Optimisation du parcage des avions à l'aéroport Paris Charles de Gaulle.

Thibault Falque, Christophe Lecoutre, Bertrand Mazure, Karim Tabia

Optimisation de parcage des avions à l'aéroport Paris Charles de Gaulle

Thibault Falque[1,2], Christophe Lecoutre[2], Bertrand Mazure[2], Karim Tabia[2]

[1] Exakis Nelite
[2] CRIL, Univ Artois & CNRS

{falque,lecoutre,mazure,tabia}@cril.univ-artois.fr

Résumé

Plus que jamais, les acteurs du transport aérien (c'est-à-dire les compagnies aériennes et les aéroports), dans un climat de forte concurrence, doivent bénéficier d'une gestion optimisée des ressources aéroportuaires afin d'améliorer la qualité de leurs services et de contrôler les coûts induits. Dans cet article, nous étudions le problème d'affectation des parkings d'aéroport (APAP). Nous introduisons plusieurs modèles de programmation par contraintes (CP), que nous comparons, et présentons quelques résultats expérimentaux prometteurs à partir de données provenant d'ADP (Aéroport de Paris).

Mots-clés

programmation par contraintes, modélisation

Abstract

More than ever, air transport players (i.e., airline and airport companies), in a strongly competitive climate, need to benefit from a carefully optimized management of the airport resources in order to improve the quality of service and to control the induced costs. In this paper, we investigate the Airport Parking Assignment Problem (APAP). We introduce a Constraint Programming (CP) model, and present some promising preliminary experimental results from data coming from ADP (Aéroport de Paris).

Keywords

constraint programming, modelization

1 Introduction

Avant la crise sanitaire du COVID-19, l'Association Internationale du Transport Aérien (IATA) prévoyait que le nombre de passagers aériens allait presque doubler d'ici 2036, pour atteindre 7,8 milliards de personnes. Dans un tel contexte, l'optimisation de la gestion des ressources aéroportuaires reste essentielle pour maîtriser les coûts induits tout en conservant une bonne qualité de services.

Pour de nombreux problèmes de planification et d'ordonnancement du transport aérien, les techniques et outils développés à partir de la programmation mathématique et de la programmation par contraintes restent essentiels. En particulier, lorsque les compagnies aériennes ont accès aux ressources fournies par l'aéroport, la consommation de ces ressources (par exemple, les banques d'enregistrement, les parkings d'avions) doit être soigneusement planifiée tout en optimisant une fonction objectif déterminée par certaines règles commerciales ; voir, par exemple, [24, 22, 10].

Un problème classique du transport aérien est le problème d'affectation des portes d'aéroport (AGAP), qui consiste à affecter chaque vol à une porte d'embarquement disponible tout en maximisant à la fois le confort des passagers et l'efficacité opérationnelle de l'aéroport ; voir les études dans [1, 6] et les modèles dans [20, 21, 23]. Concenant le problème d'allocation de parking, [14] présente une modélisation MILP du problème en considérant des contraintes de capacité, d'ombrage et de satisfaction.

Étant donné que des améliorations significatives ont été apportées ces dernières années à la programmation par contraintes (CP), comme, par exemple, des algorithmes de filtrage (et de compression) efficaces pour les contraintes de table [4, 7], ou la génération de clauses paresseuses [5], s'attaquer aux tâches d'optimisation des aéroports avec la CP reste une question très intéressante. Dans cet article, nous nous intéressons au problème d'affectation des parkings d'aéroport (APAP) tel que défini à l'aéroport international CDG. Nous proposons des approches de Programmation par Contraintes (CP), et montrons son intérêt potentiel en présentant quelques résultats expérimentaux prometteurs.

Dans la section 2 nous introduisons le problème des parkings de l'aéoroport international de Charles de Gaulle. Dans la section suivante, nous présentons le problème sous forme de modèle COP. Dans la section 4, nous comparons les différentes variantes du modèle proposé sur des données de planification réelles (post-COVID) avec la solution actuellement utilisée par ADP. Nous faisons aussi une comparaison des modèles entre eux. Enfin nous terminons par une conclusion et des perspectives.

2 Problème de parcage des avions à Paris CDG

L'aéroport CDG est le neuvième plus grand aéroport du monde en termes de trafic de passagers. On compte 498175 mouvements (décollages ou atterrissages) par an, ce qui

correspond à environ 1400 mouvements par jour.
À l'aéroport, l'un des problèmes combinatoires à résoudre consiste à affecter chaque vol à un parking disponible (Pkg) ; ce problème est appelé APAP (Airport Parking Assignment Problem). Dans cette section, nous décrivons cette problématique. Tout d'abord, il existe deux types de parkings à l'aéroport : les parkings « **contacts** », désignés par $\mathcal{P}_{ct}$, qui sont des parkings « reliés » à un terminal par une porte et une passerelle, et les parkings « **au large** », désignés par $\mathcal{P}_{rt}$, qui sont des parkings accessibles en bus.

:definition : Ensemble des parkings
L'ensemble de tous les parkings, $\mathcal{P}_{ct} \cup \mathcal{P}_{rt}$, est noté par $\mathcal{P}$.

:definition : Rotation
Une **rotation** est l'ensemble des actions (atterrissage, stationnement, décollage) nécessaires à un avion, lors de son passage à CDG.

Bien qu'une rotation implique généralement deux vols (pour l'arrivée et le départ de l'avion), elle peut parfois n'impliquer qu'un seul vol (arrivée ou départ seul). Dans le cas d'un vol arrivée seul, l'avion occupe un parking avant d'être déplacé vers un hangar. Dans le cas d'un vol départ, l'avion sort d'un hangar avant d'occuper un parking. Pour chaque rotation, nous disposons des informations (données) suivantes :

- **start**(ψ) est l'heure de début de la rotation (c'est-à-dire l'heure à laquelle l'avion doit être géré parce qu'il vient d'atterrir ou qu'il sort d'un hangar) ;
- **end**(ψ) est l'heure de fin de la rotation (c'est-à-dire l'heure à laquelle l'avion ne doit plus être géré car il décolle ou va dans un hangar) ;
- **ntasks**(ψ) est le nombre de tâches (comme expliqué ci-dessous) formant la rotation ;
- **kind**(ψ) est le type avion de la rotation ;
- **airline**(ψ) est la compagnie de la rotation.

