Global Propagation of Transition Cost for Fixed Job Scheduling

Ruixin Wang, Nicolas Barnier

24th European Conference on Artificial Intelligence - ECAI 2020
Santiago de Compostela, Spain

Global Propagation of Transition Cost for Fixed Job Scheduling

Wang Ruixin[1] and **Barnier Nicolas**[2]

Abstract. We present a new Constraint Programming (CP) model to optimize the transition cost of Fixed Job Scheduling (FJS), which improves our previous approach based on per-resource constraints by orders of magnitude. Our new model relies on a much tighter relaxation which encompasses all resources to directly propagate on the global cost, thanks to the MinWeightAllDiff optimization constraint.

We also present several strategies which exploit the optimal matching computed by the MinWeightAllDiff constraint to efficiently guide the search. The resulting CP solver, using parallel cooperation between the strategies, consistently outperforms a state-of-the-art MIP solver on real instances of an FJS application, the Gate Allocation Problem, at Paris-Charles-de-Gaulle international airport.

INTRODUCTION

Fixed Job Scheduling (FJS) [7] is an important resource allocation problem with many applications, where jobs, or *tasks*, with fixed start and end times must 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.

For FJS applications like the *Gate Allocation Problem* (GAP) [4], which consists in assigning airport gates to aircraft, the execution of a plan often deviates from the schedule because of various unpredictable operational events. In this context, a more beneficial criterion is to optimize the *robustness* of the plan to absorb potential delays and avoid costly disruptions. Despite its practical importance, the research on the robustness of FJS or the GAP is very limited.

In [16], we presented a standard Constraint Programming (CP) approach to optimize the robustness of FJS as defined by [4], which proposes to minimize the variance of idle times, or equivalently the sum of their squares, to balance and spread them over time and resources. Our main contribution was the introduction of the new *idlecost* constraint to propagate the idle times costs, or any positive transition cost, on each resource independently.

However, the corresponding relaxation is not of good quality w.r.t. the global lower bound, as a task may be simultaneously scheduled on all its compatible resources. Moreover, our CP approach was not able to compete with the Integer Linear Programming (ILP) model of [5] in terms of execution times and size of solvable instances.

To obtain a competitive CP solver, we introduce in this article a new model based on the *MinWeightAllDiff* optimization constraint [6] to compute the lower bound of a *Path Cover* (PC) of the *compatibility graph* (cf. Section 1.6) of the problem. This relaxation is much tighter as the constraint directly propagates on the total cost, considering all resources and all tasks simultaneously. We also describe how the optimal PC computed by the constraint can efficiently guide the search strategy and show that our new approach outperforms our previous one by orders of magnitude on the GAP, as well as a state-of-the-art MIP solver on real instances at Paris-Charles-de-Gaulle (Paris-CDG) international airport.

In the following, we first present an integer model for FJS with transition cost in Section 1, then we describe in Section 2 our new CP model based on an incremental implementation of the MinWeightAllDiff constraint and two linked sets of variables. Section 3 then presents several search strategies and their parallel cooperation, which is able to outperform our previous approach and MIP solvers on real instances of the GAP as shown in Section 4. We conclude and discuss further works in the last section.

1 FIXED JOB SCHEDULING

The scheduling of tasks with fixed start and end times on non-identical resources is a versatile NP-complete problem [1] 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 the transition cost between tasks, particularly to obtain robust solutions w.r.t. delays.

The next sections present the integer model used in our study (whereas classic ILP approaches consider boolean variables as in [4]), with the introduction of *fictive tasks* to model the opening and closing of resources, and of the *compatibility graph* used to define our new CP model in Section 2.

1.1 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;

[1] CAUC-ENAC Joint Research Center of Applied Mathematics for ATM, CAUC, Tianjin, China, ruixin.wang@recherche.enac.fr
[2] ENAC, Université de Toulouse, France, nicolas.barnier@enac.fr

  - $\mathcal{R}_i \subseteq \mathcal{R}$ 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 r_j. However, all resources are considered available during the same period[3] in the following, i.e. $\forall j, r_j^s = r^s$ and $r_j^e = r^e$.
  - $\mathcal{T}_j = \{t_i \in \mathcal{T} \text{ s.t. } r_j \in \mathcal{R}_i\}$ the set[4] of compatible tasks that can be executed on resource r_j.

Without loss of generality, the tasks of $\mathcal{T}$ are supposed to be numbered by increasing start time, i.e. $\forall i < i', t_i^s \leq t_{i'}^s$.

1.2 Fictive Tasks and Renumbering

As already mentioned, our model is designed to minimize the variance of idle times. To obtain a uniform formulation of our model, even when the idle time considered occurs between the opening of a resource and its first task or between its last task and its closing, we introduce $2m$ additional fictive tasks with a singleton compatible resource set and null duration corresponding to the openings and closings of the m resources.

We accordingly renumber the tasks of $\mathcal{T}$ and the resource sets $\mathcal{R}_i$, while preserving the ordering by increasing start time:

- openings: $t_1, \ldots, t_m$ with $t_j^s = t_j^e = r^s$ and $\mathcal{R}_j = \{r_j\}$;
- actual tasks: $t_{m+1}, \ldots, t_{m+n}$ with $\mathcal{R}_{m+i}$ equal to $\mathcal{R}_i$ of Section 1.1 (before the renumbering);
- closings: $t_{m+n+1}, \ldots, t_{2m+n}$ with $t_{m+n+j}^s = t_{m+n+j}^e = r^e$ and $\mathcal{R}_{m+n+j} = \{r_j\}$.

