follow all other tasks in the routing, respectively. Routings are constrained by a precedence relation $\mathcal{P}_\mathcal{R} \subset \mathcal{R} \times \mathcal{R}$, and $\mathcal{P}$ is extended to contain one task precedence $(end_{r_1}, start_{r_2})$ for each routing precedence (r_1, r_2) in $\mathcal{P}_\mathcal{R}$. We suppose that the extended version of $\mathcal{P}$ remains acyclic.

Finally, a manufacturing problem is defined by a tuple $(N, \mathbf{Takt}, \mathcal{Z}, \mathcal{T}, \mathcal{R}, \mathcal{P})$ containing all elements defined in the previous paragraphs.

Solution Schedule If we discard the non-interleaving constraint between tasks belonging to different routings, the problem obtained is a kind of RCPSP (*Resource Constrained Project Scheduling Problem* (Brucker et al. 1999)) where the resources are the aircraft section zones and where each manufacturing task t consumes q_t units of resource z_t.

The main difficulty is to clearly formalize the (informal) non-interleaving specification expressed by the end-users. To handle this specification, a first possible option could be to minimize the temporal dispersion of tasks associated with the same routing. The drawback of this approach is however that it can produce solutions which contain some interleaving and are not acceptable from an operational point of view. A second option could be to precompute, for each routing r taken independently, a compact non-preemptible high-level activity covering all tasks of r in a minimum amount of time, and then to schedule only these high-level activities. The issue with this approach is however that due to the decomposition of the temporal horizon into successive IPs, it can forbid standard operational scenarios in which the realization of a routing starts in a given IP, is preempted before the end of the IP, and continues in the next IP. This is why we use another option which is based on a non-interleaving property imposed on the *resource flow network* formulation of an RCPSP solution. We recall that such a network contains:

- one node labeled by t for each task t of the problem, plus two dummy nodes *src* and *sink* called the *source* and the *sink* of the schedule, respectively;
- one arc $t_1 \to t_2$ per project precedence $(t_1, t_2) \in \mathcal{P}$;
- resource flows represented as arcs $t_1 \xrightarrow{(z,\phi)} t_2$ between task nodes, where z is a zone and ϕ is the number of occupation units of z transferred from task t_1 to task t_2 (t_2 must wait for the end of t_1 to use these units).

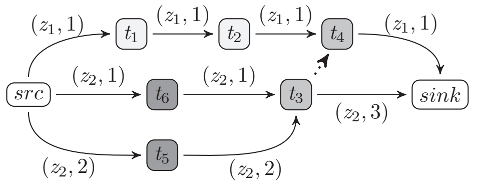
For being consistent, the resource flow network $\mathcal{N}$ must be acyclic. Moreover, the cumulated input and output flows for tasks must correspond to their resource consumptions (Eq. 1), and the output flows for the source and the input flows for the sink must cover the whole capacity of every zone (Eq. 2). All these flow transfer constraints ensure that the capacity of every zone is never exceeded.

$$\forall z \in \mathcal{Z}, \forall t \in \mathcal{T}_z, \quad \sum_{[t' \xrightarrow{(z,\phi)} t] \in \mathcal{N}} \phi = \sum_{[t \xrightarrow{(z,\phi)} t'] \in \mathcal{N}} \phi = q_t \quad (1)$$

$$\forall z \in \mathcal{Z}, \quad \sum_{[src \xrightarrow{(z,\phi)} t] \in \mathcal{N}} \phi = \sum_{[t \xrightarrow{(z,\phi)} sink] \in \mathcal{N}} \phi = K_z \quad (2)$$

An example of a small resource flow network is given on Fig. 2a: $N = 2$, $\mathcal{Z} = \{z_1, z_2\}$, $K_{z_1} = 1$, $K_{z_2} = 3$, $\mathcal{T} = \{t_1, .., t_6\}$, $\mathcal{R} = \{r_1, r_2, r_3\}$, $\mathcal{T}_{r_1} = \{t_1, t_2\}$, $\mathcal{T}_{r_2} = \{t_3, t_4\}$, $\mathcal{T}_{r_3} = \{t_5, t_6\}$; zone z_1 is the zone associated with tasks t_1, t_2, t_4 that all require 1 human operator; tasks t_3, t_5, t_6 are performed in zone z_2 and require $q_{t_3} = 3$, $q_{t_5} = 2$, and $q_{t_6} = 1$ human operator(s); $\mathcal{P}_\mathcal{R} = \emptyset$ and $\mathcal{P} = \{(t_3, t_4)\} \cup \{(start_{r_t}, t), (t, end_{r_t}) | t \in \mathcal{T}\}$. For readability reasons, Fig. 2a does not contain the nodes representing the start and end of routings, and the precedence constraints associated with them.

(a)



(b)

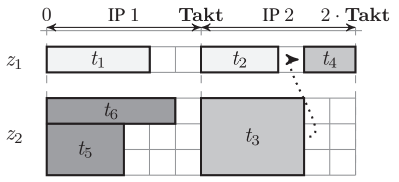


Figure 2: (a) Resource flow network; (b) line balancing

Based on the flow network formulation, we introduce a formal definition of the non-interleaving constraint in which we consider that the latter is satisfied if and only if in the resource flow graph, every path between two nodes associated with the same routing r does not traverse any node associated with another routing $r' \neq r$. For a zone with capacity 1, this definition boils down to saying that all tasks associated with a single routing r are performed successively in that zone. For a zone with capacity > 1, it expresses that zone units occupied by a task of r are either reused by other tasks of r, or never reused by r. The example on Fig. 2a satisfies the non-interleaving constraint.

