Propagation of Idle Times Costs for Fixed Job Scheduling

1st Wang Ruixin
ENAC Lab
Toulouse, France
ruixin.wang@enac.fr

2nd Barnier Nicolas
ENAC Lab
Toulouse, France
nicolas.barnier@enac.fr

***Abstract*—We present a new global constraint to propagate idle times costs for the Fixed Job Scheduling (FJS) problem, in particular to minimize their variance so as to optimize the robustness of solutions w.r.t. schedule deviations. The propagation of this constraint is based on the computation of the shortest path in the compatibility directed acyclic graph of each resource to obtain an exact lower bound. It ensures Bound Consistency on the resource cost in polynomial time, as well as the filtering of the resource variables associated with compatible tasks.**

We show on tailored FJS problems and real instances of the airport Gate Allocation Problem (a variant of the FJS problem) that this new constraint provides significant improvements in terms of number of backtracks and computation time, up to orders of magnitude in some cases.

***Index Terms*—fixed job scheduling, robustness, optimization constraint**

Introduction

Fixed Job Scheduling (FJS) [1] is a well-known resource allocation problem with many applications where jobs (or *tasks*) with fixed start and end times are to be processed on different machines (or *resources*). Overlapping tasks must execute on different resources and the set of possible resources for a given task may be restricted. Standard objectives for the FJS problem include the maximization of the number of (possibly weighted) processed tasks or the minimization of the cost associated with assigned resources.

Normally, a task can only be executed according to a predefined schedule, but for applications like the Gate Allocation Problem (GAP) [2] which aims at assigning airport stands to aircraft, the robustness of the plan should be optimized to absorb potential delays and avoid costly disruption. Despite its practical importance, the research on the robustness of solutions to the FJS problem or the GAP is very limited.

Following the robustness objective defined by [2] that minimizes the variance of idle times to balance and spread them over time and resources, we show that this kind of model can be efficiently solved with Constraint Programming (CP). Our main contribution is a new optimization constraint on each resource that achieves Bound Consistency (BC) over its idle times costs in polynomial time, as well as the filtering of the allocation variables associated with possible tasks of the resource.

More standard objectives for the GAP includes the optimization of passengers walking distance [3], the allocation of terminal gates over apron stands and minimization of the number of towing movements [4]. CP was also used to implement a solver that minimizes the allocation costs defined by airlines and the airport manager [5]. Other search methods like Genetic Algorithms have been used as well in [6] to improve the MIP approach presented in [2].

The rest of the paper is organized as follows. We first present a mathematical formulation of the problem in section I, then describe a corresponding CP model in section II and the propagation of our new optimization constraint for idle times costs in section III. Section IV presents the results of our experiments on the FJS problem and the GAP with tailored and real data to assess the performances of our approach. The conclusion and further works are discussed in the last section.

I. Fixed Job Scheduling

The scheduling of tasks with fixed start and end times on non-identical[1] resources is a versatile NP-complete problem [8] which occurs in many applications beside the GAP, like processors scheduling or staff rostering. Though various objectives can be associated with this problem, our approach is dedicated to optimize resource costs based on the idle times to ensure the robustness of solutions w.r.t. delays.

We present in the following sections the integer model used in our study (whereas more classical OR approaches rather consider boolean variables [2]).

A. Instance

An instance of the FJS problem is defined by:

- $\mathcal{T} = \{t_1, \ldots, t_n\}$ a set of n tasks, with $\forall t_i \in \mathcal{T}$:
  - t_i^s and t_i^e the start and end times of task t_i;
  - $\mathcal{R}_i \subseteq \mathcal{R}$ a set of compatible resources on which the task can be executed.
- $\mathcal{R} = \{r_1, \ldots, r_m\}$ the set of m resources, with $\forall r_j \in \mathcal{R}$:
  - r_j^s and r_j^e the opening and closing times of resource r_j. However, except when mentioned otherwise, all resources are considered available during the same period in the following, therefore $\forall j, r_j^s = r^s$ and $r_j^e = r^e$.

[1]Note that with identical resources (i.e. all tasks can be assigned to any resource), this problem becomes equivalent to the coloring of an interval graph, which is polynomial [7].

– $\mathcal{T}_j = \{t_i \in \mathcal{T} \text{ s.t. } r_j \in \mathcal{R}_i\}$ the set[2] of compatible tasks that can be executed on resource r_j.

For conciseness, we also define the *duration* function d, "overloaded" on the following sets:

- $\mathcal{T} \mapsto \mathbb{N}$ the duration of a task: $d(t_i) = t_i^e - t_i^s$;
- $2^{\mathcal{T}} \mapsto \mathbb{N}$ the total sum of the durations of a subset of (possibly overlapping) tasks: $d(\mathcal{T}') = \sum_{t_i \in \mathcal{T}'} d(t_i)$;
- $\mathcal{R} \mapsto \mathbb{N}$ the availability of a resource: $d(r_j) = r_j^e - r_j^s$;
- $2^{\mathcal{R}} \mapsto \mathbb{N}$ the total sum of the availability of a set of resources: $d(\mathcal{R}') = \sum_{r_j \in \mathcal{R}'} d(r_j)$.