The task set is therefore redefined by:

$$\mathcal{T} = \{\underbrace{t_1, \ldots, t_m}_{\text{openings}}, \underbrace{t_{m+1}, \ldots, t_{m+n}}_{\text{actual tasks}}, \underbrace{t_{m+n+1}, \ldots, t_{2m+n}}_{\text{closings}}\}$$

We also define the subsets of predecessors $\mathcal{T}^s$, successors $\mathcal{T}^e$ and actual tasks $\mathcal{T}^a$ (i.e. $\mathcal{T}$ in the previous section):

$$\begin{array}{rcl} \mathcal{T}^s & = & \{t_1, \ldots, t_{m+n}\} \\ \mathcal{T}^e & = & \{t_{m+1}, \ldots, t_{2m+n}\} \\ \mathcal{T}^a & = & \{t_{m+1}, \ldots, t_{m+n}\} \end{array}$$

We also extend each set $\mathcal{T}_j$ of compatible tasks to include its fictive opening t_j and closing t_{m+n+j}.

1.3 Decision Variables

A solution to an FJS problem consists in assigning a resource to each task while satisfying the non-overlapping constraints of equation (1) described in the next section. As the fictive tasks are already assigned to their associated resource, we define the set of decision variables over actual tasks only:

Definition 1 (Resource Variables) *A solution to an FJS problem is represented by a set of* resource variables*:*

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

where $x_i = j$ iff task t_i is assigned to resource r_j.

[3] Without loss of generality, as the unavailability of a resource can be modelled by an additional task with a singleton resource set.
[4] Redundantly defined from $\mathcal{R}_i$ to simplify notations afterwards.

1.4 Non-Overlapping Constraints

The only type of constraints of this essential version of the problem is 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:

$$x_i \neq x_{i'}, \quad \forall t_i \neq t_{i'} \in \mathcal{T}^a \text{ s.t. } [t_i^s, t_i^e[\cap [t_{i'}^s, t_{i'}^e[\neq \varnothing \qquad (1)$$

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. favor terminal gates over remote apron stands).

1.5 Transition 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, [4] 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 the mean of the idle times is constant for our problem (as the overall duration of tasks and availability of resources are constant, and all tasks must be scheduled), minimizing their variance amounts to minimizing the sum of their squares:

$$\text{cost} = \sum_{\forall t_i \in \mathcal{T}^s} (\text{next}(t_i)^s - t_i^e)^2 \qquad (2)$$

where function next : $\mathcal{T}^s \mapsto \mathcal{T}^e$ returns the successor of a task, i.e. the next task assigned on the same resource, closings having no successor and openings no predecessor.

More generally, our approach is generic and able to optimize the sum of the *transition* costs $c_{t_i,t_{i'}}$ between successive tasks $t_i \in \mathcal{T}^s$ and $t_{i'} \in \mathcal{T}^e$, with any positive cost matrix C:

$$\text{cost} = \sum_{\forall t_i \in \mathcal{T}^s} c_{t_i, \text{next}(t_i)} \qquad (3)$$

1.6 Global Compatibility Graph

To model the FJS transition cost in our CP solver, as described in Section 2, we define the notion of *compatibility* on a pair of tasks, then of the *Global Compatibility directed acyclic Graph* (GCG) of the whole problem.

The *compatibility* predicate indicates whether two ordered tasks can both be scheduled on the same resource:

Definition 2 (Compatibility) *The* compatibility *predicate $\gamma : \mathcal{T}^s \times \mathcal{T}^e \mapsto \mathbb{B}$ is defined over all pairs of ordered tasks t_i and $t_{i'}$ s.t. $i < i'$ by:*

$$\gamma(t_i, t_{i'}) = (\mathcal{R}_i \cap \mathcal{R}_{i'} \neq \varnothing) \ \wedge \ (t_i^e \leqslant t_{i'}^s)$$

If $\gamma(t_i, t_{i'})$ holds, then t_i and $t_{i'}$ are said to be compatible.

We can then define the GCG of the whole problem that represents each task (actual and fictive) as a node and each ordered pair of compatible tasks as an arc weighted by their transition cost:

Definition 3 (Global Compatibility Graph (GCG)) *The weighted* Global Compatibility Graph $G = (V, E)$ *of a FJS problem is defined by:*

- $V = \{v_i, \forall t_i \in \mathcal{T}\}$
- $E = \{(v_i, v_{i'}), \forall t_i \in \mathcal{T}^s, \forall t_{i'} \in \mathcal{T}^e,\ s.t.\ i < i' \wedge \gamma(t_i, t_{i'})\}$
- $w : E \mapsto \mathbb{R}_{\geq 0}$ *with* $w((v_i, v_{i'})) = c_{t_i,t_{i'}}$

We will take $c_{t_i,t_{i'}} = (t^s_{i'} - t^e_i)^2$ to optimize the robustness of GAP instances in Section 4. We will also use the following notation:

- $V_j = \{v_i, \forall t_i \in \mathcal{T}_j\}$, $\forall r_j \in \mathcal{R}$, i.e. the set of nodes corresponding to tasks compatible with resource r_j;
- $G_j = G[V_j]$ the subgraph of the GCG induced by the nodes of resource r_j;
- V^s, V^e and V^a the restriction of V to $\mathcal{T}^s$, $\mathcal{T}^e$ and $\mathcal{T}^a$;
- $N^+(v_i) \subset V^e$ the set of successors of node v_i.

