Nicolas Beldiceanu

De l'influence de Jacques Pitrat sur mes recherches en programmation par contraintes

Volume 3, nº 1-2 (2022), p. 141-153.

<http://roia.centre-mersenne.org/item?id=ROIA_2022__3_1-2_141_0>

Revue Ouverte d'Intelligence Artificielle
Volume 3, nº 1-2, 2022, 141-153

De l'influence de Jacques Pitrat sur mes recherches en programmation par contraintes

Nicolas Beldiceanu[a]

[a] IMT Atlantique, LS2N DAPI 4 rue Alfred Kastler 44307 Nantes, France
E-mail : nicolas.beldiceanu@imt-atlantique.fr
URL *:* https://cv.archives-ouvertes.fr/nicolasbeldiceanu.

Résumé. — Dans cet article, je montre comment les enseignements et les travaux de Jacques Pitrat ont orienté ma carrière et ont nourri mes réflexions et mes interrogations dans le domaine de la programmation par contraintes. Après un bref rappel des idées clefs que j'ai retenues, je montre comment j'ai utilisé ces idées tout au long de mon parcours, et j'indique pourquoi ces idées sont toujours d'actualité.

Mots-clés. — Contraintes, Métaconnaissances, Programmation par contraintes.

1. Introduction

J'ai eu la chance de suivre le cours de Jacques Pitrat en auditeur libre alors que je faisais le DEA système à Paris 6 en 1983–1984, puis son séminaire mensuel d'intelligence artificielle lorsque j'ai poursuivi en thèse avec Claude Girault entre les années 1985 et 1988. Ce qui m'a le plus marqué dans le cours de Jacques Pitrat, c'est la passion que l'on pouvait y ressentir, l'étendue des connaissances présentées, et le fait qu'il y transmettait des idées bien au-delà de la technique pure et simple.

Cet article explique comment ce cours a orienté ma carrière et nourrit mes réflexions et mes interrogations dans le domaine des contraintes jusqu'à aujourd'hui. Il se compose des trois parties suivantes.

- Les enseignements que j'ai gardés et qui se sont propagés dans mes travaux.
- En quoi ces enseignements sont toujours d'actualité.
- Les réflexions que cela a suscitées dans mon cas par rapport à la démarche de chercheur.

2. Enseignements de Jacques Pitrat

Dans cette section, je donne dans un premier temps sept idées clefs que j'ai retenues de ses enseignements, puis je montre comment j'ai réutilisé la plupart d'entre elles dans le cadre de mes travaux en programmation par contraintes en m'y rapportant.

2.1. Idées clefs que j'ai retenues

(i) Dissocier l'aspect définition de l'aspect calcul.
(ii) Séparer les connaissances de leur utilisation.
(iii) Décrire les choses de manière explicite.
(iv) Ne pas « idôlatrer » un formalisme particulier.
(v) Se détacher de la vue « un problème–un algorithme ».
(vi) Concevoir des systèmes dans lesquels une partie du code est synthétisée.
(vii) Prendre conscience des connaissances que l'on injecte dans un système.

2.2. Réutilisation des idées de Jacques Pitrat en programmation par contraintes

Trop souvent en programmation par contraintes, nous brûlons les étapes en allant directement à un algorithme décrivant comment filtrer une ou plusieurs contraintes spécifiques sans définir clairement ces contraintes. Même si l'effet à court terme est d'obtenir rapidement quelque chose d'opératoire, ce biais présente sur la durée plusieurs inconvénients majeurs : on se prive de raisonner de manière symbolique sur des définitions et des concepts, on se limite à une utilisation particulière, et l'on obtient ainsi des systèmes gravés dans le marbre cantonnés à un usage trop spécifique [38, 40]. Mis en perspective avec les travaux sur les contraintes globales cela peut s'interpréter comme suit.

2.2.1. *Décennie 1990–2000 : contraintes globales et algorithmes dédiés*

À l'origine, les contraintes globales furent introduites dans un contexte industriel [1, 8] pour deux raisons :

- D'une part, pour éviter de devoir modéliser un problème dans un cadre éloigné de la formulation du problème de départ. Cette idée, cf. (iv) en Section 2.1, était de fait déjà promue dans les travaux de Jacques Pitrat et Jean-Louis Laurière dans le cadre de la programmation par contraintes.
- D'autre part, pour avoir des algorithmes dédiés filtrant mieux certaines contraintes apparaissant dans nombre de problèmes combinatoires [45]. Si cette idée n'est pas dans la ligne des travaux de Jacques Pitrat, elle rejoint malgré tout la recommandation d'éviter de se couler dans un seul formalisme pour bénéficier de méthodes développées dans la résolution exacte de certains sous-problèmes combinatoires.