On peut considérer l'affectation d'un vol à un parking pendant une rotation comme une **tâche**. En effet, en fonction de la durée de la rotation, l'avion peut être déplacé plusieurs fois. Un vol peut alors être placé sur un seul parking pendant toute la durée de sa rotation (1 tâche), sur deux parkings successifs (2 tâches), voire trois parkings (3 tâches). Dans ce dernier cas le parking du milieu est nécessairement au large. Les vols à une seule arrivée ou un seul départ n'occupent nécessairement que 1 parking (tâche). La construction de l'ordonnancement du problème de stationnement consiste à trouver un stationnement pour chaque tâche de chaque rotation.
Il convient de noter que les conditions permettant de déterminer si un avion est mobile ou non (c'est-à-dire s'il utilise plus d'un stationnement) dépendent de certains temps de traitement tels que, par exemple, le temps de transit minimum, le temps de traitement minimum pour le départ, etc.
L'ensemble complet de rotations Ψ est divisé en trois sous-ensembles Ψ_1, Ψ_2, et Ψ_3, en fonction du fait que les rotations sont composées d'une, deux ou trois tâches.
Concernant la j^{me} tâche d'une rotation ψ, on suppose que l'on connaît le poids de la tâche $w_{\psi,j}$ (utilisé dans l'expression de la fonction objectif), l'heure de début de la tâche $s_{\psi,j}$, et l'heure de fin de la tâche $e_{\psi,j}$. Notez que $s_{\psi,1} = \texttt{start}(\psi)$ et $e_{\psi,k} = \texttt{end}(\psi)$ si k est l'indice de la dernière tâche de la rotation ψ.
Bien entendu, l'affectation d'un parking à une tâche est soumise aux préférences de placement des compagnies aériennes. Pour chaque tâche de chaque rotation, une récompense est donnée : la récompense de l'affectation du parking p à la tâche j de la rotation ψ est désignée par $r^p_{\psi,j}$. En supposant que nous disposons d'une série [1] de variables binaires $x^p_{\psi,j}$ associées à chaque tâche de rotation (indiquant quel parking sera utilisé), et en considérant le poids $w_{\psi,j}$ de chaque tâche de rotation, on peut alors définir la fonction objectif global comme suit :

$$\text{maximize} \sum_{\substack{\psi \in \Psi \\ j \in 1..\texttt{ntasks}(\psi)}} w_{\psi,j} \times x^p_{\psi,j} \times r^p_{\psi,j} \qquad (1)$$

En outre, il existe certaines contraintes physiques, imposées par l'infrastructure.

:definition : Capacité
Les contraintes de `capacité` empêchent certains types d'avions d'être placés sur certains parkings.

:definition : Ombrage
Les contraintes `ombrages` bloquent le positionnement d'un avion sur certains parkings voisins (quel que soit le type d'avion).

:definition : Ensemble des parkings ombrés
Pour tout parking $p \in \mathcal{P}$, on note $\mathcal{S}_p$ l'ensemble des parkings ombrés par p.

:example :
Par exemple, pour la contrainte `ombrage`, la figure 1 montre que, si un avion est placé sur le parking *A14*, alors le parking *A16* est « ombré ».

:definition : Réduction
Les contraintes `réductions` sont similaires aux contraintes `ombrages` sauf qu'elles prennent en compte les types avions. La constrainte `reduction` est définie par des

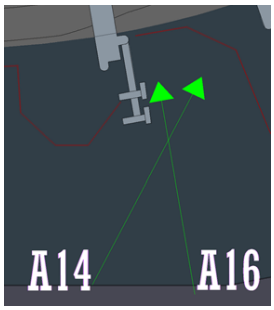


FIGURE 1 – Exemple pour la contrainte ombrage

1. Notez que nous n'utilisons pas de variables 0/1 dans le modèle proposé à la section 3.

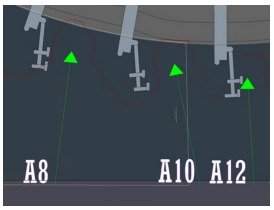


FIGURE 2 – Exemple pour la contrainte ombrages

4-tuples (k, p, k', ϕ) avec ϕ étant un ensemble de stationnements. Pour chaque stationnement p' dans ϕ, un avion de type k' est autorisé sur p' si un avion de type k est placé sur p.

:definition : Ensemble des réductions

Nous notons $\mathcal{D}$ l'ensemble de toutes ces réductions (tous les 4-tuples).

:example :

À titre d'illustration, considérons la figure 2, l'`ombrage` ne sera effectif que si un certain type d'avion a été placé sur *A10* et la réduction permettra toujours à un sous-ensemble de types avions d'être placés sur *A8* et *A12*.

:definition : NoOverlap

Une contrainte `NoOverlap` reflète l'impossibilité physique d'affecter deux tâches (deux vols) au même parking.

:definition : Chevauchement de tâches

Une paire (ψ, i) correspondant à la tâche i d'une rotation ψ se chevauche avec une paire (ψ', j) correspondant à la tâche j d'une rotation ψ' si $s_{\psi,i} < e_{\psi',j}$ et $s_{\psi',j} < e_{\psi,i}$.

:definition : Ensemble de tâches en chevauchement

Les tâches se chevauchant avec (ψ, i) sont désignées par l'ensemble $\mathcal{O}_{\psi,i}$, qui contient toutes les paires (ψ', j) se chevauchant avec (ψ, i).

3 Modèle COP

Nous allons maintenant décrire le problème plus formellement au moyen d'un réseau de contraintes. Un *Réseau de contraintes* (CN) est constitué d'un ensemble fini de variables soumises à un ensemble fini de contraintes. Chaque variable x peut prendre une valeur dans un ensemble fini appelé le *domaine* de x. Chaque contrainte c est spécifiée par une relation définie sur (le produit cartésien des domaines de x). Une *solution* d'un CN est l'affectation d'une valeur à chaque variable de sorte que toutes les contraintes soient satisfaites. Un *Réseau de contraintes sous optimisation* (CNO) est un réseau de contraintes qui comprend en plus une fonction objectif `obj` qui fait correspondre toute solution à une valeur dans $\mathbb{R}$.

Pour la modélisation des CNO, également appelés problèmes d'optimisation de contraintes (COP), plusieurs langages de modélisation et bibliothèques existent tels que, par exemple, OPL [30], MiniZinc [25, 29], Essence [13] et PyCSP^3 [18]. Notre choix s'est porté sur la bibliothèque Python récemment développée PyCSP^3 qui permet de générer des instances spécifiques (après avoir fourni des données ad hoc) au format XCSP3 [2, 3], qui est reconnu par certains solveurs de CP bien connus comme ACE (AbsCon Essence) [15], OscaR [26], Choco [27], et PicatSAT [35].