2 CP MODEL

In [16], we presented a standard CP approach to optimize the robustness of FJS, with *AllDifferent* constraints on all maximal cliques of the interval graph of the tasks to enforce equation (1), and symmetry breaking among resources and tasks. Our main contribution was the introduction of the new *idlecost* constraint to propagate the idle times cost (or any positive transition cost), on each resource independently.

Particularly, an incremental shortest path algorithm was used on the restricted compatibility graph G_j (see Section 1.6) of the possible (and assigned) tasks of each resource r_j to compute the lower bound of its contribution to the global cost. However, the resulting CP solver was only able to solve small instances up to 40 tasks and could not compete with the ILP model of [5], because the lower bound of the global cost can be $O(m)$ times worse than the actual bound when the uniqueness of task assignment is relaxed.

To improve our CP approach, we introduce a new model based on a much tighter relaxation: we compute the *Minimum Weight Path Cover* (MWPC) [12] of the GCG (cf. Definition 3) in polynomial time, thanks to an optimization constraint that directly propagates on the total cost, considering all resources and all tasks simultaneously. More precisely, MWPC in a *Directed Acyclic Graph* (DAG) can be reduced to the *Linear Assignment Problem* (LAP), solvable by the *Hungarian method* [11] in $O(|V^s||E|)$, with $|V^s| = |\mathcal{T}^s| = n + m$. [14] describes how this algorithm can be used to achieve Generalized Arc Consistency (GAC) for the *MinWeightAllDiff* optimization constraint which can be posted on *successor* variables of the tasks to model the corresponding LAP.

Even if much tighter than the former relaxation, an MWPC is not in general a solution to FJS, as consistency on resources along each path is not taken into account. So resource variables are still necessary in our new model, as well as channelling constraints to link them with the successor variables.

In the following, we first present how the optimization of the transition cost of FJS can be relaxed to MWPC, which is modelled with successor variables and reduced to the LAP. Then we describe our incremental version of the MinWeightAllDiff constraint and the channelling constraints that link the successor and resource variables. The remaining of our new model is identical to [16] and therefore omitted.

2.1 Relaxation of FJS to Path Covering

In the GCG, a solution to the FJS problem corresponds to m vertex-disjoint simple paths $(v_j, \ldots, v_{m+n+j})$, $\forall j \in [1\,..\,m]$, joining the opening v_j to the closing v_{m+n+j} of each resource r_j while covering all vertices exactly once. Therefore, an optimal solution to the FJS w.r.t. the cost defined by equation (3) corresponds to a set of m such paths with minimal total length.

More generally, a set of vertex-disjoint simple paths that covers all vertices of an unweighted directed graph is called a *vertex-disjoint Path Cover* (PC), whose objective is to minimize the number of paths [12]. For the GCG of an FJS problem, there must be exactly m paths in any minimal PC, as there are exactly m sources v_j, $\forall j \in [1\,..\,m]$, and m sinks v_{m+n+j} (corresponding to the opening and closing of all resources). So we can use a variant of the PC problem called the *Minimum Weight Path Cover* (MWPC) whose objective is to produce a cover that minimizes the total weight of its edges.

We then obtain a relaxation of the FJS problem which is much tighter than the one of [16] w.r.t. the lower bound of the total cost, as all tasks are scheduled exactly once, instead of possibly $|\mathcal{R}_i| \leq m$ times for each task t_i. However, an MWPC is not in general a solution to FJS, as the consistency on resources along a path is not entailed by the model:

- A path starting at v_j, corresponding to the opening of resource r_j, may end at $v_{m+n+j'}$, with $j' \neq j$, the closing of another resource, e.g. if $\exists t_i \in \mathcal{T}^a$ s.t. $\{r_j, r'_j\} \subseteq \mathcal{R}_i$.
- More generally, if $\gamma(t_i, t_{i'}) \wedge \gamma(t_{i'}, t_{i''})$, nodes v_i, $v_{i'}$ and $v_{i''}$ can succeed each other along the same path, even though it does not imply that $\mathcal{R}_i \cap \mathcal{R}_{i''} \neq \varnothing$, which is necessary to assign t_i and $t_{i''}$ on the same resource

To obtain an exact FJS model, resource variables are kept in our new model and channelling constraints must be added, as explained in Section 2.4, to link them with the *successor* variables introduced in Section 2.2, which model the MWPC on the GCG. To implement the minimization constraint between the successor variables and the cost, we first reduce MWPC to the Linear Assignment Problem in the next section.

2.2 Successor Variables and Reduction to the Linear Assignment Problem

A PC in the GCG can be modelled by $m+n$ variables y_i that represent the *successor* of each node v_i in V^s, with a domain equal to the indices of its possible successors:

Definition 4 (Successor Variables) *A PC in the GCG of an FJS problem is represented by the following set of* successor variables*:*

$$\mathcal{Y} = \{y_i \in \{i', \forall v_{i'} \in N^+(v_i)\}, \forall v_i \in V^s\}$$

where $y_i = i'$ *iff* $\text{next}(t_i) = t_{i'}$.

We will denote $\text{dom}(y_i)$ the current domain of variable y_i.