Si cet aspect algorithmique a suscité un engouement [28] en même temps que quelques critiques [21, 50], il a occulté un point bien plus important développé dans la décennie suivante.

2.2.2. *Décennie 2001–2011 : description de familles de contraintes et synthèse de propagateurs*

Le passage d'un cadre industriel à un environnement académique a été l'occasion pour moi de prendre du recul, et ce, à la lumière de l'enseignement de Jacques Pitrat, sur les développements algorithmiques que j'avais menés. Ces interrogations ont conduit à la création du catalogue de contraintes [13] qui essayait de décrire le sens des contraintes. Les recherches sur la description explicite, cf. (iii) en Section 2.1, du sens des contraintes globales en termes de graphes [9], d'automates [11, 37] et de formules logiques quantifiées [20] ont mis en avant la pertinence des contraintes globales comme *outil de modélisation d'un problème indépendamment de toute méthode de résolution*. Ce découplage entre la définition explicite du sens d'une contrainte et sa mise en œuvre technique, cf. (i) en Section 2.1, dans un cadre opérationnel particulier a permis les avancées suivantes :

- ce découplage promeut les contraintes globales en tant que mots et concepts commodes pour modéliser un problème combinatoire indépendamment de toute technique de résolution. L'exemple 2.1 illustre ce point en s'appuyant sur la contrainte ALLDIFFERENT.
- ce découplage permet de définir une variété de contraintes adaptées aux spécificités d'un problème en introduisant de nouvelles définitions de contraintes. De la même manière que dans certains pays froids il existe une variété de mots pour désigner différentes nuances de neige [29], une contrainte n'est pas nécessairement éloignée de tout contexte.
- ce découplage a ouvert la voie à des langages de modélisation de problèmes combinatoires qui sont complètement indépendants d'une technologie donnée. C'est par exemple le cas du langage de modélisation MiniZinc [35] qui, bien qu'offrant plus d'une centaine de contraintes globales, permet d'utiliser un même et unique modèle avec différents solveurs provenant de différentes technologies de résolution de problèmes (PPC, MIP, ou SAT).

Exemple 2.1. — La contrainte ALLDIFFERENT [32] impose à ses variables de prendre des valeurs entières distinctes. Pour cette même et unique définition, nous donnons huit contextes d'utilisation de la contrainte ALLDIFFERENT.

(1) **Test** : l'utilisation la plus simple consiste juste à s'assurer qu'une contrainte ALLDIFFERENT dont toutes les variables sont fixées est vérifiée ou pas. Par exemple, un programme testant la contrainte ALLDIFFERENT indiquera que l'instance ALLDIFFERENT $(\langle 3, 5, 1\rangle)$ est bien satisfaite, alors que l'instance ALLDIFFERENT $(\langle 1, 5, 1\rangle)$ ne l'est pas.

(2) **Faisabilité** : une deuxième utilisation lorsque les variables ne sont pas toutes déjà fixées consiste à tester si la contrainte est à coup sûr insatisfaite au vu des valeurs pouvant être prises couramment par ses variables. Par exemple, un programme vérifiant la faisabilité de la contrainte ALLDIFFERENT signalera que l'instance ALLDIFFERENT $(\langle x_1, x_2, x_3\rangle)$ avec $x_1, x_2, x_3 \in [1, 2]$ ne peut jamais être satisfaite.

(3) **Filtrage** : une troisième utilisation consiste à identifier les couples (<variable>, <valeur>) (x_i, v_j), tels que si j'affecte la valeur v_j à la variable x_i alors la contrainte ALLDIFFERENT ne pourra jamais être satisfaite [22, 45]. Par exemple, un programme filtrant les variables de la contrainte ALLDIFFERENT identifiera qu'il n'est pas possible d'affecter les valeurs 1 et 2 à la variable x_3 dans le cas de la contrainte ALLDIFFERENT $(\langle x_1, x_2, x_3\rangle)$ avec $x_1, x_2 \in [1, 2]$ et $x_3 \in [1, 3]$.