B. Decision Variables

A solution to the fixed tasks scheduling problem consists in assigning a resource to each task while satisfying the constraints described in the next section. We define the set of decision variables associated to the tasks of $\mathcal{T}$:

$$\mathcal{X} = \{x_i \in \{j \text{ s.t. } r_j \in \mathcal{R}_i\}, \forall t_i \in \mathcal{T}\}$$

C. Constraints

The only constraints of this essential version of the problem are the non-overlapping of the tasks scheduled on the same resource. As tasks execution times are fixed, we require that overlapping tasks are assigned to different resources:

$$\forall i \neq i',\ [t_i^s, t_i^e[\cap [t_{i'}^s, t_{i'}^e[\neq \varnothing \Rightarrow x_i \neq x_{i'} \tag{1}$$

However, specific applications of the fixed tasks scheduling problem like the GAP are often described with many additional hard and soft constraints to account for operational requirements (e.g. a large aircraft might occupy two adjacent stands) or user preferences (e.g. use of terminal gates rather than remote apron stands).

D. Cost

Many different kind of costs can be taken into account to optimize the allocation of fixed tasks on non-identical resources. For our target application, the GAP, one of the most crucial objectives is the robustness of the schedule as air traffic operations can be burdened by many uncertainties such as late arrival or departure. To be able to absorb those possible delays, [2] proposes to minimize the variance of *idle times*, which tends to balance them over resources and time while allowing necessary short or large pauses required by some instances.

Since their mean is constant for our problem (as the overall duration of tasks and availability of resources are constant, and all tasks must be scheduled), minimizing the variance of idle times amounts to minimizing the sum of their squares:

$$\text{cost} = \sum_{r_j \in \mathcal{R}} c_j$$

where c_j is the cost of a single resource r_j:

$$c_j = (t^s_{\text{first}(r_j)} - r_j^s)^2 + \sum_{t_i \in \mathcal{T}_j \text{ s.t. } x_i = j} (\text{next}(t_i) - t_i^e)^2$$

[2]Redundantly defined from $\mathcal{R}_i$ to simplify notations in the next sections.

with:

- $\text{next}(t_i) = r_j^e$ if t_i is the last task assigned on resource r_j or the start time of the task immediately following t_i on r_j otherwise;
- $\text{first}(r_j)$ is the index of the first task scheduled on resource r_j.

However, our approach could be generalized to any objective that aggregates resource costs defined as a positive additively separable function of its idle times (or even the start and end times of its tasks).

II. Constraint Programming Model

We present in this section how the previous formulation of the FJS problem translates in a CP context, similarly to the approach described in [5]. We first describe how all maximal cliques of the interval graph can be easily computed to efficiently model the mutual exclusion of overlapping tasks, then how symmetry on resources and tasks can be broken, eventually focusing on idle times costs with the definition of a new global constraint named *idlecost*, the propagation rules of which are detailed in section III.

A. Constraints on Maximal Cliques

To specify the mutual exclusion constraints 1 on unitary resources, it would be sufficient to model each of them directly with a binary difference constraint. However, stronger propagations can be obtained with the well-known *all-different* global constraint on cliques of the associated interval graph, as noted in [5]. Only constraints corresponding to all distinct *maximal* cliques need to be added, as any other clique would be subsumed by a maximal one.

In the general case, computing the *maximum* clique of an arbitrary graph is NP-Hard. However, [7] mentions how all maximal cliques (including the maximum one) of an interval graph of n vertices can be generated by a sweep algorithm in $\Theta(n \log n)$, or even linear time $\Theta(n)$ if the endpoints of the intervals are already sorted.

Indeed, once the $2n$ endpoints of the tasks are sorted by ascending order, all maximal cliques of the corresponding intersection graph can be easily detected by maintaining the list of overlapping tasks at each endpoints. While scanning the sorted list of endpoints, the corresponding task is added whenever its left endpoint is encountered and removed upon reaching its right endpoint.

For each minimal size of the maintained list (except when it is empty), a new clique can be started with all currently overlapping intervals, collecting all subsequently opened new intervals until a local maximum is reached, which corresponds to a new maximal clique. All-different constraints can then be posted in linear time on all maximal cliques to improve propagation and allow global reasoning over multiple resources.

However, a collection of all-different constraints that share subsets of variables and propagate independently may keep inconsistent values w.r.t. their conjunction. [5] mentions the use of the non-overlap global constraint *diffn* introduced by [9],

which was generalized to multiple dimensions and new propagation rules in [10]. Nevertheless, it has been proved in [11] that the corresponding decision problem is NP-complete and it was deemed efficient enough to use a simple collection of all-different constraints in our model because the instances of our target application, the GAP, are relatively easy to solve w.r.t. the allocation problem, and because the objective, which is hard to optimize (see IV-B), does not depend on the makespan but on the even and balanced distribution of idle times.

B. Symmetry

Depending on the instance at hand, allocation problems may exhibit symmetries on equivalent resources and equivalent tasks. For the GAP, the former is much more frequent on real instances: many adjacent stands in a terminal share the same set of characteristics, whereas equivalent flights with the same aircraft type, company and dates are seldom. Breaking such symmetries often lead to drastic speed-up while proving optimality (see section IV).

1) Resources: Resources that have exactly the same set of possible tasks can be exchanged while preserving the admissibility and optimality of solutions. Therefore, whenever a (yet) unused resource is assigned to a task, all other unused equivalent resources should be removed from its domain upon backtracking.

To this end, all the equivalence classes $\mathcal{C}_k$ of resources (with at least two elements) are computed before search. Before each assignment of a task i to an unused resource r_j , we check if other unused resources remain in the corresponding class $\mathcal{C}$ to add the following goal:

$$(x_i = r_j) \vee (x_i \notin \{j' \text{ s.t. } r_{j'} \in \mathcal{C} \wedge \text{unused}(r_{j'})\})$$

with predicate $\text{unused} : \mathcal{T} \mapsto \mathbb{B}$ indicating whether a resource is still free of task or not.