A solution to the manufacturing problem also assigns an IP $ip_t \in \{1, \ldots, N\}$ to every task t. Based on the flow network and on the IP choice, the start time σ_t of each task t is then given by the maximum between the start time of its IP (equal to $(ip_t - 1) \cdot \mathbf{Takt}$), and the maximum end time of a task which must precede t owing to the resource flow network. The end date of t is given by $\sigma_t + d_t$, and a solution is said to be consistent iff every task is fully contained in its allocated IP, that is iff:

$$\forall t \in \mathcal{T},\ \sigma_t + d_t \leq ip_t \cdot \mathbf{Takt} \tag{3}$$

The line balancing corresponding to the previous example is illustrated on Fig. 2b. Tasks t_1, t_5 and t_6 start at the beginning of the first IP. Tasks t_5 and t_6 can be performed in parallel in zone z_2. Task t_2 cannot start directly at the end of task t_1 because there is not enough place left in the first IP. Instead, task t_2 starts only at the beginning of the second IP ($ip_{t_2} = 2$). Task t_4 has a precedence with task t_3 so it cannot start directly at the end of task t_2.

3 Existing Line Balancing Analysis

One key aspect in the problem is that our goal is to help architects improve the design of the aircraft *given the current line balancing*.[2] The latter cannot be completely changed for operational disruption reasons, hence there is a need to build a first schedule which is representative of the existing line balancing. To do this, the manufacturing managers do not directly give to us a resource flow network, but they provide a set of data from which we are able to extract:

- the IP associated with each task in the current line balancing; we consider here that this IP must not be changed;
- for each routing r, an ordering $\mathcal{O}_r^{\mathcal{T}}$ between tasks composing r, where tasks are ordered by increasing start time in the existing line balancing;
- for each zone z, an ordering between routings $\mathcal{O}_z^{\mathcal{R}}$ that use z, where routings are ordered by increasing start time of their first task that occupies z; from these elements, we also build a global ordering $\mathcal{O}^{\mathcal{R}}$ between routings which is consistent with all individual orderings $\mathcal{O}_z^{\mathcal{R}}$ in zones; this means that if routing r appears before routing r' in one ordering $\mathcal{O}_z^{\mathcal{R}}$, then r also appears before r' in $\mathcal{O}^{\mathcal{R}}$.

The set of data does not contain the start dates of tasks. We then use a Schedule Generation Scheme (SGS) to build a full schedule. This SGS, here called *SGS - AS IS*, starts from an empty schedule and incrementally computes the resource flow network together with the start and end times of tasks. To do this, the SGS follows global order $\mathcal{O}^{\mathcal{R}}$ (list scheduling approach) and iteratively enqueues routings and tasks at the end of the schedule.

Technically speaking, the SGS maintains, for each zone z and each IP i, a list of so-called *pending resource flows*. The latter is a list $\Phi_{z,i} = [(t_1, \phi_1), \ldots, (t_n, \phi_n)]$ where each pair (t_j, ϕ_j) represents that ϕ_j resource units of zone z are released at the end of task t_j and can be reused by future tasks placed in IP i. Pairs (t_j, ϕ_j) are ordered in the list by increasing end time of t_j. Initially, the list of pending resource flows is given by $\Phi_{z,i} = \{(s_i, K_z)\}$ where s_i is the source node associated with IP i, meaning that at the beginning of the IP the whole capacity of the zone is available. In the following, we consider that fictitious task s_i is added to the flow network and that the release time of this task is equal to the start time of the IP, that is to $(i-1) \cdot \mathbf{Takt}$.

Then, at each step, the SGS considers the next routing r in ordering $\mathcal{O}^{\mathcal{R}}$ and enqueues all tasks composing r in the schedule, following task order $\mathcal{O}_r^{\mathcal{T}}$ extracted from the input data. To insert a task t placed in zone z and IP i in the reference line balancing, we consider the current list of pending resource flows $\Phi_{z,i} = [(t_1, \phi_1), \ldots, (t_n, \phi_n)]$. We compute the smallest index $j^* \in [1..n]$ such that flows $\phi_1, \ldots, \phi_{j^*}$ suffice to cover consumption q_t, *i.e.* such that $\sum_{j \in [1..j^*]} \phi_j \geq q_t$. According to zone z, task t cannot start before time $\alpha = \sigma_{t_{j^*}} + d_{t_{j^*}}$. If t has mandatory predecessors, then t cannot start before the maximum end time β of one of these predecessors. The start time of t in IP i is chosen as $\sigma_t = \max(\alpha, \beta)$, so that both the predecessors of t are already finished at that time and the number of resource units required by t are available at that time.