(4) **Explication** : une quatrième utilisation consiste à expliquer le filtrage de la contrainte ALLDIFFERENT. Par exemple, le filtrage de la valeur 3 de la variable x_2 dans le cas de la contrainte ALLDIFFERENT $(\langle x_1, x_2, x_3, x_4, x_5\rangle)$ avec $x_1, x_2 \in [1, 3]$, $x_3, x_4 \in [3, 4]$, $x_5 \in [5, 6]$ (et en supposant que les variables x_3 et x_4 pouvaient initialement prendre toutes les valeurs entre 1 et 6) sera expliqué par le fait que les valeurs 1, 2, 5 et 6 ne font pas partie des valeurs possibles pour les variables x_3 et x_4 : en effet, si l'une de ces valeurs pouvait être prise par x_3 ou x_4, le filtrage de la valeur 3 des valeurs possibles de x_2 ne pourrait plus avoir lieu.

(5) **Degré de violation** : une cinquième utilisation pour une contrainte ALLDIFFERENT violée dont toutes les variables sont déjà fixées consiste à déterminer le nombre minimum de variables dont il faut changer la valeur pour retrouver une solution [14]. Par exemple, dans l'instance ALLDIFFERENT $(\langle 2, 5, 2, 2, 5\rangle)$ qui n'est pas satisfaite, il faudra réaffecter au moins trois variables pour pouvoir satisfaire la contrainte ALLDIFFERENT.

(6) **Réification** : une sixième utilisation lorsque l'on doit, par exemple, relaxer certaines contraintes, consiste à associer une variable 0-1 b à une contrainte CTR, et à trouver une reformulation exprimant le fait que b vaut 1 si et seulement si la contrainte CTR est satisfaite. Une réification compacte de la contrainte ALLDIFFERENT $(\langle x_1, x_2, \ldots, x_n\rangle)$ utilise la contrainte de tri [18, 36] : SORT $(\langle x_1, x_2, \ldots, x_n\rangle, \langle y_1, y_2, \ldots, y_n\rangle) \wedge ((y_1 < y_2 < \cdots < y_n) \Leftrightarrow b)$. La contrainte SORT impose que les variables $y_1, y_2, \ldots, y_n$ correspondent à une permutation des variables $x_1, x_2, \ldots, x_n$ vérifiant la condition $y_1 \leqslant y_2 \leqslant \cdots \leqslant y_n$. La deuxième contrainte impose un inégalité stricte entre deux y consécutifs.

(7) **Dénombrement** : un septième usage consiste à estimer le nombre de solutions d'une contrainte ; en effet, certaines heuristiques d'affectation de valeurs aux variables nécessitent l'identification des sous-problèmes les plus contraints. Par exemple, la contrainte ALLDIFFERENT $(\langle x_1, x_2, x_3\rangle)$ avec $x_1, x_2 \in [1, 2]$, $x_3 \in [2, 4]$ possède quatre solutions. Comme le problème équivalent de dénombrer le nombre de couplages maximum d'un graphe biparti est #*P*-complet [49] on utilise des estimateurs pour évaluer le nombre de solutions de la contrainte ALLDIFFERENT [51].

(8) **Reformulation** : un huitième usage revient à reformuler une contrainte en termes de contraintes linéaires et/ou de variables 0-1 ceci, par exemple, pour utiliser un même modèle mentionnant certaines contraintes globales avec un solveur MIP ou SAT. Une reformulation de ALLDIFFERENT$(\langle x_1, x_2, \ldots, x_n\rangle)$ en termes de variables 0-1 et de contrainte linéaires est la suivante [17].

Pour *chaque* intervalle $[\ell, u]$ de valeurs pouvant être prises par les variables $x_1, x_2, \ldots, x_n$ on a :

- n variables 0-1 $b_{1,\ell,u}, b_{2,\ell,u}, \ldots, b_{n,\ell,u}$ pour représenter que chaque variable $x_1, x_2, \ldots, x_n$ prend une valeur dans l'intervalle $[\ell, u]$ (c'est-à-dire, $\forall i \in [1, n] \ : b_{i,\ell,u} \Leftrightarrow x_i \in [\ell, u]$).
- Une contrainte d'inégalité pour imposer la condition que la somme de ces variables 0-1 soit plus petite ou égale à la taille $u - \ell + 1$ de l'intervalle (c'est-à-dire $b_{1,\ell,u} + b_{2,\ell,u} + \cdots + b_{n,\ell,u} \leqslant u - \ell + 1$).