Note that the union of the domains of $\mathcal{Y}$ must be equal to the indices of V^e, i.e. $\bigcup_{y_i \in \mathcal{Y}} \text{dom}(y_i) = [m+1\,..\,2m+n]$, as all actual tasks (with indices in $[m+1\,..\,m+n]$) have at least one predecessor (the openings of its compatible resources) and all

closings (with indices in $[m+n+1\,..\,2m+n]$) as well (its corresponding opening and compatible tasks).

To obtain a valid PC, the variables of $\mathcal{Y}$ must have distinct values, such that the assignment is a perfect matching between the $m+n$ variables and the $m+n$ values. So finding an MWPC on G reduces to the *Linear Assignment Problem* (LAP) with cost $\sum_{\forall v_i \in V^s} w((v_i, v_{y_i}))$, i.e. the cost of the FJS in equation (3).

We can model an MWPC with the *MinWeightAllDiff* global optimization constraint introduced by [6], for which [14] proposes a GAC algorithm based on the Hungarian algorithm.

2.3 The MinWeightAllDiff Constraint

To constrain the transition cost of FJS, we add to our model a MinWeightAllDiff constraint over the successor variables and the global cost, where the contribution of an assignment $y_i = i'$ is defined by $w'(i,i') = w((v_i, v_{i'})) = c_{t_i,t_{i'}}$ for all ordered pairs of compatible tasks t_i and $t_{i'}$ with $i < i'$:

$$\text{MinWeightAllDiff}(\mathcal{Y}, w', \text{cost}) \tag{4}$$

which is satisfied when variables of $\mathcal{Y}$ have distinct values and cost $= \sum_{\forall y_i \in \mathcal{Y}} w'(i, y_i)$. We will denote $[lb\,..\,ub]$ the current domain of the cost.

[14] describes how the Hungarian algorithm can be used to compute lb, then how the variables of $\mathcal{Y}$ can be pruned to withdraw values that only belong to assignments exceeding ub. To our knowledge, there is no implementation of the MinWeightAllDiff constraint in any of the main publicly available CP solvers. Therefore, we implemented an incremental version of the Hungarian algorithm for the FaCiLe OCaml constraint library [2], based on the C++ source code of [13], to compute lb in $O(|\mathcal{Y}|d)$, with $|\mathcal{Y}| = n+m$ and $d = \sum_{\forall y_i \in \mathcal{Y}} |\text{dom}(y_i)| = |E|$.

Our constraint propagates only when edges belonging to the previous minimal matching are removed: the remaining assignments are kept and augmented until a new perfect matching is obtained, then the optimal matching is computed and a failure is triggered if its weight is strictly greater than ub.

As the results obtained with the computation of lb alone outperformed by orders of magnitude our previous approach [16], we postponed to a later study the implementation of the pruning of $\mathcal{Y}$ described in [14] and the assessment of potential additional speed-ups for FJS. Note that the MinWeightAllDiff constraint is intended for constraint programs that minimize the assignment cost, so the constraint does not compute ub (nor prunes successor values w.r.t. lb). However, the same algorithm could be used to implement a maximization version and achieve BC for the cost and GAC for $\mathcal{Y}$.

However, as mentioned in Section 2.1, a solution to the LAP (therefore an MWPC), is generally not a solution to the FJS problem and variables of $\mathcal{Y}$ must be constrained with the resource variables $\mathcal{X}$ as described in the next section.

2.4 Channelling Constraints

To obtain valid FJS solutions, implication constraints that ensure *resource consistency* along each path of an MWPC must be added to link successor and resource variables. Moreover, a task succeeds to another one iff the *hole* (see Definition 5) between them is empty, so additional propagation rules can be associated with each resource to improve our model.

2.4.1 Resource Consistency

As mentioned in Section 2.1, each path of a PC must connect the opening t_j to the closing t_{m+n+j} of a resource r_j, and all tasks in-between must be assigned to r_j to obtain a valid FJS solution. Therefore, the following d channelling constraints must be added to our model to ensure that successive tasks are assigned to the same resource:

$$y_i = j \;\Rightarrow\; x_i = x_j, \quad \forall y_i \in \mathcal{Y}, \forall j \in \text{dom}(y_i) \tag{5}$$

where resource variables x_i of $\mathcal{X} = \{x_{m+1}, \ldots, x_{m+n}\}$ are extended to openings, i.e. $\forall j \leq m$, $x_j = j$, and closings, i.e. $x_{m+n+j} = j$, with bound variables. Note that we use *arc-consistent* equality constraints (instead of bound-consistent ones) on the RHS of the implication to improve the filtering of successor variables whenever the contraposition holds, i.e. $x_i \neq x_j \Rightarrow y_i \neq j$.

2.4.2 Tasks Exclusion

When a successor variable is assigned, i.e. $y_i = i'$, on a known resource, all tasks that could be scheduled between t_i and $t_{i'}$ should be removed from the resource. Conversely, when no task can be scheduled between t_i and $t_{i'}$, the successor variable must be assigned $y_i = i'$.

First, we define the notion of *hole* between assigned tasks to help specify the tasks exclusion constraints:

Definition 5 (Hole) *For any pair of tasks t_i and $t_{i'}$ assigned on resource r_j (i.e. $x_i = x_{i'} = j$) with no assigned task in-between (i.e. $\nexists t_{i''} \in \mathcal{T}_j$ s.t. $x_{i''} = j \wedge t^e_i \leq t^s_{i''} \wedge t^e_{i''} \leq t^s_{i'}$), hole $\mathcal{H}^{i,i'}_j$ is defined as the set of all unassigned tasks that fit between t_i and $t_{i'}$:*

$$\mathcal{H}^{i,i'}_j = \{t_{i''} \in \mathcal{T}_j \text{ s.t. } j \in \text{dom}(x_{i''}) \wedge t^e_i \leq t^s_{i''} \wedge t^e_{i''} \leq t^s_{i'}\}$$

We denote $\mathcal{H}_j$ the set of all holes of resource r_j.