Then, given the list of pending flows $\Phi_{z,i} = [(t_1, \phi_1), \ldots, (t_n, \phi_n)]$ for zone z and IP i, we compute the largest index k such that the end time of t_k is less than or equal to σ_t, that is we compute the index k such that the idle period between the end of t_k and the start of t is as small as possible. If $q_t \leq \phi_k$, then q_t resource units are transferred from task t_k to task t. Otherwise ($q_t > \phi_k$), ϕ_k

[2]The current line balancing is associated with the current state of the design.

resource units are transferred from t_k to t and resource units are also transferred from t_{k-1} to t, and so on until t receives the right number of resource units. The list of pending flows $\Phi_{z,i}$ is updated according to these modifications.

At the end of this process, all routings and tasks have been considered by the SGS and a full schedule together with a consistent resource flow network are obtained. The non-interleaving constraint is satisfied by construction since the SGS enqueues tasks by considering each routing successively. In this sense, we use a kind of *hierarchical SGS*.

The only constraint that might not be satisfied is Constraint 3, which expresses that the end time of a task must not exceed the end time of the IP in which it is placed. This constraint should never be violated by the existing line balancing, but it might be violated because the actual duration of a few operations is not consolidated yet in the ongoing digitalization process. The capacity to compute such constraint violations actually helps the architects to build a consolidated view of the existing line balancing database.

The schedule obtained, even if it is inconsistent, also helps aircraft designers understand the bottlenecks in the manufacturing process. Indeed, from the resulting schedule, we get the start time σ_t of every task t. We can also compute the minimum temporal distance between the start time of any task t and the sink node of the schedule. This temporal distance is given by the length of the longest path $D_{t,sink}$ from t to $sink$ in the precedence graph associated with the schedule. This graph contains the same arcs as the resource flow network but its arcs are labeled by the duration of tasks. The *temporal flexibility* of a task is then given by:

$$\mathit{flex}_t = makespan - (\sigma_t + D_{t,sink}) \quad (4)$$

that is by the difference between the maximum end time of a task in the schedule ($makespan = max_{t' \in \mathcal{T}}(\sigma_{t'} + d_{t'})$), and the total duration required to perform all tasks that must precede and follow t. This temporal flexibility is negative for tasks involved in a chain of precedences inducing a violation of an IP end time, it is zero for tasks which have no temporal flexibility, and positive for all other tasks.

All computations realized (pending resource flow reasoning, resource flow network construction, computation of longest paths in the precedence graph...) are actually automatically obtained based on InCELL (Invariant-based Constraint EvaLuation Library), a generic constraint-based scheduling tool (Pralet 2017). For the problem at stake, new elements have been added to this library to deal with hierarchical task orderings instead of flat task orderings.

4 Line Balancing Repair and Optimization

Line Balancing Repair The previous section explains the scheduling techniques used for bridging the gap between aircraft designers and aircraft manufacturers in terms of consolidation of the reference line balancing, the goal being to progressively get a reference schedule where tasks completely fit into IPs.

In parallel to this consolidation process, to automatically get a consistent reference schedule and start working on design optimization, one solution is to relax the constraint which forces a task to be located in a fixed IP. To do this, we use a second schedule generation scheme in which when a task is late in a given IP, it is automatically postponed to the next IP. The number of IPs required might be increased, but we get a consistent schedule. Such a relaxation is also useful for design architects to quickly simulate the line balancing obtained for a new design solution.

In the new model introduced, each task has a set of possible realization windows $\{[(i-1) \cdot \mathbf{Takt}..i \cdot \mathbf{Takt}] \mid i \in [1..N]\}$. In InCELL, such a task is automatically postponed to the first window in which it fits, and to the last window if the task cannot be consistently placed in any window. More precisely, consider a task t to be enqueued in the schedule. As before, it is possible to compute the earliest start time σ_t of t given (a) the current pending resource flows, (b) the resource consumption associated with t, and (c) the precedence constraints imposed over t. Then, either $[\sigma_t, \sigma_t + d_t]$ is fully contained in an IP and t is placed in this IP, or σ_t is postponed to the start of the next IP (if it exists).

From this, the only tasks that might finish after the end time of their IP are the tasks which are placed in the last IP. By artificially increasing the number of IPs, we can always get a schedule which satisfies all the constraints. With this new schedule generation scheme, called *SGS - Repair*, we can also decrease the duration of an IP to evaluate the impact of a production rate increase on the line balancing.

Line Balancing Optimization From a given line balancing, we can also perform optimization. We consider here that an *optimal* solution minimizes the makespan, defined as the end time of the last task ($max_{t \in \mathcal{T}}(\sigma_t + d_t)$).

As mentioned before, InCELL is able to automatically provide an output schedule given an ordering of routings and an ordering of tasks for each routing. Basically, this library uses *Constraint-Based Local Search* (Van Hentenryck and Michel 2005) and it is specifically designed to implement local search algorithms in which a given schedule is updated step by step through local moves such as activity insertions, activity removals, or activity reinsertions.