2) Tasks: Two tasks i and i' that have the same compatible resource set and arrival and departure dates can also be exchanged while preserving solutions. So they can be arbitrarily ordered with the following constraint:

$$\forall i \neq i',\ (\mathcal{R}_i = \mathcal{R}_{i'} \wedge t^s_i = t^s_{i'} \wedge t^e_i = t^e_{i'}) \Rightarrow (x_i < x_{i'})$$

However, there is no occurrence of equivalent flights in the GAP data set used to assess the performance of our approach (see section IV-B).

C. Idle Times Cost

Modeling idle times costs with standard CP reification constraints would be cumbersome and inefficient. Therefore, we introduce a new global optimization constraint *idlecost* to tighten the bounds of the cost of a single resource and the domains of its possible tasks. In the following sections, we define the semantic of this constraint and static bounding schemes for the overall sum of all idle costs. Propagation rules for the *idlecost* constraint are then discussed in section III.

1) Optimization Constraint for a Single Resource: We introduce the *idlecost* optimization constraint on a single resource $r_j \in \mathcal{R}$ to tighten its idle times cost and filter the resource variables of its set of possible tasks $\mathcal{T}_j$:

Definition 1 (The idlecost *Constraint):* Let $\mathcal{X}_{\mathcal{T}_j} \subseteq \mathcal{X}$ be the set of resource variables associated to $\mathcal{T}_j$ and $f : \mathbb{N} \mapsto \mathbb{N}$ a non-decreasing elementary cost function that represents the contribution of a single idle time interval. The optimization constraint $\text{idlecost}(r_j, \mathcal{T}_j, \mathcal{X}_{\mathcal{T}_j}, f, c_j)$ is satisfied iff:

- $c_j = f(t^s_{\text{first}(r_j)} - r^s_j) + \sum_{t_i \in \mathcal{T}_j \text{ s.t. } x_i = j} f(\text{next}(t_i) - t^e_i)$ with the first and next functions defined as in section I-D;
- the set of mutual exclusion constraints 1 of section I-C restricted to variables of $\mathcal{X}_{\mathcal{T}_j}$ is satisfied.

2) Static Lower Bound: A static lower bound for the overall sum of the squares of idle times can be easily computed as it is minimal when all idle times have the same size and are evenly distributed among all resources. For n tasks to be executed on m resources, there are exactly $n+m$ idle time periods (taking the first and last idle times of every resource into account). Therefore, we can compute the following global lower bound for the objective:

$$\text{cost} \geq (n+m) \left\lfloor \frac{d(\mathcal{R}) - d(\mathcal{T})}{n+m} \right\rfloor^2 \tag{2}$$

In many instances, the availability is identical for all resources, i.e. $r^s_j = r^s$ and $r^e_j = r^e$, $\forall j \in [1, m]$. We can then benefit from the computation of the first and last maximal cliques $\mathcal{K}^s$ and $\mathcal{K}^e$ mentioned in section II-A to take into account necessary idle times before the execution of the tasks of the first clique and after the end of the tasks of the last one:

$$\text{cost}_{\mathcal{K}} = \sum_{t_i \in \mathcal{K}^s} (t^s_i - r^s)^2 + \sum_{t_i \in \mathcal{K}^e} (r^e - t^e_i)^2$$

As in 2, we can compute the duration of idle times corresponding to the lower bound of the objective for the tasks that does not belong to the first or last cliques, i.e. the total amount of idle times divided by the number of pauses $n+m-k$ with $k = |\mathcal{K}^s \cup \mathcal{K}^e|$ (as the cliques might intersect if some tasks span from the first one to the last one):

$$\text{idle}_{\text{LB}} = \left\lfloor \frac{d(\mathcal{R}) - d(\mathcal{T}) - \text{idle}_{\mathcal{K}}}{n+m-k} \right\rfloor$$

with $\text{idle}_{\mathcal{K}} = \sum_{t_i \in \mathcal{K}^s} (t^s_i - r^s) + \sum_{t_i \in \mathcal{K}^e} (r^e - t^e_i)$. Therefore, we obtain the following tighter lower bound:

$$\text{cost} \geq \text{cost}_{\mathcal{K}} + (n+m-k)\text{idle}^2_{\text{LB}} \tag{3}$$

which can help reduce the optimality gap and prove solutions.

3) Static Upper Bound: For instances with identical availability for all resources, a simple upper bound could also be obtained by saturating the first resources except the last non-empty one (where tasks are stacked at the beginning). The corresponding bound would then be:

$$\text{cost} \leq m' d(r)^2 + (d(r) - k)^2 \tag{4}$$

with $d(r) = r^e - r^s$, $m' = m - \left\lceil \frac{d(\mathcal{R}) - d(\mathcal{T})}{d(r)} \right\rceil$ the number of empty resources and $k = (d(\mathcal{R}) - d(\mathcal{T})) \mod d(r)$ the time

taken by the tasks scheduled on the last non-empty resource. But this bound is very loose and not really significant to help close the optimality gap.

III. Propagation of Idle Times Costs

We present in this section a new polynomial algorithm to achieve *Bound Consistency* (BC) on the cost of a single resource for the *idlecost* constraint: after its execution, a partial assignment can be extended to the possible tasks of the resource such that the cost can be assigned either its lower or upper bound.

After describing a tight upper bound and a naive relaxed lower bound in section III-A, we introduce in more details our algorithm to achieve BC on the lower bound in section III-B. We define the following notations, used in these sections, w.r.t. a resource r_j:

- $\overline{\mathcal{T}_j} = \{t_i \in \mathcal{T}_j \text{ s.t. } j \in \mathrm{dom}(x_i)\}$ its possible tasks;
- $\underline{\mathcal{T}_j} = \{t_i \in \mathcal{T}_j \text{ s.t. } x_i = j\}$ its necessary tasks;
- ub_j the upper bound induced by $\underline{\mathcal{T}_j}$;
- lb_j the lower bound induced by $\underline{\mathcal{T}_j}$ and $\overline{\mathcal{T}_j}$;

with $\mathrm{dom}(x_i)$ the set of currently possible values for x_i.