Rot.	`Compagnie`	#`Tches`	`Type`	`Dbut`	`Fin`
ψ_1	$\mathtt{a}_1$	1	k_1	$8h00$	$10h00$
ψ_2	$\mathtt{a}_2$	2	k_2	$8h00$	$12h00$
ψ_3	$\mathtt{a}_2$	2	k_2	$12h00$	$16h00$
ψ_4	$\mathtt{a}_3$	3	k_3	$9h00$	$15h00$
ψ_5	$\mathtt{a}_3$	3	k_4	$10h00$	$16h00$

TABLE 1 – Données sur les rotations

Nous allons définir dans cette section plusieurs variantes du problème d'affectation des parkings. Nous commencerons par les élements communs des différents variantes puis définirons les spécificités de chaque variante.

3.1 Définitions et contraintes communes

Tout d'abord, nous présentons les variables de notre modèle. En plus de la variable utilisée pour compter le nombre de rotations qui ne sont pas cassées, nous avons besoin de deux matrices de variables pour représenter le stationnement assigné, et les récompenses associées :

- p est une matrice de $|\Psi| \times 3$ variables ayant pour domaine $\{0, ..., |\mathcal{P}| - 1\}$; $p[\psi][j]$ représente l'index du parking (code) assigné à la *jth* tâche de ψ ;
- r est une matrice de $|\Psi| \times 3$ variables ayant pour domaine $\{0, ..., 100\}$; $r[\psi][j]$ représente la satisfaction de la compagnie pour la *jth* tâche de la rotation ψ.

Notez que, par souci de simplicité, nous introduisons toujours trois variables dans p et r pour chaque rotation, même si une seule tâche (ou deux tâches) est impliquée. Pour traiter ces cas, nous introduisons des contraintes d'égalité obligeant les variables à prendre la même valeur. En pratique, on pourrait éviter d'introduire de telles variables redondantes.

Deuxièmement, nous devons introduire les contraintes dans notre modèle. En raison de la nature du problème (et des données), il est naturel d'utiliser ce qu'on appelle des contraintes de table, qui énumèrent explicitement soit les tuples autorisés (tableau positif) ou les tuples interdits (tableau négatif) pour une séquence de variables (représentant la portée de la contrainte). Des algorithmes efficaces pour ces tableaux de contraintes ont été développés au cours de la dernière décennie. [8, 16, 19, 31].

Pour la suite, nous introduisons un exemple afin d'illustrer les contraintes ajoutées. Considérons un exemple avec un ensemble de 5 rotations $\{\psi_1, ..., \psi_5\}$, et un ensemble de 4 stationnements $\{\mathtt{p}_1, ..., \mathtt{p}_4\}$. Les informations concernant les rotations sont données dans le tableau 1.

Pkg	Capacité
$\mathtt{p}_1$	$\{k_1, k_2\}$
$\mathtt{p}_2$	$\{k_3, k_2\}$
$\mathtt{p}_3$	$\{k_1, k_2, k_3\}$
$\mathtt{p}_4$	$\{k_1, k_2, k_3, k_4\}$

TABLE 2 – Capacité

3.1.1 Contraintes sur les tâches

Tout d'abord, il est nécessaire de présenter quelques contraintes de stationnement concernant les tâches des rotations. Lorsque la division d'une rotation en 2 ou 3 tâches n'est pas possible (parce que le temps de rotation n'est pas assez grand), nous devons nous assurer que les 3 variables de la rotation sont affectées au même parking (parce que, pour simplifier, comme déjà mentionné, nous avons toujours 3 variables de p qui sont introduites pour chaque rotation). Nous ajoutons donc les contraintes suivantes :

$$p[\psi][1] = p[\psi][2] = p[\psi][3], \forall \psi \in \Psi_1 \tag{2}$$

$$p[\psi][2] = p[\psi][3], \forall \psi \in \Psi_2 \tag{3}$$

$$p[\psi][2] \in \mathcal{P}_{rt}, \forall \psi \in \Psi_3 \tag{4}$$

L'équation 2 oblige les 3 variables de p pour la rotation ψ à être assignées au même parking (car la rotation n'implique qu'une seule tâche). L'équation 3 force les deuxième et troisième variables de p de la rotation ψ à être assignées au même parking (parce que la rotation n'implique que deux tâches). Enfin, l'équation 4 oblige le parking du milieu à être placé sur un parking au large.

Pour notre exemple, supposons que $\Psi_1 = \{\psi_1\}$, $\Psi_2 = \{\psi_2, \psi_3\}$, et $\Psi_3 = \{\psi_4, \psi_5\}$, nous devons alors spécifier :

$$p[\psi_1][1] = p[\psi_1][2] = p[\psi_1][3]$$
$$p[\psi_2][2] = p[\psi_2][3] \wedge p[\psi_3][2] = p[\psi_3][3]$$
$$p[\psi_4][2] \in \mathcal{P}_{rt} \wedge p[\psi_5][2] \in \mathcal{P}_{rt}$$

3.1.2 Contraintes capacité

Pour appliquer de telles contraintes, nous avons simplement besoin de contraintes unaires. En supposant que l'ensemble des stationnements respectant la capacité d'une rotation ψ est noté $\mathcal{P}_\psi$, nous ajoutons la contrainte suivante :

$$p[\psi][j] \in \mathcal{P}_\psi, \forall \psi \in \Psi, \forall j \in 1..3 \tag{5}$$

Nous forçons ici toutes les tâches d'une rotation ψ à prendre leurs valeurs dans l'ensemble $\mathcal{P}_\psi$.

La capacité de chaque parking (dans notre exemple) est définie dans la Table 2. Nous devons donc ajouter les contraintes unaires suivantes par rapport à la rotation ψ_1 :

$$p[\psi_1][j] \in \{\mathtt{p}_1, \mathtt{p}_3, \mathtt{p}_4\}, \forall j \in 1..3$$

3.1.3 Calcul de la satisfaction compagnie

Selon les préférences des compagnies, des récompenses différentes sont associées à des stationnements différents. Nous pouvons utiliser des contraintes de table binaires pour « calculer » les récompenses lors du filtrage de ces contraintes.

$$\langle p[\psi][j], r[\psi][j] \rangle \in \{(p, r^p_{\psi,j}) \mid p \in \mathcal{P}_\psi\}, \\ \forall \psi \in \Psi, \forall j \in 1..\mathtt{ntasks}(\psi) \tag{6}$$

Nous ajoutons également une contrainte table en conflit pour interdire les parkings incompatibles avec cette tâche.