When tackling the line balancing problem based on these reinsertion capabilities, the main issue is to maintain the satisfaction of the non-interleaving constraint. To solve this issue, we developed a function called `selectBestFlexReinsertionMove`(s, r) which takes as an input a current schedule s and a routing r, and which tries to reinsert this routing (and all its tasks) at a better place in the schedule. This function works as follows:

- it removes r and all tasks covered by r from the schedule;
- after that, it tests all possible insertion positions for r in $\mathcal{O}^{\mathcal{R}}$ which are consistent with the set of precedence constraints between routings; for each insertion position of r, it quickly *estimates* the temporal flexibility flex_t (Eq. 4) that would be obtained for every task covered by r; the quality of the insertion of r at the position tested is then defined as the minimum of these temporal flexibilities;
- last, it performs the insertion of r at one position that has the best estimated insertion quality, the rationale being that such a position makes the whole schedule less tight; the exact effect of this insertion over the whole schedule

is incrementally computed; in the current version of the algorithm, the tasks covered by r are inserted in the same order as in the initial schedule, but we could also search for better orderings at this level.

Based on this new reinsertion function, we perform iterative reinsertions of routings in order to decrease the makespan. However, to avoid being stuck at a local optimum or on a plateau of the search space, we also use a tabu search mechanism. More precisely, we associate a tabu status $tabu_r$ with each routing r, expressing whether a local move on r is allowed at a given step of the search. From this, the library is also used to incrementally maintain during search expressions such as the set of routings which are not tabu and which have no temporal flexibility (called the non tabu critical routings):

$$\mathcal{R}^{critical} \leftarrow \{r \in \mathcal{R} \mid \neg tabu_r \wedge \exists t \in \mathcal{T}_r \text{ s.t. } flex_t = 0\} \quad (5)$$

On this basis, we use Algorithm 1 for improving the makespan starting from the reference line balancing s_0. The algorithm maintains a current schedule s and the best line balancing computed so far s^*. While a maximum CPU time is not elapsed, it randomly selects one routing r in $\mathcal{R}^{critical}$ (line 3), and it tries to reinsert r in the schedule using function `selectBestFlexReinsertionMove` detailed above (line 4). If the makespan of the new schedule s' obtained is worse than the makespan before the reinsertion of r, then the current schedule s is not changed and r is marked as tabu (line 5), meaning that the reinsertion of r is temporarily forbidden. Otherwise, schedule s' is accepted as the new reference schedule (line 7). If the new makespan obtained is strictly better than the best known makespan, the new schedule is recorded as the best schedule generated so far (line 8). Otherwise, if the new makespan is the same as before, routing r is marked as tabu (line 9), to avoid selecting r again and again. Last, whenever set $\mathcal{R}^{critical}$ becomes empty, the tabu status of each routing is reset to *false* (line 10). Finally, the best schedule found is returned (line 11). As the resulting line-balancing is obtained by local moves of routings in the original line balancing, this search scheme is intuitive for the end-users. Many other metaheuristics could be considered for improving the results.

Algorithm 1: `LineBalancingOptimizer`$(s_0, Tmax)$: schedule optimization through tabu search and iterative temporal flexibility improvement, from an initial schedule s_0 and for a given maximum CPU time $Tmax$

```
1  $s \leftarrow s_0$, $s^* \leftarrow s_0$
2  while CpuTimeElapsed() $< Tmax$ do
3      $r \leftarrow$ select($\mathcal{R}^{critical}$)
4      $s' \leftarrow$ selectBestFlexReinsertionMove($s, r$)
5      if makespan($s'$) $>$ makespan($s$) then $tabu_r \leftarrow true$
6      else
7          $s \leftarrow s'$
8          if makespan($s$) $<$ makespan($s^*$) then $s^* \leftarrow s$
9          else $tabu_r \leftarrow true$
10     if $\mathcal{R}^{critical} = \emptyset$ then foreach $r \in \mathcal{R}$ do $tabu_r \leftarrow false$
11 return $s^*$
```

5 Obtaining Optimization Bounds

Makespan Bound With the previous optimization approach, aircraft designers do not have any way of knowing the gap between the solution that they have and the quality of an optimal solution. To get a bound on the makespan value, one option is to use an exact solver working on a simplified (relaxed) problem. In this direction, most parts of the manufacturing problem can be easily modeled using the scheduling primitives of CpOptimizer.[3] To do this, it is first possible to define, for each task t, one integer variable ip_t corresponding to the index of the IP in which task t is realized (see Eq. 6). It is also possible to associate one temporal interval itv_t with each task t (so-called *interval variables* in CpOptimizer, defined in Eq. 7). Each such temporal interval is defined by a start time variable $startOf(itv)$, an end time variable $endOf(itv)$, and a length $lengthOf(itv)$ giving the distance between the start and the end of the interval. For each task $t \in \mathcal{T}$, the duration of itv_t is fixed to d_t in Eq. 7. Constraints 8-9 enforce that each task is placed within the IP chosen for this task.[4] Constraint 10 enforces the precedence constraints between tasks (use of the *endBeforeStart* primitive of CpOptimizer). Constraints 11 and 12 enforce that the capacity of working zones is never exceeded. For a zone z with capacity 1, Constraint 11 simply imposes a *noOverlap* constraint between intervals associated with z. For a zone with capacity > 1, Constraint 12 states that the cumulated consumption of tasks must never exceed the capacity of the zone (use of the *pulse* keyword in CpOptimizer that builds an elementary consumption profile, and of a sum of elementary profiles to get a cumulated consumption profile).