A. Bounds Based on the Union of Tasks

In the next sections, we present simple algorithms to compute the upper bound and an approximation of the lower one.

1) Assignment and Upper Bound: The upper bound of a single resource can be computed thanks to the set of necessary tasks $\underline{\mathcal{T}_j}$ already assigned to r_j. If the cost was linear, it would be enough to maintain the sum of the durations of the tasks of $\underline{\mathcal{T}_j}$ and subtract it from the availability of the resource to obtain a tight upper bound on the cost: $d(r_j) - \sum_{t_i \in \underline{\mathcal{T}_j}} d(t_i)$. Upon assignment of a task t_i, we would just have to subtract its duration to incrementally maintain the bound in constant time: $\mathrm{ub}_j \leftarrow \mathrm{ub}_j - d(t_i)$.

Though for a positive *non-linear* cost like the sum of the *squares* of idle times, we have to thoroughly maintain the set of idle intervals to be able to determine which one will be impacted by the new assigned task t_i, and compute the new upper bound incrementally. Once the idle interval $[a, b[$ is identified, the upper bound is updated accordingly:

$$\mathrm{ub}_j \leftarrow \mathrm{ub}_j - (b-a)^2 + (t_i^s - a)^2 + (b - t_i^e)^2 \qquad (5)$$

As $\underline{\mathcal{T}_j}$ only contains disjoint tasks, a simple *binary search tree* could be used to maintain the set of intervals and update the upper bound in logarithmic time upon assignment, provided that the data structure be easily "backtrackable". However, considering the low number of tasks associated with each resource in the instances of our target application (the GAP, cf. section IV-B), we only implemented a simple linear algorithm in the solver presented in section IV to avoid the likely overhead costs of a more sophisticated data structure.

Fig. 1. The resource and tasks of example 1 with unbounded lower bound approximation ratio.

2) Removal and Lower Bound: The removal is not as straightforward as the assignment because tasks are not disjoint within $\overline{\mathcal{T}_j}$, so the removal of one task from a resource r_j does not entail that a new necessary idle time appears on r_j. However, by maintaining the union of intervals corresponding to the tasks of $\overline{\mathcal{T}_j}$, necessary idle times can be detected whenever $\overline{\mathcal{T}_j}$ does not span the entire availability. The lower bound can then be updated as follows upon a task removal:

$$\mathrm{lb_j} \leftarrow \sum_{k=1}^{l} (h_k^e - h_k^s)^2 \qquad (6)$$

with $\overline{\mathcal{H}_j} = [r_j^s, r_j^e[\setminus \bigcup_{\forall t_i \in \overline{\mathcal{T}_j}} [t_i^s, t_i^e[= \bigcup_{k=1}^{l} [h_k^s, h_k^e[$ the necessary "holes" (possibly $\varnothing$, in which case $\mathrm{lb}_j \leftarrow 0$).

A *segment tree* data structure [12] can be used to maintain the tasks intervals, and augmented to also aggregate the upper bound of each subtree, including the root node which holds the bound for the whole resource, in $\Theta(k \log n)$ with n the number of intervals and k the number of newly discovered necessary idle times. However, for reasons already mentioned in the previous section, our solver only use a naive (but simple) quadratic algorithm.

B. Tight Lower Bound

Among the bounds presented in the previous section, only the upper one is tight, because the assignment of a new possible task on a resource necessarily decreases the cost. In contrast, the lower bound is not, as the union of tasks doesn't take into account temporal conflicts. Therefore, the actual lower bound could be arbitrarily larger as, for example, with the following set of n possible tasks on a given resource depicted in figure 1:

Example 1 (Overlapping tasks with unbounded lower bound approximation ratio): $\forall i \in [0..\frac{n}{2} - 1]$:

- $t_{2i+1}^s = 2i(g+1)$ and $t_{2i+1}^e = 2i(g+1) + g + 2$
- $t_{2i+2}^s = (2i+1)(g+1)$ and $t_{2i+2}^e = (2i+1)(g+1)+g+2$

with $r^s = 0$, $r^e = n(g+1)+1$ and some constant $g \in \mathbb{N}_{>0}$. The lower bound of section III-A applied to example 1 would be 0 whereas the actual lower bound is $\frac{n}{2}g^2$. Therefore, the ratio of the actual lower bound over the one defined by equation 6 can be arbitrarily large.

To achieve BC on the cost lower bound, the best admissible solution for the single resource should be taken into account. This best solution must be a maximal independent set of the conflict graph, as adding another task can only decrease the cost of a resource, but it is not necessarily either the maximum independent set (as the number of tasks is not relevant on instances with different durations) or even the largest (duration wise) maximal one as it would be the case with a linear cost.

More precisely, a solution corresponding to the lower bound must be a *shortest path* between fictive vertices v_0 and v_{n+1} corresponding to the opening and closing of the resource in the weighted *compatibility* Directed Acyclic Graph (DAG) $G = (V, E)$ of the n possible tasks of resource r_j:

- $V = \{v_i, \forall t_i \in \mathcal{T}_j\} \cup \{v_0, v_{n+1}\}$
- $E = \{(v_i, v_{i'}) \text{ s.t. } t_i^e \leq t_{i'}^s\}$
- $w : E \mapsto \mathbb{N}$ with $w((v_i, v_{i'})) = (t_{i'}^s - t_i^e)^2$

with $t_0^e = r^s$ and $t_{n+1}^s = r^e$. Note that any positive function could be used to weigh the idle times instead of their square.