For each resource r_j, we introduce a new *TaskExclusion* constraint on $\mathcal{X}_j$ and $\mathcal{Y}_j$, the restrictions of $\mathcal{X}$ and $\mathcal{Y}$ to $\mathcal{T}_j$, to remove j from the resource variables of the hole between connected tasks and, conversely, to chain the sides of holes that become empty:

$$\text{TaskExclusion}(r_j, \mathcal{X}_j, \mathcal{Y}_j), \quad \forall r_j \in \mathcal{R} \tag{6}$$

which is satisfied when $y_i = i' \Leftrightarrow \mathcal{H}^{i,i'}_j = \varnothing, \forall \mathcal{H}^{i,i'}_j \in \mathcal{H}_j$.

To propagate constraint (6), the following rules are triggered upon successor assignment or resource modification (with $t_{i''} \in \mathcal{H}^{i,i'}_j$ initially):

- $y_i = i' \Rightarrow x_{i''} \neq j, \forall t_{i''} \in \mathcal{H}^{i,i'}_j$
- $x_{i''} = j \Rightarrow (\mathcal{H}^{i,i''}_j = \varnothing \Rightarrow y_i = i'') \wedge (\mathcal{H}^{i'',i'}_j = \varnothing \Rightarrow y_{i''} = i')$
- $x_{i''} \neq j \Rightarrow (\mathcal{H}^{i,i'}_j = \varnothing \Rightarrow y_i = i')$

For each resource, the non-empty holes of $\mathcal{H}_j$ (initialized to $\{\mathcal{H}^{j,m+n+j}_j\}$, with $\mathcal{H}^{j,m+n+j}_j = \mathcal{T}_j \setminus \{t_j, t_{m+n+j}\}$) can be maintained in logarithmic time with a Binary Search Tree[5] (BST) upon a successor assignment (removal of a hole) and a task assignment (a hole must be divided in two) or exclusion

[5] Lexicographically ordered by pair $(t^e_i, t^s_{i'})$ for each hole $\mathcal{H}^{i,i'}_j$.

(the task must be removed from its hole). However, we only implemented a simple linear algorithm to query holes, letting the implementation of the BST for a later study, as a similar optimization of our previous model only improved resolution times by less than 10 %.

3 SEARCH STRATEGIES

This section presents several search strategies to optimize the robustness of the GAP instances solved in Section 4, i.e. to balance the idle times over time and resources. As an MWPC is a tight relaxation of FJS, all our strategies follow the optimal assignment computed by the MinWeightAllDiff constraint when trying to assign a successor variable.

The simplest strategy aims at selecting the least loaded resource, while the second one generalizes the notion of resources to non-empty holes (see Definition 5). Then both first assign a task on the resource or hole, before assigning its successor according to the optimal matching. The third strategy simply follows the optimal matching on one of the side of the best selected hole. As no single strategy was robust enough to efficiently solve our various instances, we eventually obtained the best results with their parallel cooperation.

3.1 Resource Balancing

In [16], we observed that the resource loads were balanced in optimal schedules. So the *Resource Balancing* (RB) strategy selects the least loaded resource r_j, i.e. with the greatest amount of idle time, among the ones where unassigned tasks can still be scheduled. Then it assigns to r_j either the task with the *Earliest Start Time* (EST) or the one that improves the cost the most, noted *Best Cost* (BC), before assigning the successor variable of the selected task according to the optimal matching computed by the MinWeightAllDiff constraint. Note that this second assignment contributes to the reduction of the search space as one of the channelling constraints (5) will be able to propagate. We will note:

- **RB-EST**: least loaded resource, earliest task;
- **RB-BC**: least loaded resource, best cost.

3.2 Critical Hole

To better take into account the structure of partial solutions where blocks of successive tasks can be discarded, we may consider each non-empty hole as an independent resource. So the *Critical Hole* (CH) strategy selects the one (among all resources) with the fewest (noted *Min*) or the most (noted *Max*) tasks, using the largest span to break ties, rather than the least loaded resource of strategy RB. Then it unfolds identically. This leads to four variants:

- **CH-Min-EST**: hole with fewest tasks, earliest task;
- **CH-Min-BC**: hole with fewest tasks, best cost;
- **CH-Max-EST**: hole with most tasks, earliest task;
- **CH-Max-BC**: hole with most tasks, best cost.

3.3 Task Chaining

Instead of selecting a new task to insert into the critical hole based on its start time or its impact on the cost as the previous strategy, the *Task Chaining* (TC) strategy simply follows the optimal matching provided by the MinWeightAllDiff constraint from the beginning of the hole, or backward from its end, by assigning the corresponding successor variable. We then choose to extend the side which reduces the most the span of the hole, called *Best Extension* (BE), or the one corresponding to the successor variable with the smallest domain, called *Next Size* (NS), which gives four variants:

- **TC-Min-BE**: hole with fewest tasks, best hole reduction;
- **TC-Min-NS**: hole with fewest tasks, smallest domain;
- **TC-Max-BE**: hole with most tasks, best hole reduction;
- **TC-Max-NS**: hole with most tasks, smallest domain.