Variables:

$$\forall t \in \mathcal{T},\ \textbf{dvar int}\ ip_t\ in\ [1..N] \quad (6)$$

$$\forall t \in \mathcal{T},\ \textbf{dvar interval}\ itv_t \text{ duration } d_t \text{ in } [0..\text{MaxTime}] \quad (7)$$

Constraints:

$$\forall t \in \mathcal{T},\ startOf(itv_t) \geq (ip_t - 1) \cdot \mathbf{Takt} \quad (8)$$

$$\forall t \in \mathcal{T},\ endOf(itv_t) \leq ip_t \cdot \mathbf{Takt} \quad (9)$$

$$\forall (t_1, t_2) \in \mathcal{P},\ endBeforeStart(itv_{t_1}, itv_{t_2}) \quad (10)$$

$$\forall z \in \mathcal{Z} \text{ s.t. } K_z = 1,\ noOverlap(\{itv_t \mid t \in \mathcal{T}_z\}) \quad (11)$$

$$\forall z \in \mathcal{Z} \text{ s.t. } K_z > 1,\ \textstyle\sum_{t \in \mathcal{T}_z} pulse(itv_t, q_z) \leq K_z \quad (12)$$

Given this problem, by setting MaxTime $= N \cdot \mathbf{Takt}$, the first goal can be to minimize the makespan:

$$\text{minimize} \max_{t \in \mathcal{T}} endOf(itv_t) \quad (13)$$

The optimal objective value mk^* provides a *lower bound* on the best makespan that can be obtained for the full problem.

[3]https://www-01.ibm.com/software/commerce/optimization/cplex-cp-optimizer/

[4]In CpOptimizer, such a constraint can also be imposed by introducing, for each task t and each IP i, an *optional* interval $itvAlt_{t,i}$ whose temporal bounds are the start and end times of IP i, and by imposing an *alternative* constraint expressing that there must be exactly one IP i such that itv_t coincides with $itvAlt_{t,i}$. This alternative version provides lower quality results in our case.

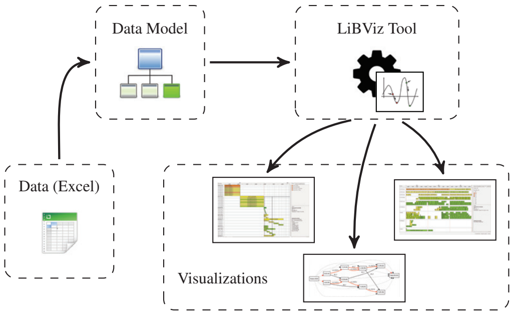


Figure 3: Process used to generate and visualize plans

Task Grouping Optimization Modeling non-interleaving aspects based on the scheduling primitives of CpOptimizer is not easy, however it is still possible to set MaxTime $= mk^*$ and to try and build solutions minimizing the dispersion of tasks associated with the same routing. The solutions obtained in this way are not operationally feasible in our case, but the approach goes in the right direction with regards to the non-interleaving constraint (use of an optimization function instead of a hard constraint). To minimize the dispersion of routings, we can introduce one additional interval variable for each routing r (Eq. 14), and use the *span* constraint of CpOptimizer to impose that this interval must contain all intervals of tasks covered by r (Eq. 15).

Additional variables:

$$\forall r \in \mathcal{R},\ \textbf{dvar interval}\ itv_r \text{ in } [0..\text{MaxTime}] \quad (14)$$

Additional constraints:

$$\forall r \in \mathcal{R},\ span(itv_r, \{itv_t \mid t \in \mathcal{T}_r\}) \quad (15)$$

On this extended problem, we can consider the new objective function given below:

$$\text{minimize} \sum_{r \in \mathcal{R}} lengthOf(itv_r) \quad (16)$$

6 Tool and Experiments

In this section, we first describe the tool developed for this study and that is currently used by Airbus architects. Then, we present results of the experiments performed.

6.1 Tool Description

For the A320 program, most of the documentation is not digitalized. This is why we first defined a data model at the interface between the design and the manufacturing architects (Polacsek et al. 2017). Compared to the model used on the manufacturing side, human operators (along with their qualification, their working hours, etc.) and tooling and logistics are not taken into account. To create data instances of this model, we have retrieved all relevant pieces of information for the existing line balancing, and merged them inside a spreadsheet. From a data model instance, the tool can use all algorithms described in Sections 3 and 4, and offers several visualizations of the generated schedule as illustrated on Fig. 3.

The LiBViz (Line Balancing Visualization) tool provides three types of visualizations:

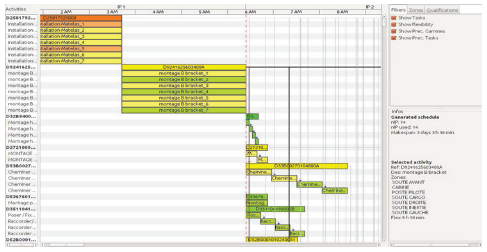


Figure 4: Gantt-based view

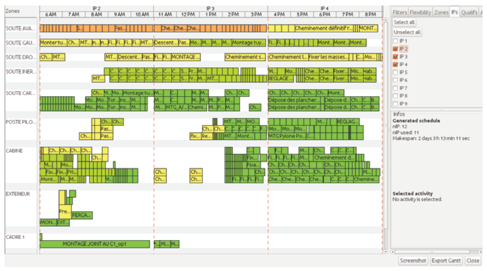


Figure 5: Resource usage view

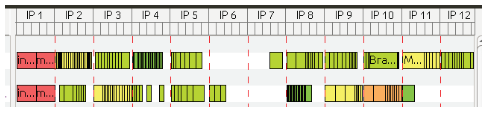


Figure 6: Resource usage view for a tight zone

- a gantt-based view (see Fig. 4) that allows to visualize the generated line balancing along with tasks and routings and associated project precedences, like in a classical Gantt diagram. The graphical interface provides filters such as the selection of specific sets of zones and/or IPs to facilitate the navigation within the line balancing. All tasks and routings are displayed with a specific color that indicates their degree of temporal flexibility in the schedule, as defined in Eq. 4: purple when the flexibility is negative, red when it is zero then gradually from orange to green when the flexibility increases. This view is particularly suited for analyzing the delays in an existing line balancing, as presented in Sect. 3;
- a resource usage view (see Fig. 5) that allows to visualize the occupation of each zone of the aircraft all along the schedule. As for the gantt-based view, filters are provided to navigate within the line balancing. This view is particularly suited for analyzing the bottlenecks associated with the occupation of zones. For instance, as illustrated on Fig. 6, if a small zone like a cargo compartment (first line in the resource view) is not used at all during IP 6, then a way of improving the makespan is to modify the ordering or the design to reduce the idle time in this zone.
- a PERT diagram (precedence graph induced by the schedule, as detailed in Sect. 2). It allows to visualize the critical path without the notion of IP. It is not interactive yet.

6.2 Benchmarks

We have worked on three real benchmarks coming from the production line of the A320 nose section:

- *Rate 50* : it corresponds to the data model used for the line balancing at the beginning of our study, when 50 nose sections were produced per month;
- *Rate 54* : it corresponds to the current data model used on the production line, with a production rate equal to 54 sections per month;
- *Rate 54 ND* : it corresponds to the data model used for benchmark *Rate 54* but with a New Design for some parts of the nose section; tasks and routings for this new design (that is not used in production yet) have been created by manufacturing experts in a few hours.

These benchmarks share the same set of zones: eight zones with a capacity between 1 and 5. Tasks are rather equally distributed amongst those zones. Specific features of these benchmarks are shown in Table 1.

To get non-private instances that are representative of our industrial benchmarks, we also performed experiments based on the classical *j90* and *j120* RCPSP instances, which contain 4 resources. Starting from these instances, we first group tasks by groups of size k (here $k = 5$) to define routings. Precedences between tasks belonging to distinct routings are translated to precedences between routings. To get closer to the real benchmarks, (a) for each group of tasks, we only keep the consumptions over a single resource, by choosing at each step the resource that contains the fewest routings; (b) resource capacities are set to 1, 2, 3, 4 respectively, and we update all resource consumptions proportionally; (c) we duplicate the model n times (here $n = 10$) to get a representative problem size; (d) we set the **Takt** value to m times (here $m = 25$) the mean duration of tasks; (e) we deduce from the most consumed resource a consistent number of stations. We also present the results associated with three representative instances obtained in this way.

6.3 Results

Following the different approaches defined in Sect. 3-5, there are 5 running modes associated with LiBViz:

- the *SGS - AS IS* mode reproduces the existing line balancing based on the techniques presented in Sect. 3;
- the *SGS - Repair* mode postpones the tasks that are late, as presented in Sect. 4;
- the *Local Search* mode performs line balancing optimization based on Algorithm 1 presented in Sect. 4;
- the *CpOptimizer (Mk,Gr)* mode uses CpOptimizer 12.5 along with the model defined in Sect. 5, that first optimizes the makespan and then the grouping criterion. For memory issues, all task durations are divided by 10 for this mode. The computation time is written $x + y$ where x and y are the times for the makespan minimization phase and the task grouping optimization phase, respectively;
- the *CpOptimizer (Gr,Mk)* mode also uses CpOptimizer, but it considers the grouping criterion as strictly more important than the makespan. More precisely, this mode aggregates the two criteria as follows: minimize MaxTime $\cdot \sum_{r\in\mathcal{R}} lengthOf(itv_r) + \max_{t\in\mathcal{T}} endOf(itv_t)$.

All the experiments have been run on a four-Xeon 2.80GHz processors with 8GiB of RAM. Results of experiments are presented in Table 1.[5]

As expected, the SGS - Repair mode uses more IPs than the other ones. Nevertheless, it runs in less than a second and allows to repair a line balancing with very simple and realistic moves: a task that cannot be performed in one IP is performed in the next one.

CpOptimizer (Mk,Gr) finds the optimal makespan for the Airbus benchmarks in less than one minute. However, it cannot find the optimal value for the second criterion with a timeout of 300 seconds. For benchmark *Rate 54 ND* and the RCPSP benchmarks, this mode is unable to find a solution in the second phase of the search: the result in the table is the value of the grouping criterion after the minimization of the makespan. CpOptimizer (Gr,Mk) clearly improves the grouping criterion while being close to the best makespan for the Airbus benchmarks. In the case of the RCPSP instances, CpOptimizer cannot get low values of the grouping criterion for the two biggest instances in 300 seconds.