In a DAG, the shortest path between two vertices can be computed in linear time $\Theta(|V| + |E|)$ [13], provided a topological ordering of the vertices (corresponding to the tasks sorted by increasing start time). Therefore, the lower bound of an *idlecost* optimization constraint can be computed in linear time with respect to G (i.e. possibly quadratic time w.r.t. the number of tasks n):

$$\mathrm{lb_j} \leftarrow \mathrm{dist}(v_0, v_{n+1}) \tag{7}$$

with $\mathrm{dist} : V^2 \mapsto \mathbb{N}$ the length of the shortest path from v_0 to v_{n+1} in G.

After the initialization steps where $\overline{\mathcal{T}_j}$ and G are built and the bounds computed, the *idlecost* constraint must propagate whenever:

- A task t_i is assigned to r_j:
  - the upper bound must be updated (cf. section III-A);
  - the arcs of the predecessors of v_i pointing to vertices $v_{i'}$ s.t. $i' > i$ (i.e. that "skip" v_i) must be deleted;
  - if v_i does not belong to the previous shortest path, a new one must be computed and the lower bound updated accordingly.
- A task t_i is removed from r_j:
  - the arcs of the predecessors of v_i pointing to v_i must be deleted;
  - if v_i belongs to the previous shortest path, a new one must be computed and the lower bound updated accordingly.

Moreover, thanks to the previous tightening algorithms, we are also able to filter the domains of the decision variables corresponding to tasks of $\overline{\mathcal{T}_j}$ whenever the bounds of the resource cost variable c_j are updated:

- If the upper bound $\overline{c_j}$ of c_j is modified:
  - if $\overline{c_j} < \mathrm{lb}_j$, a failure is triggered;
  - if $\overline{c_j} < \mathrm{ub}_j$, we try to determine if some tasks of $\overline{\mathcal{T}_j} \backslash \mathcal{T}_j$ (i.e. the set of possible tasks not yet assigned) must be added: for each task t_i of this set, if its removal entails a new lower bound lb'_j s.t. $\mathrm{lb}'_j > \overline{c_j}$, then the task must be assigned to r_j, i.e. $x_i = j$.
- Conversely, if the lower bound $\underline{c_j}$ of c_j is modified:
  - if $\underline{c_j} > \mathrm{ub}_j$, a failure is triggered;
  - if $\underline{c_j} > \mathrm{lb}_j$, we try do determine if some unassigned tasks must excluded from r_j: if the addition of a task entails a new upper bound ub'_j s.t. $\mathrm{ub}'_j < \underline{c_j}$, then the task must be excluded from r_j, i.e. $x_i \neq j$.

Note that all the modifications of the maintained data structures induced by these tests must systematically be undone.

To sum up, the *idlecost* optimization constraint achieves the BC on the cost of a single resource and the filtering of the resource variables of its possible tasks in:

- $O(n^2)$ when a task is assigned: $O(\log n)$ to update the upper bound, $O(n^2)$ to delete the bypassing edges of the DAG and $O(n^2)$ to find a shortest path;
- $O(n^2)$ when a task is removed: $O(n)$ to delete edges and $O(n^2)$ to compute the shortest path;
- $O(n^3)$ when the upper bound of c_j is modified: $O(n)$ shortest path to compute;
- $O(n^2 \log n)$ when the lower bound of c_j is modified: there might be $O(n)$ uncovered idle times at most for the $O(n)$ tasks.

The overall propagation algorithm has then a worst-case time complexity in $O(n^3)$. However, most of the time, the propagation events and the structure of the compatibility DAG of the resource at a given node will not trigger the necessary conditions to meet this worst case.

Nonetheless, the computation of shortest paths, involved in the most costly rules, can also be implemented incrementally. Each time a task t_i is assigned on the resource, let $t_{i'}$ and $t_{i''}$ be the preceding and succeeding assigned tasks (possibly t_0 and t_{n+1}) w.r.t. t_i, then only the shortest path between $v_{i'}$ and $v_{i''}$ needs to be recomputed (therefore the lower bound can be updated incrementally), and the shortest path problem is divided into two independent subproblems between $v_{i'}$ and v_i, and v_i and $v_{i''}$. The results presented in section IV were obtained with this incremental algorithm, not detailed here for the sake of brevity.

IV. Results

We report in this section the performances of the various techniques described in sections II and III to solve the FJS problem, especially the BC version of the *idlecost* constraint, implemented with the FaCiLe CP OCaml library [14].

We first show how the incremental BC version of the *idlecost* constraint (named BC-IC) described in section III-B largely outperforms the approximate one (named AP-IC) presented in section III-A2 on a specific class of instances. Then we compare the two versions on instances of the GAP extracted from real data, as well as the effect of symmetry breaking (cf. section II-B) and incremental maintenance of the shortest path mentioned at the end of section III-B. An ad hoc search strategy that balances the workload over resources and selects the best task w.r.t. the idle times cost was used throughout the tests.

All the experiments were carried out on an AMD Ryzen 1920X at 3.5GHz with 32GB of RAM, running GNU/Linux kernel 4.15 and OCaml 4.05.0 compiler. Note that all execution time and backtracks amount graphs are plotted with a base 10 logarithmic scale, with the AP-IC version in red and the BC-IC one in green, dashed lines corresponding to the discovery of the optimal solution and plain ones to its proof.

Fig. 2. Structured problem with m resources and $n = km$ tasks.