Pour notre exemple, supposons que les récompenses sont données par la matrice suivante :

Compagnie / Pkg	$\mathtt{p}_1$	$\mathtt{p}_2$	$\mathtt{p}_3$	$\mathtt{p}_4$	$\mathtt{p}_5$
a_1	75	75	100	50	50
a_2	60	0	100	50	50
a_3	0	100	80	50	50

À partir de ces données, nous devons par exemple ajouter la contrainte suivante concernant la première rotation ψ_1 (notez que $\mathtt{ntasks}(\psi_1) = 1$) :

$$\langle p[\psi_1][1], r[\psi_1][1] \rangle \in \\ \{(\mathtt{p}_1, 75), (\mathtt{p}_2, 75), (\mathtt{p}_3, 100), (\mathtt{p}_4, 50), (\mathtt{p}_5, 50)\}$$

3.1.4 Contraintes Ombrage

Rappelons que lorsqu'un parking p_1 est « ombré » par un parking p_2 alors les deux valeurs associées ne peuvent pas être assignées ensemble à une paire de de tâches qui se chevauchent. Cela conduit à des contraintes de table négatives binaires. Bien que cela ne soit pas explicitement indiqué ci-dessous, l'attribution de la même valeur deux fois pour une paire de tâches qui se chevauchent est également interdite (contraint `NoOverlap`). Les contraintes d'ombrage s'écrivent comme suit :

$$\langle p[\psi][i], p[\psi'][j] \rangle \notin \{(p, p') \mid p \in \mathcal{P}_\psi \wedge p' \in \mathcal{P}_{\psi'} \cap \mathcal{S}_p\}, \\ \forall \psi \in \Psi, \forall i \in 1..3, \forall (\psi', j) \in \mathcal{O}_{\psi,i} \tag{7}$$

Dans le contexte de notre exemple, une contrainte d'`ombrage` existe entre les parkings p_1 et p_3. Nous avons $\mathcal{O}_{\psi_1,1} = \{(\psi_2, 1), (\psi_4, 1)\}$, $\mathcal{P}_{\psi_1} = \mathcal{P}_{\psi_2} = \{p_1, p_3, p_4, p_5\}$, $\mathcal{P}_{\psi_4} = \{p_2, p_3, p_4\}$, et $\mathcal{S}_{p_1} = \{p_3\}$.

Par conséquent, nous pouvons ajouter les deux contraintes suivantes concernant ψ_1 : $\langle p[\psi_1][1], p[\psi_2][1] \rangle \notin \{(p_1, p_3)\}$ et $\langle p[\psi_1][1], p[\psi_4][1] \rangle \notin \{(p_1, p_3)\}$.

3.1.5 Contraintes Réduction

Ce dernier sous-ensemble de contraintes prend en compte les réductions sous la forme de contraintes binaires négatives de table.

$$\langle p[\psi][i], p[\psi'][j] \rangle \notin \{(p, p') \mid p' \in \phi\}, \\ \forall \psi \in \Psi, \forall i \in 1..\mathtt{ntasks}(\psi), \\ \forall \langle k, p, k', \phi \rangle \in \mathcal{D} \mid k = \mathtt{kind}(\psi), \\ \forall (\psi', j) \in \mathcal{O}_{\psi,i} \mid k' \neq \mathtt{kind}(\psi') \tag{8}$$

Pour notre exemple, une contrainte réduction existe entre les parkings p_1 et p_2. Supposons que $\mathcal{D} = \{(k_1, p_1, k_3, \{p_2\}), (k_2, p_1, k_4, \{p_2\})\}$. Nous avons également $\mathcal{O}_{\psi_1,1} = \{(\psi_2, 1), (\psi_4, 1)\}$ et $\mathcal{P}_{\psi_1} = \{p_1, p_3, p_4\}$.
La tâche 1 de la rotation ψ_2 chevauche la tâche 1 de la rotation ψ_1. Rappelons que $\texttt{kind}(\psi_2) = k_2$. Ainsi, pour la réduction de ψ_1, nous ajoutons la contrainte suivante qui interdit la paire de valeurs (p_1, p_2) pour la paire de variables $\langle p[\psi_1][1], p[\psi_2][1]\rangle$. En d'autres termes, il est interdit d'utiliser p_2 avec la rotation ψ_2 car ψ_2 n'a pas le type autorisé par p_2 après avoir placé ψ_1 sur p_1 (on dit que sa capacité est réduite).
Nous ajoutons alors la contrainte suivante $\langle p[\psi_1][1], p[\psi_2][1]\rangle \notin \{(p_1, p_2)\}$.
Bien que ψ_1 chevauche également avec ψ_4, il n'y a aucune restriction à considérer avec ψ_1 car ψ_4 a pour capacité le type k_3 (voir Table 1) qui est autorisé par rapport à la réduction.

3.1.6 Fonction objectif

La fonction objectif s'exprime comme suit en considérant les variables de récompenses :

$$\text{maximize} \sum_{\substack{\psi \in \Psi \\ j \in 1..\texttt{ntasks}(\psi)}} w_{\psi,j} \times r[\psi][j] \qquad (9)$$

3.2 Variante classique

Avec les variables et contraintes présentées dans la précédente section, nous pouvons définir une première variante du problème composée des contraintes 2, 3, 4, 5, 6,7 8 et de la fonction objectif 9. Nous noterons cette variante no-variant.

3.3 Variante AllDifferent

Dans cette variante nous proposons de modifier la contrainte ombrage en postant en plus des contraintes tables déjà proposées des contraintes alldifferent. Notons que pour cette contrainte, il est tentant de vouloir utiliser une contrainte AllDifferent comme déjà proposé dans l'état de l'art [28, 11]. Dans ces approches, une contrainte AllDifferent est ajoutée entre toutes les paires de tâches qui se chevauchent cependant cela ne peut pas fonctionner ici car elle serait moins contraignante que ce qu'exprime la contrainte ombrage. En effet une contrainte AllDifferent obligerait $p[\psi][i]$ et $p[\psi'][j]$ à être différent. Or en mettant p et p' comme valeur à ces deux variables, la contrainte serait respectée mais violerait la contrainte ombrage.
Néanmoins il est possible d'utiliser des contraintes AllDifferent avec un graphe d'intervalles. Chaque sommet du graphe représente une tâche (l'intervalle de temps de la tâche) et est connecté par une arête à un autre sommet, si et seulement si, il existe un chevauchement entre les deux intervalles. Le graphe d'intervalles serait donc, pour notre problème, noté $\mathcal{G} = (\mathtt{V}, \mathtt{E})$ avec $\mathtt{V} = \{\mathtt{I}_{\psi,i}, \forall \psi \in \Psi, \forall i \in 1..3\}$ et $\mathtt{E} = \{(\mathtt{I}_{\psi,i}, \mathtt{I}_{\psi',j}), \forall \psi \in \Psi, \forall i \in 1..3, \forall (\psi', j) \in \mathcal{O}_{\psi,i}\}$. $\mathtt{I}_{\psi,i}$ représente l'intervalle de temps de la tâche i de la rotation ψ. C'est-à-dire $e_{\psi,i} - s_{\psi,i}$. On note alors $\mathcal{C}_\mathcal{G}$ l'ensemble des cliques maximums. Pour un tel ensemble, nous pouvons poster les contraintes AllDifferent suivantes.