3.4 Parallel Cooperation

Even if the TC-Min-BE strategy gives excellent results with many instances of the GAP (as shown on Figure 3), solving them in a few dozens of backtracks, others instances were more time-consuming and better solved by one of the other variants. Therefore, we can build a new strategy that benefits from all of the previous variants by exploring the search space in parallel while exchanging upper bounds between processes, provided there are enough available cores on the computer.

The development version of FaCiLe provides a parallel search goal that forks its process for each strategy and solves the same model while communicating bounds to all the children through their parent whenever a better solution is found. Note that this parallel cooperation of strategies may be strictly better than any single strategy on some instances as the acquisition of a new upper bound might shorten its resolution time. This strategy will be noted **COOP**.

4 RESULTS

We report in this section the performances of the various techniques described in Sections 2 and 3 to optimize the robustness of FJS, especially the parallel cooperation of strategies implemented with the FaCiLe CP OCaml library [2].

We first show that our new model based on the MinWeightAllDiff constraint outperforms by orders of magnitude our previously published model using the *idlecost* constraint [16]. Then we show that our CP approach also outperforms the ILP model of [5] solved by the state-of-the-art solver Gurobi [9], on instances recorded from real data[6] at Paris-CDG international airport.

All experiments were carried out on a standard workstation with a 2.0 GHz Intel® Xeon® 16-core processor with 48 GB of RAM running Debian GNU/Linux 9.6 with OCaml 4.05.0 and Gurobi 8.1. In all our tests, Gurobi and FaCiLe were allowed to exploit all 16 cores of our workstation, though the CP solver used only 11, one for the master program and ten for the parallel cooperation of the various strategies described in the previous section. Note that all execution times and backtrack amounts graphs are plotted with a base 10 logarithmic scale.

4.1 Gate Allocation Problem

The GAP mainly focuses on finding an allocation of a given set of aircraft with fixed occupancy periods to a number of

[6] All the data used in our experiments are available at `http://recherche.enac.fr/~wangrx/ecai_gap`.

gates. If there were no compatibility restrictions, this decision problem could be modelled as the coloring of an interval graph, which is polynomial [8]. 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 [3]. Moreover, an aircraft with scheduled arrival and departure 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 an FJS problem as defined in Section 1.

Furthermore, gates may be associated with other secondary features (e.g. compatible airlines, domestic vs. international, terminal gate vs. remote stand, etc.) which should match the characteristics of the flight and the preferences of airlines as much as possible. These preferences can be modelled as costs (or soft constraints) associated with each possible assignment, and standard GAP objectives usually aim at minimizing their sum, which is NP-Hard [10]. 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 [4], 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, equipment failures, etc. Hence, our version of the GAP is an FJS problem where the sum of the transition costs, defined as the square of idle times (cf. Section 1.5), should be minimized. Nevertheless, many of the aforementioned side-constraints or secondary objectives could be easily added to our CP model.

Table 1. Number of backtracks and execution time w.r.t. the number of aircraft to prove optimality for ICTAI and MWAD with 7 gates.

# a/c	# backtracks		execution time		speed-up
	ICTAI	MWAD	ICTAI	MWAD	
33	17,925	**68**	1.41 s	**0.06 s**	23
34	373,502	**22**	19.2 s	**0.03 s**	640
35	516,989	**2**	39.6 s	**0.02 s**	1,980
36	6,768,752	**24**	458.1 s	**0.04 s**	11,453

4.2 Idlecost vs. MinWeightAllDiff Models

Table 1 and Figure 1 compare the performance of our previously published CP model, named *ICTAI*, and our new model, named *MWAD*, with the TC-Min-BE strategy (cf. Section 3.3). They show the number of backtracks and execution time (in seconds) to prove optimality w.r.t. the number of aircraft on instances of the GAP with 7 gates published in [16]. Note that both y-axes are in logarithmic scale.

MWAD systematically outperforms ICTAI by orders of magnitude in terms of backtracks as well as execution time with speed-ups exceeding 10,000 for the largest instance.

4.3 Real-Size Instances at Paris-CDG

We present our results on 120 instances of actual traffic recorded on the busiest month (July) of 2017 at four terminals

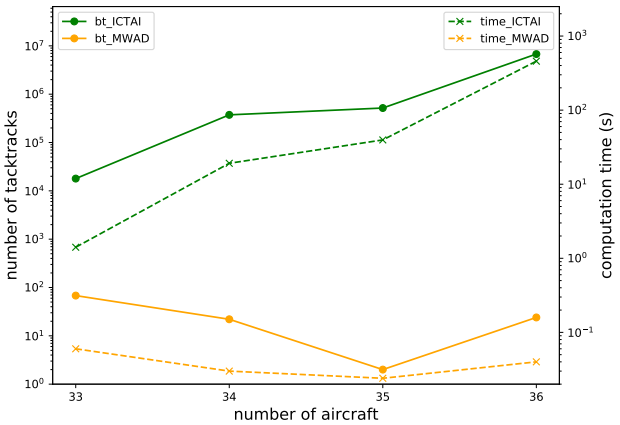

Figure 1. Number of backtracks (solid lines) and execution time in seconds (dashed lines) w.r.t. the number of aircraft to prove optimality for ICTAI and MWAD with 7 gates.

of Paris-CDG airport, one of the biggest European airports. Table 2 shows the mean of the number of flights and gates per day (as well as their ratio) for each selected terminal. Figure 2 presents the Gantt diagram of an optimal solution to an instance with 20 gates and 78 aircraft at terminal J.