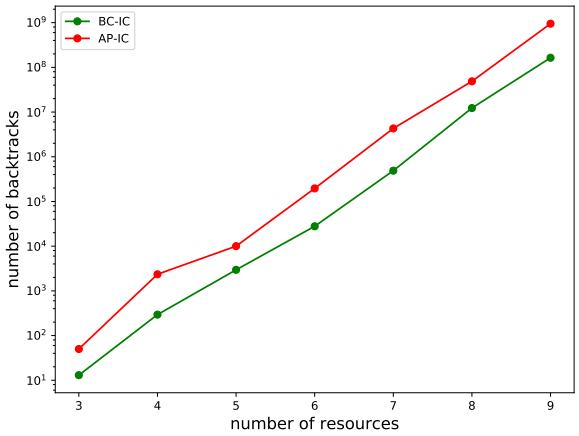

Fig. 3. Number of backtracks needed by the AP-IC and BC-IC versions to prove optimality for the structured problem w.r.t. the number of resources and with $k = 4$.

A. Structured Problem

To assess the benefit of the BC version of the *idlecost* constraint over the approximate one, we have tailored instances that generalize the class presented in example 1 for several resources. As shown in figure 2, the tasks of the instances are structured like successive "stairs" with m resources and $n = km$ tasks, which leads to an obvious optimal solution with one resource per level. As in example 1, the exact lower bound $k-1$ computed by BC-IC, can be arbitrarily better than the null value of its AP-IC counterpart.

Figures 3 and 4 respectively compare the resolution time and amount of backtracks to prove optimality for AP-IC and BC-IC with an increasing number of resources and $k = 4$ (i.e. 4 tasks per resource). The BC-IC version systematically outperforms the AP-IC one, up to 8 times better for the number of backtracks and 3 times better for the optimality proof, showing that the extra effort to compute an exact lower bound pays off on this class of instances.

B. Application to the Gate Allocation Problem

The GAP mainly focus on finding an allocation of a given set of aircraft with fixed occupancy periods to a number of gates. If there was no compatibility restriction, this decision problem could be modeled as the coloring of an interval graph, which is polynomial [7]. But gates can only accept a restricted set of aircraft types, so the set of compatible gates for an aircraft is limited and the decision problem of the allocation is rather a list-coloring problem, which is NP-Complete [15]. Moreover, an aircraft with scheduled arrival and departure

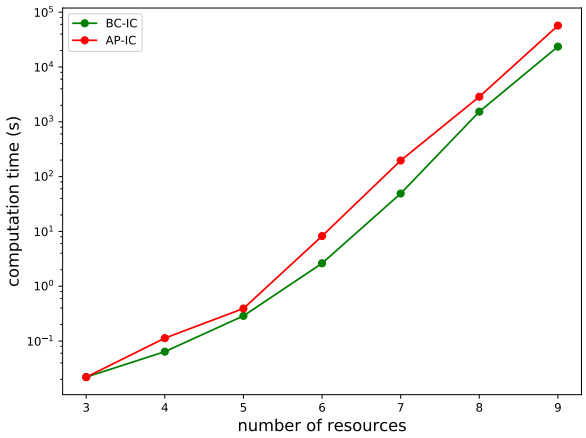

Fig. 4. Execution time (in seconds) corresponding to the number of backtracks of figure 3.

times can be considered as a task with fixed start and end times, and a gate as a specific resource. The GAP can therefore be considered as a FJS problem as defined in section I.

However, gates are endowed with many other secondary features (e.g. compatible airlines, domestic/international, terminal/remote etc.) which should match the characteristics of the flight and the preferences of airlines as much as possible. These preferences can be modeled as costs (or *soft* constraints) associated with each possible assignment, and standard GAP objectives usually aim at minimizing their sum, which is NP-Hard [16]. Other classic objectives include the walking distance of passengers or other connection means (e.g. buses), and there can be many side constraints like the simultaneous occupancy of adjacent gates for large aircraft.

As proposed by [2], we focus here on optimizing the robustness of the overall schedule, in order to absorb possible deviations from the original schedule due to traffic delays, severe weather conditions or equipment failures. Hence, the GAP version presented here is a FJS problem with an additively separable objective on the idle times (as mentioned in section I-D). Nevertheless, as our model is a CP one, many of the aforementioned side-constraints or objectives could be added easily.

In the following, we compare the performances of the AP-IC and BC-IC versions of the *idlecost* constraint on instances extracted from real traffic on a busy airport from 09:00 to 18:00. The instances are randomly built from small subsets of all the airport gates in order to obtain optimality proofs in reasonable time. An example of the Gantt diagram of an optimal solution is shown on figure 5 for an instance with 7 gates and 35 aircraft. We show as well the benefits of symmetry breaking on gates and incremental maintenance of the shortest path used by BC-IC.

1) AP-IC vs BC-IC: Figures 6 and 7 shows the number of backtracks and execution time to find and prove an optimal

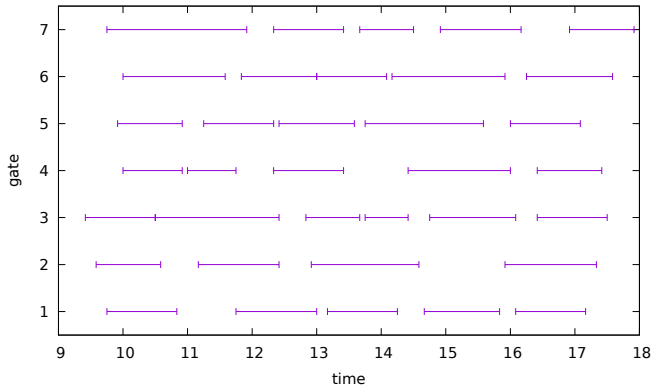

Fig. 5. Gantt diagram (in hours) of an optimal solution for an instance of the GAP with 7 gates and 35 aircraft extracted from real traffic.