$$\text{allDifferent}(\{p[\psi][i], \forall (\psi, i) \in \mathtt{c}\}), \forall \mathtt{c} \in \mathcal{C}_\mathcal{G} \qquad (10)$$

Pour cette variante nous avons donc les mêmes contraintes que la variante précédente et ajoutons la contrainte 10. Nous notons alldifferent cette variante du problème.

3.4 Variante not-break

Cette variante du problème tient compte du fait que le déplacement d'un avion d'un parking à un autre a un certain coût. Pour cette raison, le nombre de stationnements utilisés pour une rotation (avec 2 ou 3 tâches) doit être minimisé. Pour faire cela, nous ajoutons une nouvelle variable nb, ayant pour domaine $\{0, ..., |\Psi|\}$, comptant le nombre de fois où rotation n'est pas rompue, c'est-à-dire le nombre de fois où un seul stationnement est utilisé pendant toute la durée de la rotation.
Nous devons ajouter la contrainte suivante afin de calculer le nombre de rotations « non-cassées » :

$$\mathtt{nb} = \sum_{\psi \in \Psi_2} s_{\psi,1=\psi,2} + \sum_{\psi \in \Psi_3} s_{\psi,1=\psi,2=\psi,3} \qquad (11)$$

où $s_{\psi,1=\psi,2}$ est une variable booléenne valant vraie si le même parking a été affecté pour la première et troisième tâche de la rotation ψ et où $s_{\psi,1=\psi,2=\psi,3}$ est vraie si toutes les tâches de la rotation ψ ont été affectées au même parking.
La fonction objectif est modifée comme suit :

$$\text{maximize} \sum_{\substack{\psi \in \Psi \\ j \in 1..\texttt{ntasks}(\psi)}} w_{\psi,j} \times r[\psi][j] + \mathtt{nb} \qquad (12)$$

Dans cette variante nous ajoutons la contrainte 11 et nous changeons la fonction objectif en 12. Nous noterons cet ensemble de contraintes not-break.

4 Expérimentations

Cette section présente quelques résultats expérimentaux concernant notre modèle et ses variantes sur des instances réelles d'ADP. Pour réaliser les expériences, nous avons utilisé le solveur de contraintes ACE, qui est le nouvel avatar de AbsCon [2]. Les options par défaut du solveur ont été utilisées : $\texttt{wdeg}^{\texttt{ca.cd}}$ [34] comme heuristique de choix de variables, lexico comme heuristique de choix de valeurs, last-conflict (lc) [17] comme méthode paresseuse pour simuler un retour en arrière intelligent, solution-saving [32, 9] pour simuler une forme de recherche de voisinage, et une politique de redémarrage géométrique [33] avec un cutoff de base fixé à 10 mauvaises décisions et une raison fixée à 1.1. Une seule option a été modifiée de manière empirique : Bivs [12] comme heuristique d'ordonnancement des valeurs. Bivs est une heuristique de choix de valeurs qui sélectionne la valeur dont

2. https ://githubs.com/xcsp3team/ace

Instances	# Tâches	# Variables	# Contraintes
2A - 1 jour	34	180	483
2A - 2 jours	68	362	932
2A - 3 jours	105	570	1459
2A - 7 jours	267	1398	3749
2A - 14 jours	552	2868	7777

TABLE 3 – Description des instances de $\mathcal{I}_{no-variant}$

l'affectation, une fois propagée, offre la plus petite (resp. grande) borne inférieure (resp. supérieure) pour la variable objectif. Nous noterons $\texttt{ACE}^{\texttt{BIVS}}$ lorsque l'éxécution a été réalisée avec cette heuristique.
Enfin le temps limite a été fixé à 180s afin de respecter les temps de résolution imposés par ADP.

4.1 A propos des instances

Le système actuel d'ADP permet de planifier au maximum 2 semaines d'utilisation des parkings. En raison de la quantité de vols, il ne permet pas de faire l'ensemble de l'aéroport en une seule fois, les planifications sont toujours effectuées terminal par terminal ou par groupe de terminaux.
Pour notre étude, nous nous sommes restraints au terminal $2A$ de CDG. L'étude est composée de 5 instances faisant respectivement 1, 2, 3,7 et 14 jours de planification.
L'étude est composée de 3 variantes du modèle comme présenté dans la section précédente. Nous notons $\mathcal{I}_{no-variant}$ les instances de la variante définie dans la section 3.2. L'ensemble des instances de la variante de la section 3.4 est noté $\mathcal{I}_{variant-breaking}$. Enfin l'ensembe des instances de la section 3.3 est noté $\mathcal{I}_{variant-alldifferent}$.

4.2 Solution d'ADP

Le système d'ADP utilise un solveur MILP (*Mixed-Integer Linear Programming*) pour résoudre le problème. La résolution s'arrête dès qu'une première solution est trouvée. En fonction du paramètrage, il est possible de faire une recherche locale pour améliorer la solution. Cette recherche locale vient notamment regrouper les tâches sur un même parking, pour éviter de « casser » des rotations. Nous noterons $\texttt{ADP}_{default}$ le solveur d'ADP sans recherche locale et $\texttt{ADP}_{ls}$ le solveur d'ADP avec recherche locale.

4.3 Comparaison entre la solution d'ADP et les modèles proposés

Nous allons d'abord comparer les modèles COP proposés avec la solution d'ADP en faisant varier également l'heuristique de choix de valeur puis nous ferons une comparaison des modèles proposés entre eux toujours en faisant varier l'heuristique de choix de valeur.