Mon intérêt pour la génération de code à partir de connaissances, cf. (vi) en Section 2.1, prend son origine dans un projet que donnait Jacques Pitrat dans son cours, à savoir partant d'une même représentation déclarative des formes des mots d'une langue choisie, écrire deux programmes (1) l'un reconnaissant la ou les formes d'un mot donné, et l'autre (2) pouvant également générer un mot de forme particulière. Bien que la première version du catalogue de contraintes était une version purement textuelle, j'avais bien en tête d'avoir une représentation du catalogue en termes de métadonnées. J'ai défini ces métadonnées et mon collègue Mats Carlsson a écrit le premier programme synthétisant le catalogue de contraintes à partir de ces métadonnées, programme que j'ai repris et étendu par la suite. Un autre événement m'a conduit à m'intéresser à prendre le contrepied d'une approche algorithmique pure et dure pour filtrer des contraintes : un algorithme dédié filtrant de manière complète la contrainte d'ordre lexicographique entre deux vecteurs [27] avait été présenté à la conférence sur les contraintes en 2002. Avec mon collègue Mats Carlsson, nous avons montré que l'on pouvait simplement obtenir le même filtrage tout en ayant une performance opérationnelle équivalente, en exprimant cette contrainte d'ordre lexicographique en termes d'automate, et en reformulant le comportement induit par l'automate comme une conjonction de contraintes de transitions et de signatures [10, 19] tel qu'illustré par l'exemple 2.2 dans le cadre de la contrainte d'ordre lexicographique, cf. (v) en Section 2.1. Toujours dans la même ligne de travaux sur la synthèse de propagateurs, nous avons décrit en 2008 comment compiler un langage de règles basé sur l'arithmétique et la logique du premier ordre en formules arithmétiques de Presburger sans quantificateurs pour synthétiser des générateurs d'ensembles de points interdits, et pour finalement filtrer un vecteur de variables en utilisant un algorithme de balayage multidimensionnel réduit à sa plus simple expression [20].

Exemple 2.2. — Étant donnés deux vecteurs de variables $\langle x_0, x_1, \ldots, x_{n-1}\rangle$ et $\langle y_0, y_1, \ldots, y_{n-1}\rangle$, la contrainte d'ordre lexicographique LEX_LESSEQ($\langle x_0, x_1, \ldots, x_{n-1}\rangle$, $\langle y_0, y_1, \ldots, y_{n-1}\rangle$) est satisfaite si et seulement si *(i)* $n = 0$, ou *(ii)* $x_0 < y_0$, ou bien *(iii)* $x_0 = y_0$ et $\langle x_1, \ldots, x_{n-1}\rangle$ est lexicographiquement plus petit ou égal à $\langle y_1, \ldots, y_{n-1}\rangle$. Cette contrainte LEX_LESSEQ peut se représenter par l'automate donné dans la partie (A) de la Figure 2.1 ; la reformulation de cet automate en termes d'un réseau de contraintes est donnée dans la partie (B) de cette même figure.

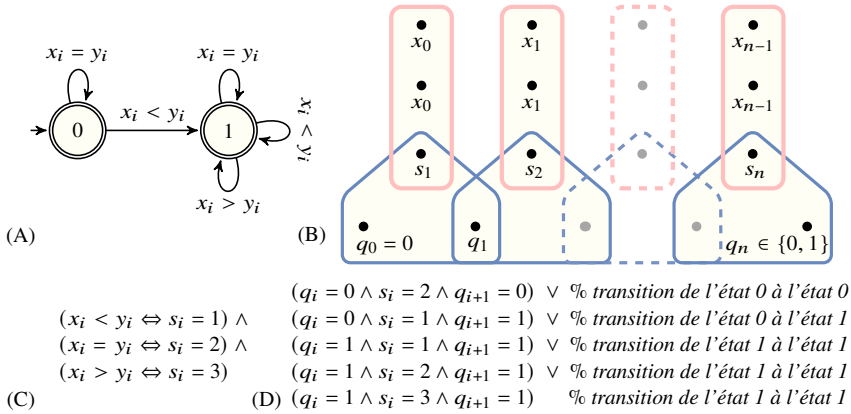

FIGURE 2.1. (A) Automate acceptant uniquement toutes les solutions de la contrainte LEX_LESSEQ($\langle x_0, x_1, \ldots, x_{n-1}\rangle, \langle y_0, y_1, \ldots, y_{n-1}\rangle$) ; (B) Hypergraphe associé à la reformulation de l'automate de la contrainte LEX_LESSEQ avec ses contraintes de signature comparant deux composantes des vecteurs $\langle x_0, x_1, \ldots, x_{n-1}\rangle$ et $\langle y_0, y_1, \ldots, y_{n-1}\rangle$ (en rose), et ses contraintes de transition représentant le franchissement d'une transition (en bleu), respectivement détaillées dans les parties (C) et (D) ; les variables $s_1, s_2, \ldots, s_n$ correspondent aux variables de signature encodant les conditions associées aux transitions successivement franchies ; finalement, les variables $q_0, q_1, \ldots, q_n$ donnent les états consécutivement traversés lorsque l'automate consomme les composantes des deux vecteurs de la contrainte LEX_LESSEQ.