We first discuss the performance of the search strategies mentioned in Section 3, then compare the execution times of our approach against an ILP model solved with Gurobi.

Table 2. Mean over the days of July 2017 of the number of aircraft and gates per day for four terminals at Paris-CDG.

terminal	# a/c	# gates	a/c per gate
B	15.4	10	1.5
J	71.7	20	3.6
K	53.4	19	2.8
Q	50.1	17	2.9

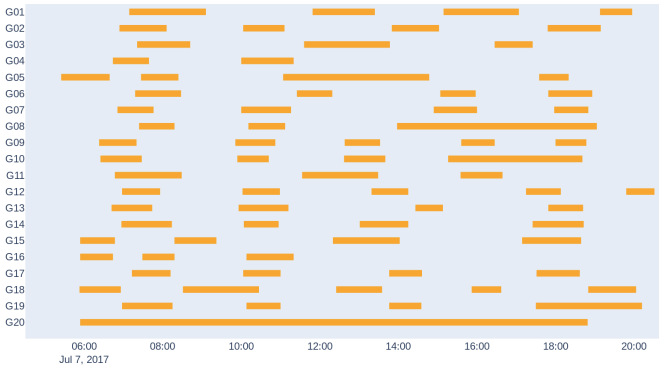

Figure 2. Gantt diagram of an optimal solution to an instance with 20 gates and 78 aircraft at terminal J of Paris-CDG airport.

4.3.1 Comparison of Search Strategies

To compare the robustness of the search strategies described in Section 3, Figure 3 gives the percentage of instances that MWAD is able to optimally solve within 60 s w.r.t. a single search strategy (grey bars), and for their cooperation (orange bar), on the four terminals mentioned in Table 2 during the busiest month of 2017. Each experiment was allowed to use the 16 cores of our workstation, though FaCiLe is only able to run a unique thread for single strategies.

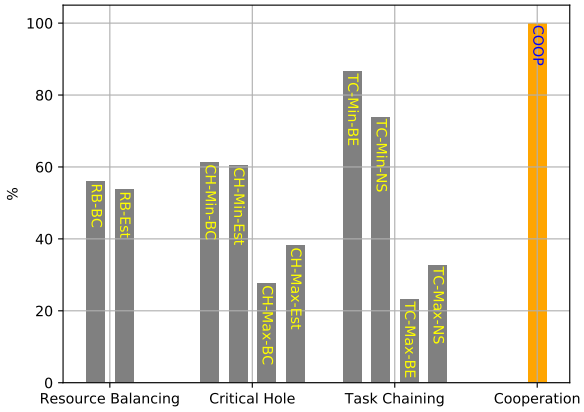

Figure 3. Percentage of optimally solved instances within 60 s by MWAD for all instances at Paris-CDG, w.r.t. the strategy.

As expected, the CH and TC strategies are more robust than the simpler RB one, with a success rate of more than 80 % for TC-Min-BE, which is the best variant for our instances. We can also observe that it is more efficient to concentrate the search efforts on holes with the fewest number of compatible flights (or tasks), i.e. the Min variants of CH and TC, following the "first-fail" principle, than to schedule more requested ones, as the gate restrictions are generally not too strict and still leave many assignment opportunities. However, TC-Min-BE is not systematically the best strategy, and none of the variants could easily be discarded as each one occasionally obtained the best results on some instances.

As anticipated, the cooperation of all variants exhibits a better behavior as it is able to consistently prove optimality for all instances, while no single strategy could, as depicted in Figure 3. Note that even though we use 60 s as the time limit, the average time spent by COOP to solve an instance is much smaller than 60 s, only slightly less than 10 s for the most difficult instances as shown in Figure 4.

4.3.2 Comparison with ILP

Finally, we compare our MWAD model using the COOP strategy and solved with the development version of FaCiLe to the ILP model named "P5" in [5] and solved by the Gurobi Commercial Optimizer 8.1.0 [9], with the same real traffic.

In Figure 4, we show the mean of execution times (in seconds with a logarithmic scale) to prove optimality over all

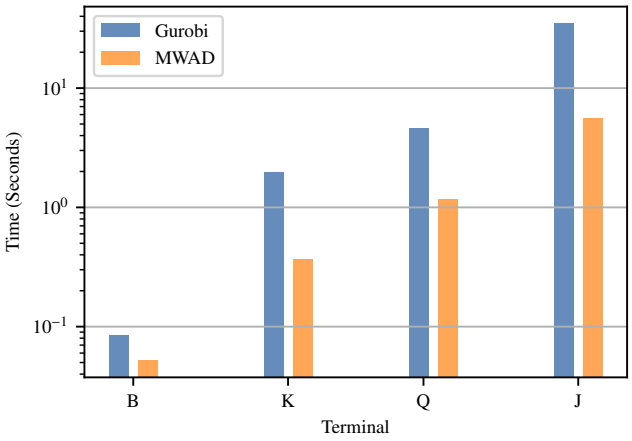

Figure 4. Mean of execution times (in seconds) to prove optimality with the ILP model (solved by Gurobi) and our new MWAD CP model (solved with FaCiLe) for all instances at Paris-CDG, w.r.t. the terminal.

the instances w.r.t. each terminal. Our new CP solver largely outperforms the state-of-the-art ILP solver on all terminals, up to 6.2 times faster for the busiest terminal (J).