Étude de $\mathcal{I}_{no-variant}$ La figure 3a montre le score maximal obtenu du premier modèle COP proposé ($\mathcal{I}_{no-variant}$) par rapport à la solution trouvée par ADP. Nous pouvons observer que ACE dépasse les performances de la méthode actuelle. Notons que des optima sont trouvés pour les instances 1, 2 et 3 jours. La figure 3b présente les résultats en utilisant $\texttt{ACE}^{\texttt{BIVS}}$. Là encore, nous pouvons observer que les résultats de notre modèle sont meilleurs que ceux de la solution actuelle.

Étude de $\mathcal{I}_{variant-alldiff}$ La figure 3c montre le score maximal obtenu du second modèle COP proposé ($\mathcal{I}_{no-alldiff}$) qui ajoute les contraintes `AllDifferent` entre toutes les cliques maximales du graphe d'intervalles. Nous pouvons observer que là encore ACE dépasse les performances de la méthode actuelle. Notons que des optima sont trouvés pour les instances 1, 2 et 3 jours. La figure 3d présente les résultats en utilisant $\texttt{ACE}^{\texttt{BIVS}}$. Nous pouvons observer aussi que les résultats de notre modèle sont meilleurs que ceux de la solution actuelle.

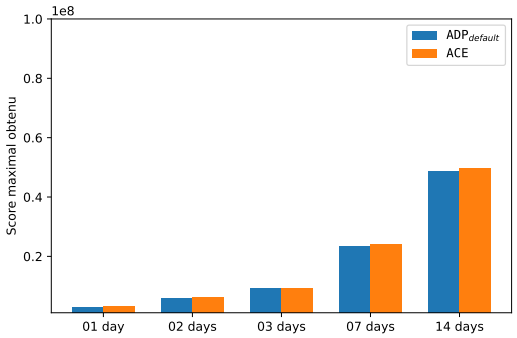
Étude de $\mathcal{I}_{variant-notbreak}$ La figure 3e montre le score maximal obtenu par le second modèle COP proposé prenant en compte le fait de ne pas « casser » les rotations par rapport à la solution d'ADP avec recherche locale dédiée. La figure 3f présente la même approche mais avec $\texttt{ACE}^{\texttt{BIVS}}$. Dans les deux cas, l'approche générique sans recherche locale dépasse la solution utilisée actuellement.
Nous pouvons dire que peu importe la combinaison de l'approche que nous proposons, on constate une amélioration de la borne et nous supprimons la nécessité d'avoir une recheche locale dédiée et donc une phase supplémentaire de calcul comme c'est le cas actuellement.

4.4 Comparaison des modèles proposés

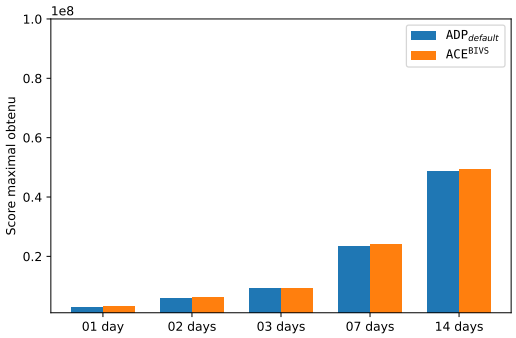
Dans cette dernière partie des experimentations, nous allons comparer les approches proposées entre elles en retirant les solutions d'ADP.
La figure 4 présente l'évolution des bornes trouvées en fonction du temps pour chaque instance du problème. Nous comparons les différentes approches proposées. Nous pouvons remarquer que sur la première instance (figures 4a), pour laquelle un optimum est trouvé, les différentes approches ont un comportement équivalent. Sur les instances de 2 et 3 jours (figures 4b et 4c), nous pouvons observer que les approches avec les contraintes `AllDifferent` permettent de converger plus vite vers l'optimum. Enfin l'utilisation de `Bivs` permet de converger encore plus vite vers l'optimum.
La différence est plus marquée sur les deux instances « difficiles » (figure 4d et 4e). Pour l'instance de 7 jours, les approches `AllDifferent` avec ou sans l'utilisation de `Bivs` stagnent à la première solution trouvée. Dans le cas de l'instance 14 jours, l'approche `AllDifferent` trouve initialement une meilleure solution que les autres approches puis stagnent à la même valeur jusqu'à la fin alors les autres approches progressent.

5 Conclusion

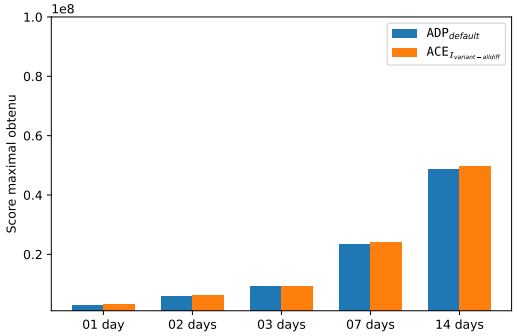
Dans cet article, nous nous intéressons au problème d'affectation des parkings d'aéroport (APAP) tel que défini à l'aéroport de Paris CDG. Nous avons proposé plusieurs variantes d'un modèle d'optimisation par contraintes (COP), en exploitant principalement les contraintes de table, et nous avons présenté une première évaluation empirique de notre approche. Nous avons également montrer que le pro-



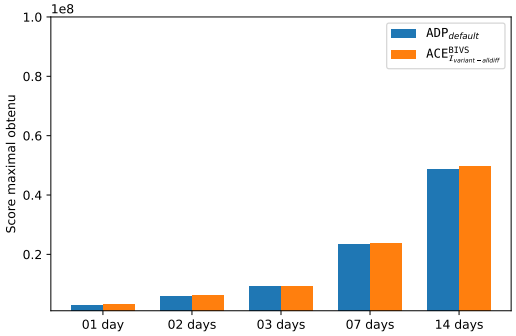
(a) Comparaison des bornes obtenues sur l'ensemble $\mathcal{I}_{no-variant}$ par ACE et ADP$_{default}$



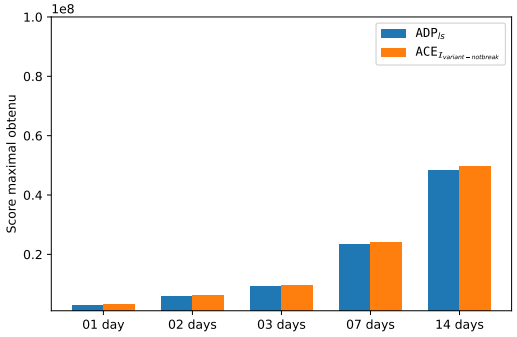
(b) Comparaison des bornes obtenues sur l'ensemble $\mathcal{I}_{no-variant}$ par ACE^{BIVS} et ADP$_{default}$