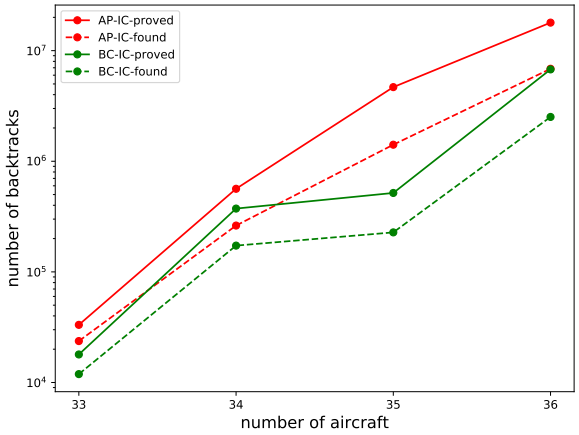

Fig. 6. Number of backtracks to find and prove an optimal solution with AP-IC and BC-IC for an instance of the GAP with 7 gates w.r.t. the number of aircraft (or tasks).

solution when using AP-IC and BC-IC for instances of the GAP with 7 gates (as in figure 5) w.r.t. the number of aircraft. BC-IC systematically outperforms AP-IC, especially in terms of backtracks which can differ by orders of magnitude.

2) Symmetry Breaking: As mentioned in section II-B, instances of the FJS problem (and the GAP) may present symmetry on equivalent resources or tasks (rarely for the GAP). Figures 8 and 9 present the number of backtracks and execution time to find and prove an optimal solution for the instances described in section IV-B1, with and without symmetry breaking on gates. As expected, the proof of optimality can be obtained orders of magnitude faster with symmetry breaking for the instances considered, and the optimal solution may be reached sooner for particularly hard instances with enough symmetry.

3) Incremental Maintenance of the Shortest Path: As mentioned in section III, the shortest path of the compatibility DAG used by IC-BC when the *idlecost* constraint propagates can be

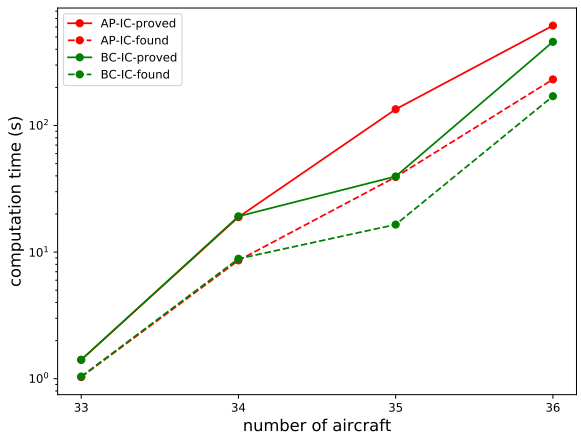

Fig. 7. Execution time (in seconds) corresponding to the number of backtracks of figure 6.

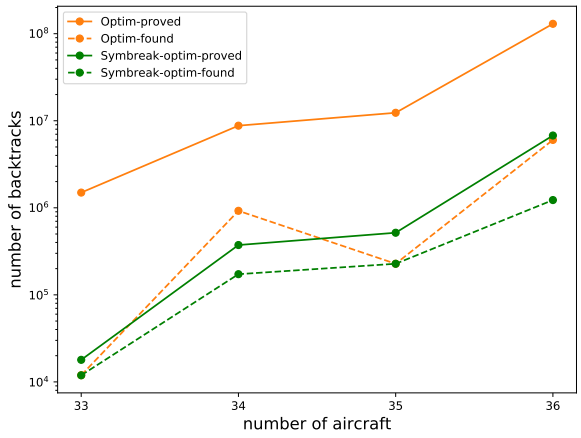

Fig. 8. Number of backtracks for BC-IC with (green) and without (orange) symmetry breaking on equivalent gates for the instances of figure 6.

computed incrementally, by restricting the shortest path to subproblems between fixed tasks. Figure 10 shows the execution time for the instances described in section IV-B1, with and without the incremental maintenance of the shortest path. The more sophisticated incremental algorithm consistently outperforms the simpler version based on the recomputation of the shortest path by approximately 10%.

Conclusion and Further Works

In this article, we present a new global constraint to improve the efficiency of CP solvers to find and prove robust solutions to the FJS problem that minimize the variance of idle times as proposed by [2]. This new optimization constraint, named *idlecost*, ensures the Bound Consistency of the idle times cost associated with each resource and the filtering of the resource variables associated with each possible task.

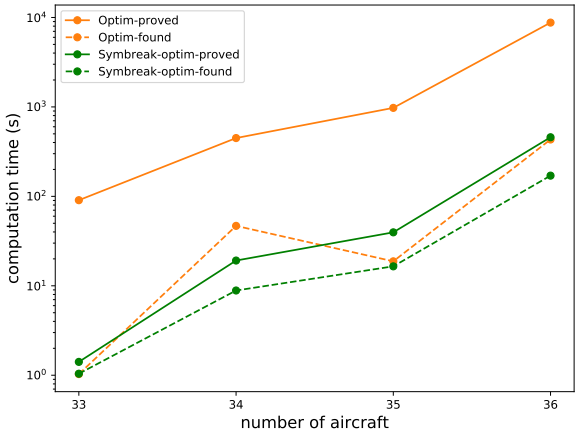

Fig. 9. Execution time (in seconds) corresponding to the number of backtracks of figure 8.