With a timeout equal to one minute, the Local Search approach gives results of high quality. The makespan of the obtained solutions is only one hour longer than the makespan lower bound provided by CpOptimizer (Mk,Gr), while the task grouping criterion is much better than with the two variants of CpOptimizer, essentially thanks to the schedule generation scheme used that does not interleave tasks.

Concerning the analysis done on the design side, the results highlight that the makespan is significantly improved with the new design (ND) as there is a need for 10 IPs instead of 11 IPs (5 hours less on the production line). In fact, the new design allows to remove several operations from the tightest zones of the aircraft. Following this result, the design architects can pursue the update of the design in this direction. On the manufacturing side, architects can also foresee what could be a line balancing according to this new design. When the optimization is realized through Local Search, they can trace all the local moves performed to optimize the schedule. This way, they can validate the moves according their feasibility in the real production line.

7 Related Work

Co-engineering is not a new concept (Shenas and Derakhshan 1994), notably in the automotive industry (Göpfert and Schulz 2013) or in the space industry (Bandecchi et al. 2000), but it is still a challenge in the aeronautics industry as the manipulated objects are quite complex and produced in low quantities compared to the automotive industry, and most operations are still handled by human operators (Pardessus 2004). In the context of co-engineering in aeronautics, using scheduling tools for evaluating design impacts on a line balancing is, from our knowledge, novel. In terms

[5]For the SGS - AS IS mode, the makespan does not take into account the delays, which can be significant on some benchmarks.

Benchmark	**Takt**	$\lvert\mathcal{R}\rvert$	$\lvert\mathcal{T}\rvert$	$\lvert\mathcal{P}\rvert$	Mode	nIPs	Makespan	Delays	Grouping Crit.	Time (s)
Rate 50	324	317	780	2324	SGS - AS IS	10	2d 3h 36min	yes	1w 5d 2h 32min	< 1
					SGS - Repair	12	2d 14h 5min	no	1w 5d 19h 4min	< 1
					Local Search	11	2d 7h 11mins	no	1w 5d 3h 24min	60
					CpOpt. (Mk,Gr)	11	2d 6h 6min	no	2w 6d 14h 30min	15 + 300
					CpOpt. (Gr,Mk)	11	2d 6h 26min	no	2w 3d 22h 47min	300
Rate 54	300	387	915	2669	SGS - AS IS	12	2d 8h 12min	yes	1w 6d 1h 31min	< 1
					SGS - Repair	12	2d 11h 52min	no	1w 6d 11h 53min	< 1
					Local Search	11	2d 3h 3min	no	1w 6d 6h 5min	60
					CpOpt. (Mk,Gr)	11	2d 2h 4min	no	3w 3d 9h 45min	17 + 300
					CpOpt. (Gr,Mk)	11	2d 4h 30min	no	2w 3d 12h 58min	300
Rate 54 ND	300	341	849	2478	SGS - AS IS	12	2d 8h 12min	yes	1w 4d 8h 0min	< 1
					SGS - Repair	12	2d 11h 10min	no	1w 4d 15h 3min	< 1
					Local Search	10	1d 22h 33min	no	1w 4d 12h 35min	60
					CpOpt. (Mk,Gr)	10	1d 21h 45min	no	7w 0d 8h 52min	33 + 300
					CpOpt. (Gr,Mk)	10	2d 0h 53min	no	2w 4d 15h 13min	300
j90_1_1	822	190	470	280	Local Search	6	2d 22h 24min	no	3w 6d 23h 26min	60
					CpOpt. (Mk,Gr)	5	2d 21h 24min	no	6w 4d 11h 12min	300+300
					CpOpt. (Gr,Mk)	6	3d 9h 18min	no	2w 2d 11h 6min	300
j90_28_9	762	190	840	800	Local Search	17	1w 1d 19h 54min	no	2w 0d 8h 0min	60
					CpOpt. (Mk,Gr)	16	1w 1d 0h 54min	no	9w 4d 1h 12min	300+300
					CpOpt. (Gr,Mk)	16	1w 1d 9h 30min	no	9w 2d 20h 12min	300
j120_9_6	864	250	900	720	Local Search	11	6d 10h 48min	no	2w 1d 38h 24min	60
					CpOpt. (Mk,Gr)	10	5d 14h 18min	no	6w 0d 16h 12min	300+300
					CpOpt. (Gr,Mk)	10	5d 17h 42min	no	8w 0d 10h 24min	300

Table 1: Results of experiments. Columns are defined as follows: name of the benchmark (*Benchmark*), **Takt** value in minutes (**Takt**), number of routings ($|\mathcal{R}|$), number of tasks ($|\mathcal{T}|$), number of precedence constraints ($|\mathcal{P}|$), name of the applied algorithm (*Mode*), number of IPs in the result schedule (*nIPs*), makespan[5] of this schedule (*Makespan*), boolean indicating whether the schedule contains tasks with delays, i.e. with a negative temporal flexibility (*Delays*), value of the task grouping criterion as defined in the CpOptimizer model (*Grouping Criterion*), and CPU time needed for computing the schedule (*Time*). In the results, "a w b d c h d min' should be read as "a weeks, b days, c hours and d minutes".

of scheduling, this corresponds to looking for changes in the input data to improve the quality of the schedule obtained.