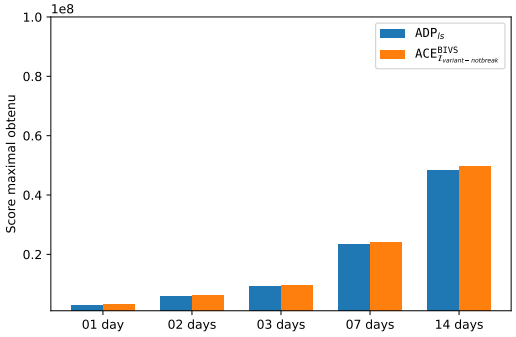
(c) Comparaison des bornes obtenues sur l'ensemble $\mathcal{I}_{variant-alldiff}$ par ACE et ADP



(d) Comparaison des bornes obtenues sur l'ensemble $\mathcal{I}_{variant-alldiff}$ par ACE^{BIVS} et ADP$_{default}$

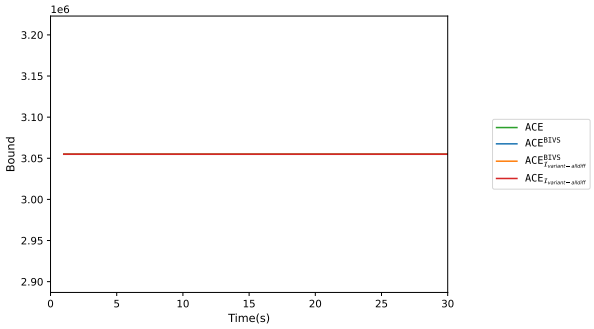


(e) Comparaison des bornes obtenues sur l'ensemble $\mathcal{I}_{variant-notbreak}$ par ACE et ADP$_{ls}$

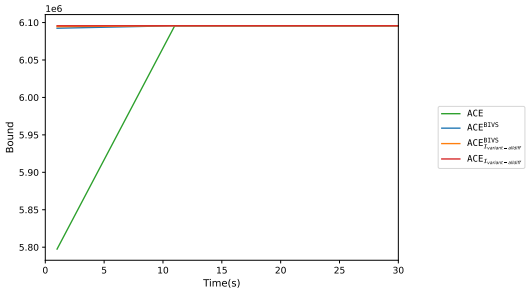


(f) Comparaison des bornes obtenues sur l'ensemble $\mathcal{I}_{variant-notbreak}$ par ACE^{BIVS} et ADP$_{ls}$

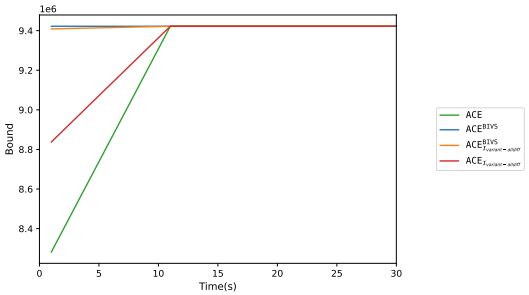
FIGURE 3 – Comparaison des différentes approches avec la solution d'ADP



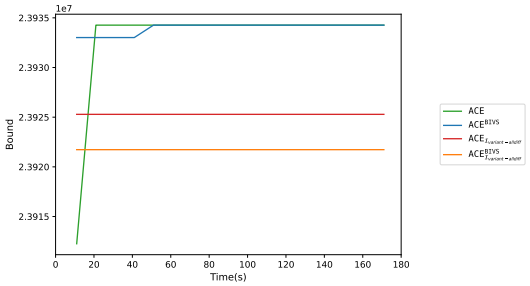
(a) Évolution de la borne en fonction du temps pour l'instance 1 jour



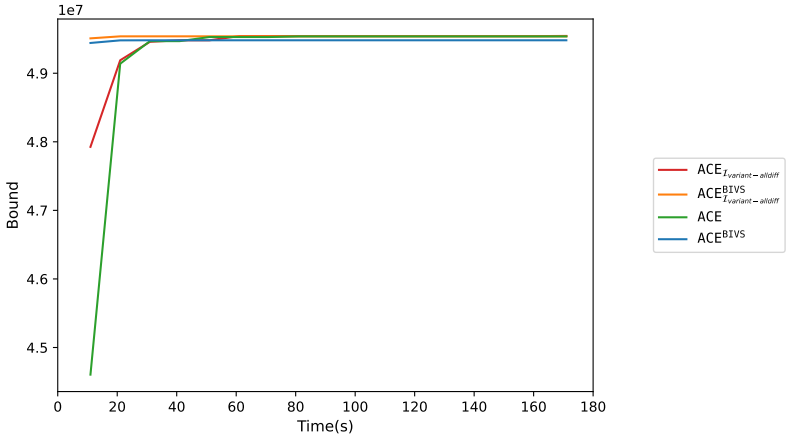
(b) Évolution de la borne en fonction du temps pour l'instance 2 jours



(c) Évolution de la borne en fonction du temps pour l'instance 3 jours



(d) Évolution de la borne en fonction du temps pour l'instance 7 jours



(e) Évolution de la borne en fonction du temps pour l'instance 14 jours

FIGURE 4 – Evolution des bornes en fonction du temps pour l'ensemble des instances

blème n'était pas équivalent au problème de l'état de l'art et que l'on ne pouvait pas directement utiliser les approches de l'état de l'art basée sur des contraintes `AllDifferent` pour toutes les tâches qui se chevauchent. Cependant une approche avec des contraintes `AllDifferent` et le calcul des cliques maximum du graphe d'intervalles est possible. Les résultats obtenus insitent, le groupe ADP à remplacer sa solution propriétaire actuelle par la nôtre, basée sur un solveur de contraintes générique open-source et l'utilisation d'un format open-source XCSP facilitant la modélisation des problèmes d'optimisations et leurs modifications. Le problème d'affectation des parkings est une première étape. Dans le futur, l'objectif est de remplacer l'ensemble des optimisations actuelles par une approche COP. Parmi les ressources aéroportuaires à optimiser en plus des parkings avion, nous travaillons sur des solutions pour les banques d'enregistrements, les tapis et les jetées bagages mais également les flux passagers.

Références

[1] Abdelghani Bouras, Mageed A. Ghaleb, Umar S. Suryahatmaja, and Ahmed M. Salem. The Airport Gate Assignment Problem : A Survey. *The Scientific World Journal*, 2014 :e923859, November 2014.