The combination of a tight relaxation (thanks to the MinWeightAllDiff constraint), efficient heuristics that follows the corresponding optimal cover and their parallel cooperation (to be able to solve instances with distinct features) seems decisive to enable our CP solver to compete and outperform a state-of-the-art MIP solver.

CONCLUSION AND FURTHER WORKS

In this article, we have presented a new CP model and search strategies which considerably improves the approach described in [16] to optimize the transition cost of FJS. The main contribution of our novel approach consists in a much better relaxation to constrain the global cost which simultaneously takes all resources into account, instead of independent constraints that propagate the cost for a single resource and may underestimate the lower bound by far.

We show that FJS can be relaxed to a Path Covering problem in a DAG and that minimizing the sum of transition costs corresponds to Minimum Weight Path Covering, which can itself be reduced to the Linear Assignment Problem. Therefore, our model is able to compute a much better lower bound of the global cost thanks to a MinWeightAllDiff constraint [14] on successor variables. However, a path cover is generally not a solution to FJS and the resource variables of our previous model must be linked by channelling constraints to the successor variables in order to obtain a valid schedule.

Our new model, implemented with an incremental version of the MinWeightAllDiff constraint for the FaCiLe CP library and solved with a parallel cooperation of various strategies guided by the optimal covering computed by the constraint, outperforms our previous approach by orders of magnitude, as well as the ILP model of [5] on real instances of the GAP.

To further improve our solver, we intend to complete our implementation of the MinWeightAllDiff constraint, which

lacks the removal of inconsistent successors w.r.t. the upper bound of the cost, and to use BSTs to maintain holes for the *TaskExclusion* channelling constraint. Our search strategy could also benefit from the reduced costs computed by the MinWeightAllDiff constraint as suggested by [15], or from the *idlecost* constraint [16] of our former model to obtain the local lower bound for each hole.

ACKNOWLEDGEMENTS

We would like to thank the referees for their useful comments, which helped improve this paper.

REFERENCES

[1] Esther M. Arkin and Ellen B. Silverberg, 'Scheduling jobs with fixed start and end times', *Discrete Applied Mathematics*, **18**(1), 1 – 8, (1987).
[2] Nicolas Barnier and Pascal Brisset, 'FaCiLe: a Functional Constraint Library', in *CICLOPS – Colloquium on Implementation of Constraint and LOgic Programming Systems, CP'01 Workshop*, Paphos, Cyprus, (December 2001).
[3] M. Biró, M. Hujter, and Zs. Tuza, 'Precoloring extension. I. Interval graphs', *Discrete Mathematics*, **100**(1), 267 – 279, (1992).
[4] Ahmet Bolat, 'Procedures for providing robust gate assignments for arriving aircrafts', *European Journal of Operational Research*, **120**(1), 63–80, (2000).
[5] Ahmet Bolat, 'Models and a genetic algorithm for static aircraft-gate assignment problem', *Journal of the Operational Research Society*, **52**(10), 1107–1120, (Oct 2001).
[6] Yves Caseau and François Laburthe, 'Solving various weighted matching problems with constraints', in *Principles and Practice of Constraint Programming - CP'97*, ed., G. Smolka, volume 1330 of *Lecture Notes in Computer Science*, pp. 17–31, Berlin, Heidelberg, (1997). Springer.
[7] Deniz Türsel Eliiyi and Meral Azizoğlu, 'Heuristics for operational fixed job scheduling problems with working and spread time constraints', *International Journal of Production Economics*, **132**(1), 107–121, (2011).
[8] U. I. Gupta, D. T. Lee, and J. Y.-T. Leung, 'Efficient algorithms for interval graphs and circular-arc graphs', *Networks*, **12**(4), 459–467, (1982).
[9] LLC Gurobi Optimization. Gurobi optimizer reference manual, 2019.
[10] Leo G. Kroon, Arunabha Sen, Haiyong Deng, and Asim Roy, 'The optimal cost chromatic partition problem for trees and interval graphs', in *Graph-Theoretic Concepts in Computer Science*, pp. 279–292. Springer, (1997).
[11] H. W. Kuhn, 'The Hungarian method for the assignment problem', *Naval Research Logistics Quarterly*, **2**(1-2), 83–97, (1955).
[12] S. C. Ntafos and S. Louis Hakimi, 'On path cover problems in digraphs and applications to program testing', *IEEE Transactions on Software Engineering*, **SE-5**(5), 520–529, (September 1979).
[13] James Payor. Fast C++ implementation of the Hungarian algorithm. GitHub repository `https://github.com/jamespayor/weighted-bipartite-perfect-matching`, 2017.
[14] Meinolf Sellmann, 'An arc-consistency algorithm for the minimum weight all different constraint', in *Principles and Practice of Constraint Programming - CP 2002*, ed., Pascal Van Hentenryck, pp. 744–749, Berlin, Heidelberg, (2002). Springer Berlin Heidelberg.
[15] Willem-Jan van Hoeve, *Operations Research Techniques in Constraint Programming*, Ph.D. dissertation, Institute for Logic, Language and Computation (ILLC), University of Amsterdam, April 2005.
[16] Ruixin Wang and Nicolas Barnier, 'Propagation of idle times costs for fixed job scheduling', in *2018 IEEE 30th International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 718–725, (Nov 2018).