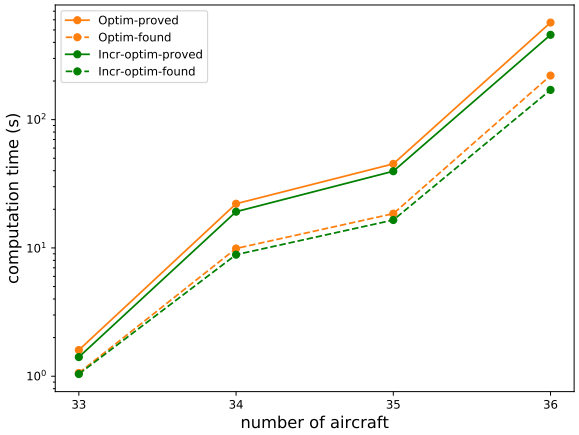

Fig. 10. Execution time (in seconds) for BC-IC with (green) and without (orange) incremental maintenance of the shortest path for the instances of figure 7.

In particular, we use a $O(|V|+|E|)$ shortest path algorithm on the compatibility DAG of the possible tasks to compute an exact lower bound for the cost of each resource and to include tasks when their removal would be incompatible with the current upper bound of the cost. We show that our approach consistently outperforms more naive approximations, sometimes by several orders of magnitude, for tailored structured problems and instances of the GAP extracted from real traffic.

We describe as well several useful techniques that improve resolution time like the computation of all maximal cliques of the interval graph to ensure mutual exclusion of overlapping tasks, the computation of global lower and upper bounds, the symmetry breaking on equivalent gates which helps to prove optimal solutions and sometimes to find them, and the incremental maintenance of the shortest path of the compatibility DAG at the core of our new optimization constraint.

Furthermore, our solver could benefit from additional refinements like the global processing of several *all-different* constraints [17] or the computation of a better, possibly dynamic, global lower bound to refine the one presented in this article. Our CP approach could also be hybridized with a MIP solver to improve its efficiency, or meta-heuristics to handle larger instances. Eventually, other standard objectives for the GAP, including the gate occupancy rate or allocation costs (corresponding to airlines and airport manager preferences), and side constraints (e.g. large aircraft may occupy two adjacent stands) could easily be added within our model.

REFERENCES

[1] D. T. Eliiyi and M. Azizoğlu, "Heuristics for operational fixed job scheduling problems with working and spread time constraints," *International Journal of Production Economics*, vol. 132, no. 1, pp. 107–121, 2011.

[2] A. Bolat, "Procedures for providing robust gate assignments for arriving aircrafts," *European Journal of Operational Research*, vol. 120, no. 1, pp. 63–80, 2000.

[3] S. H. Kim, "Airport control through intelligent gate assignment," Ph.D. dissertation, Georgia Institute of Technology, 2013.

[4] J. Guépet, R. Acuna-Agost, O. Briant, and J. Gayon, "Exact and heuristic approaches to the airport stand allocation problem," *European Journal of Operational Research*, vol. 246, no. 2, pp. 597 – 608, 2015.

[5] H. Simonis, "Models for global constraint applications," *Constraints*, vol. 12, no. 1, pp. 63–92, March 2007.

[6] A. Bolat, "Models and a genetic algorithm for static aircraft-gate assignment problem," *Journal of the Operational Research Society*, vol. 52, no. 10, pp. 1107–1120, Oct 2001.

[7] U. I. Gupta, D. T. Lee, and J. Y. Leung, "Efficient algorithms for interval graphs and circular-arc graphs," *Networks*, vol. 12, no. 4, pp. 459–467, 1982.

[8] E. M. Arkin and E. B. Silverberg, "Scheduling jobs with fixed start and end times," *Discrete Applied Mathematics*, vol. 18, no. 1, pp. 1 – 8, 1987.

[9] N. Beldiceanu and E. Contejean, "Introducing global constraints in CHIP," *Mathematical and Computer Modelling*, vol. 20, no. 12, pp. 97 – 123, 1994.

[10] N. Beldiceanu and M. Carlsson, "Sweep as a generic pruning technique applied to the non-overlapping rectangles constraint," in *International Conference on Principles and Practice of Constraint Programming*. Springer, 2001, pp. 377–391.

[11] M. Kutz, K. Elbassioni, I. Katriel, and M. Mahajan, "Simultaneous matchings: Hardness and approximation," *Journal of Computer and System Sciences*, vol. 74, no. 5, pp. 884 – 897, 2008.

[12] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*, 2nd ed. Springer-Verlag, 2000.

[13] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*, 3rd ed. The MIT Press, 2009.

[14] N. Barnier and P. Brisset, "FaCiLe: a Functional Constraint Library," in *CICLOPS – Colloquium on Implementation of Constraint and LOgic Programming Systems, CP'01 Workshop*, Paphos, Cyprus, December 2001.

[15] M. Biró, M. Hujter, and Z. Tuza, "Precoloring extension. I. Interval graphs," *Discrete Mathematics*, vol. 100, no. 1, pp. 267 – 279, 1992.

[16] L. G. Kroon, A. Sen, H. Deng, and A. Roy, "The optimal cost chromatic partition problem for trees and interval graphs," in *Graph-Theoretic Concepts in Computer Science*. Springer, 1997, pp. 279–292.

[17] F. Lardeux, E. Monfroy, and F. Saubion, "Interleaved alldifferent constraints: CSP vs. SAT approaches," in *Artificial Intelligence: Methodology, Systems, and Applications*, D. Dochev, M. Pistore, and P. Traverso, Eds. Springer, 2008, pp. 380–384.