[2] F. Boussemart, C. Lecoutre, G. Audemard, and C. Piette. XCSP^3 : An Integrated Format for Benchmarking Combinatorial Constrained Problems. Technical Report arXiv :1611.03398, Specifications, CoRR, 2016.

[3] F. Boussemart, C. Lecoutre, G. Audemard, and C. Piette. XCSP^3-core : A Format for Representing Constraint Satisfaction/Optimization Problems. Technical Report arXiv :2009.00514, Specifications 3.0.6, CoRR, 2020.

[4] B. Le Charlier, M. T. Khong, C. Lecoutre, and Y. Deville. Automatic Synthesis of Smart Table Constraints by Abstraction of Table Constraints. In *Proceedings of IJCAI'17*, pages 681–687, 2017.

[5] G. Chu, P. Stuckey, M. Garcia de la Banda, and C. Mears. Symmetries and Lazy Clause Generation. In *Proceedings of IJCAI'11*, pages 516–521, 2011.

[6] Gülesin Sena Daş, Fatma Gzara, and Thomas Stützle. A review on airport gate assignment problems : Single versus multi objective approaches. *Omega*, 92 :102146, April 2020.

[7] J. Demeulenaere, R. Hartert, C. Lecoutre, G. Perez, L. Perron, J.-C. Régin, and P. Schaus. Compact-Table : Efficiently Filtering Table Constraints with Reversible Sparse Bit-Sets. In *Proceedings of CP'16*, pages 207–223, 2016.

[8] J. Demeulenaere, R. Hartert, C. Lecoutre, G. Perez, L. Perron, J.-C. Régin, and P. Schaus. Compact-Table : Efficiently Filtering Table Constraints with Reversible Sparse Bit-Sets. In *Proceedings of CP'16*, pages 207–223, 2016.

[9] E. Demirovic, G. Chu, and P. Stuckey. Solution-based phase saving for CP : A value-selection heuristic to simulate local search behavior in complete solvers. In *Proceedings of CP'18*, pages 99–108, 2018.

[10] Guido Diepen, J.M. Akker, J.A. Hoogeveen, and J.W. Smeltink. Using column generation for gate planning at Amsterdam Airport Schiphol. January 2007.

[11] Mehmet Dincbas and Helmut Simonis. Apache - a constraint based, automated stand allocation system. pages 267–282, 10 1991.

[12] Jean-Guillaume Fages and Charles Prud'Homme. Making the First Solution Good ! In *2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 1073–1077, Boston, MA, November 2017. IEEE.

[13] A. Frisch, M. Grum, C. Jefferson, B. Martinez Hernandez, and I. Miguel. The design of ESSENCE : A constraint language for specifying combinatorial problems. In *Proceedings of IJCAI'07*, pages 80–87, 2007.

[14] J. Guépet, R. Acuna-Agost, O. Briant, and J.P. Gayon. Exact and heuristic approaches to the airport stand allocation problem. *European Journal of Operational Research*, 246(2) :597–608, 2015.

[15] C. Lecoutre. *ACE : A Generic Constraint Solver*. Technical Report. v1on CoRR, to appear, October 2021.

[16] C. Lecoutre, C. Likitvivatanavong, and R. Yap. STR3 : A path-optimal filtering algorithm for table constraints. *Artificial Intelligence*, 220 :1–27, 2015.

[17] C. Lecoutre, L. Sais, S. Tabary, and V. Vidal. Reasoning from last conflict(s) in constraint programming. *Artificial Intelligence*, 173(18) :1592–1614, 2009.

[18] C. Lecoutre and N. Szczepanski. *PYCSP^3 : Modeling Combinatorial Constrained Problems in Python*. Technical Report. v1 on CoRR, arXiv :2009.00326, September 2020. 100 pages.

[19] Christophe Lecoutre. STR2 : Optimized Simple Tabular Reduction for Table Constraints. *Constraints*, 16(4) :341–371, 2011.

[20] Chendong Li. Airport Gate Assignment : New Model and Implementation. *arXiv :0811.1618 [cs]*, November 2008.

[21] Chendong Li. Airport Gate Assignment A Hybrid Model and Implementation. *arXiv :0903.2528 [cs]*, March 2009.

[22] A. Lim, B. Rodrigues, and Y. Zhu. Airport Gate Scheduling with Time Windows. *Artif Intell Rev*, 24(1) :5–31, September 2005.

[23] J. L'Ortye, M. Mitici, and H.G. Visser. Robust flight-to-gate assignment with landside capacity constraints. *Transportation Planning and Technology*, 44(4) :356–377, May 2021.

[24] R. S. Mangoubi and Dennis F. X. Mathaisel. Optimizing Gate Assignments at Airport Terminals. *Transportation Science*, 19(2) :173–188, 1985.

[25] N. Nethercote, P. Stuckey, R. Becket, S. Brand, G. Duck, and G. Tack. MiniZinc : Towards a Standard CP Modelling Language. In *Proceedings of CP'07*, pages 529–543, 2007.

[26] OscaR Team. OscaR : Scala in OR, 2012.

[27] C. Prud'homme, J.-G. Fages, and X. Lorca. Chocosolver, TASC, INRIA Rennes, LINA, Cosling S.A. 2016.

[28] Helmut Simonis. Models for global constraint applications. *Constraints*, 12(1) :63–92, mar 2007.

[29] P. Stuckey, R. Becket, and J. Fischer. Philosophy of the MiniZinc challenge. *Constraints*, 15(3) :307–316, 2010.

[30] P. van Hentenryck. *The OPL Optimization Programming Language*. The MIT Press, 1999.

[31] H. Verhaeghe, C. Lecoutre, and P. Schaus. Extending Compact-Table to Negative and Short Tables. In *Proceedings of AAAI'17*, pages 3951–3957, 2017.

[32] J. Vion and S. Piechowiak. Une simple heuristique pour rapprocher DFS et LNS pour les COP. In *Proceedings of JFPC'17*, pages 39–45, 2017.

[33] T. Walsh. Search in a small world. In *Proceedings of IJCAI'99*, pages 1172–1177, 1999.

[34] H. Wattez, C. Lecoutre, A. Paparrizou, and S. Tabary. Refining constraint weighting. In *Proceedings of ICTAI'19*, pages 71–77, 2019.

[35] N. F. Zhou, H. Kjellerstrand, and J. Fruhman. *Constraint Solving and Planning with Picat*. Springer, 2017.