However, there is a wide literature for the optimization of a line balancing from a manufacturing point of view, as presented in the survey (Battaïa and Dolgui 2013). In this paper, we are more at a strategic decision level, which is why we keep a rather simple model of the manufacturing (without human operators or multiple lines).

From a scheduling point of view, if we discard non-interleaving constraints, the problem considered when IPs are fixed is a kind of RCPSP. When task IPs can be changed, we are closer to works on scheduling with time windows available for realizing activities (Brucker and Knust 2000). In another direction, we are not aware of RCPSP extensions considering hierarchical tasks (such as routings decomposed into basic tasks) coupled with non-interleaving constraints between tasks. Several existing approaches consider the minimization of the temporal extent of a set of related tasks, for instance to group multiple observations of the same target to get similar observation conditions (Colomé et al. 2012). However, this does not necessarily produce solutions which are operationally feasible in our case. Interleaving concerns are also present in SHOP2, a planner for HTNs (*Hierarchical Task Networks* (Nau et al. 2003)). Basically, SHOP2 plans for tasks in the same order as they will be executed, and it has an anti-interleaving mechanism which can be used to force the addition of all subtasks of an abstract task before considering the next abstract task. Our schedule generation scheme uses similar principles, however we adopt a scheduling approach in which we directly reason about tasks and resources instead of actions and states.

8 Conclusion

In this paper, we describe a scheduling tool for helping the design architects and the manufacturing managers to work in a closer loop. This tool is currently used and it already helps the design architects to better understand the manufacturing process The compact representation it offers allowed the correction of dozens of mistakes in the current representation of the line balancing. Our approach also reduces the time required for evaluating a new design: for the new design mentioned in the paper, it took two days to define the updated routings and tasks (and their features), and our tool then provided answers within minutes. For the end-users, our tool also has visualization and navigation capabilities that allow to grasp the content of a schedule containing almost one thousand tasks.

On the scheduling side, we will have to consider new types of constraints. In particular, there is a need to handle maximum temporal distance constraints between two tasks or two routings, for instance to express that two tasks focusing on the same part should be performed within a given amount of time. We could also consider constraints stipulating that some tasks neutralize a zone for a given duration.

References

Bandecchi, M.; Melton, B.; Gardini, B.; and Ongaro, F. 2000. The ESA/ESTEC Concurrent Design Facility. In

Proc. of the 2nd European Systems Engineering Conference (EuSEC'00), 329–336.

Battaïa, O., and Dolgui, A. 2013. A Taxonomy of Line Balancing Problems and their Solution Approaches. *International Journal of Production Economics* 142(2):259 – 277.

Bouissière, F.; Cuiller, C.; Dereux, P.; Kersuzan, S.; Polacsek, T.; Pralet, C.; and Roussel, S. 2018. Co-engineering in Aeronautics? The A320 Forward Section Case Study. In *Proc. of the 9th European Congress Embedded Real Time Software And Systems (ERTS'18)*.

Brucker, P., and Knust, S. 2000. Resource-constrained Project Scheduling and Timetabling. In *Proc. of the 3rd International Conference on Practice and Theory of Automated Timetabling (PATAT'00)*, 277–293.

Brucker, P.; Drexl, A.; Möring, R.; Neumann, K.; and Pesch, E. 1999. Resource-constrained Project Scheduling: Notation, Classification, Models, and Methods. *European Journal of Operational Research* 112(1):3–41.

Colomé, J.; Colomer, P.; Guardia, J.; Ribas, I.; Campreciós, J.; Coiffard, T.; Gesa, L.; Martinez, F.; and Rodler, F. 2012. Research on Schedulers for Astronomical Observatories. In *Proc. of SPIE, Observatory Operations: Strategies, Processes, and Systems IV*, volume 8448.

Göpfert, I., and Schulz, M. 2013. *Logistics Integrated Product Development in the German Automotive Industry: Current State, Trends and Challenges*. Springer Berlin Heidelberg. 509–519.

Nau, D.; Au, T.-C.; Ilghami, O.; Kuter, U.; Murdock, W.; Wu, D.; and Yaman, F. 2003. SHOP2: An HTN Planning System. *Journal of Artificial Intelligence Research* 20:379–404.

Pardessus, T. 2004. Concurrent Engineering Development and Practices for Aircraft Design at Airbus. In *Proc. of the 24th International Congress of Aeronautical Sciences (ICAS'04)*.

Polacsek, T.; Roussel, S.; Bouissière, F.; Cuiller, C.; Dereux, P.; and Kersuzan, S. 2017. Towards Thinking Manufacturing and Design Together: An Aeronautical Case Study. In *Proc. of the 36th International Conference on Conceptual Modeling (ER'17)*, 340–353.

Pralet, C. 2017. An Incomplete Constraint-Based System for Scheduling with Renewable Resources. In *Proc. of the 23rd International Conference on Principles and Practice of Constraint Programming (CP'17)*, 243–261.

Shenas, D. G., and Derakhshan, S. 1994. Organizational Approaches to the Implementation of Simultaneous Engineering. *International Journal of Operations & Production Management* 14(10):30–43.

Van Hentenryck, P., and Michel, L. 2005. *Constraint-based Local Search*. MIT Press.