2.2.3. *Décennie 2012–2021 : description des propriétés des contraintes, synthèse d'invariants et de modèles*

A) L'essor de l'analyse de données a conduit à se pencher sur la question de l'acquisition d'un modèle à contraintes à partir d'exemples d'un nombre restreint de solutions et non-solutions d'un problème. Cette interrogation a abouti au développement du *constraint seeker* [15] et du *model seeker* [16], deux outils permettant, pour le premier d'identifier et de classer les contraintes globales pertinentes par rapport à des exemples positifs et négatifs, et pour le second d'extraire un modèle à contraintes à partir d'un nombre réduit de solutions. Dans les deux cas, l'une des difficultés principales consistait à identifier les contraintes et modèles non pertinents. Au lieu d'une approche directe consistant à concevoir des algorithmes dédiés à ces deux tâches, j'ai suivi la démarche promue par Jacques Pitrat, à savoir donner de manière déclarative un certain nombre de connaissances pertinentes dans ce contexte, cf. (ii) en Section 2.1. Dans un premier temps, j'ai complété les métadonnées décrivant les contraintes du catalogue en représentant également en termes de métadonnées des propriétés des contraintes globales telles que :

- Les restrictions s'appliquant forcément aux arguments d'une contrainte.
- Les conditions typiquement vérifiées par les arguments d'une contrainte.

- Les dépendances fonctionnelles indiquant que certains arguments d'une contrainte sont déterminés fonctionnellement par un sous-ensemble d'arguments donnés.
- Les symétries de variables et de valeurs s'appliquant aux arguments d'une contrainte.
- Les propriétés essentielles d'une contrainte, telles la rétractabilité ou l'extensibilité indiquant respectivement si une instance satisfaite d'une contrainte reste toujours satisfaite lorsque l'on enlève ou l'on ajoute des variables dans certains de ses arguments [34].

En parallèle, mon collègue Helmut Simonis a écrit le *constraint seeker* et le *model seeker*, deux programmes s'appuyant essentiellement sur les métadonnées du catalogue de contraintes.

L'exemple 2.3 illustre quelques métadonnées utilisées pour décrire différents aspects des contraintes du catalogue en s'appuyant sur la contrainte ALLDIFFERENT.

Exemple 2.3. — À chaque contrainte du catalogue correspondent deux fichiers : un premier fichier Prolog donnant des métadonnées décrivant la contrainte concernée (par des faits Prolog) avec du code (Prolog, C et Java), ainsi qu'un fichier texte donnant des explications textuelles sur différents points. Si les volumes 1 et 2 du catalogue utilisent bien les mêmes éléments pour décrire une contrainte, ces éléments ont été écrits par un humain dans le cadre du volume 1, alors qu'ils ont été générés par un programme dans le cas du volume 2. Ces fichiers sont utilisés par un même programme pour synthétiser les deux volumes du catalogue. Ils servent également au constraint seeker [15] et au model seeker [16].

Voici un échantillon des métadonnées décrivant la contrainte ALLDIFFERENT en présentant la forme LaTeX synthétisée qui est plus lisible que les faits Prolog utilisés en interne.

• **Constraint**	ALLDIFFERENT (VARIABLES)
• **Argument**	VARIABLES : collection(var – dvar)
• **Restriction**	required (VARIABLES, var)
• **Typical**	|VARIABLES| > 2
• **Symmetries**	◇ Items of VARIABLES are permutable. ◇ Two distinct values of VARIABLES.var can be swapped; a value of VARIABLES.var can be renamed to any unused value.
• **Arg. properties**	Contractible wrt. VARIABLES.
• **See also**	◇ **implied by:** ALLDIFFERENT_CONSECUTIVE_VALUE, CIRCUIT, CYCLE, STRICTLY_DECREASING, STRICTLY_INCREASING. ◇ **implies:** ALLDIFFERENT_EXCEPT_0, NOT_ALL_EQUAL, MULTI_GLOBAL_CONTIGUITY. ◇ **negation:** SOME_EQUAL. ◇ **conditional implication:** ALLDIFFERENT (VARIABLES) ⇒ BALANCE (B, VARIABLES) when B = 0.

Le champ **Constraint** donne le nom de la contrainte et de ses arguments. Le champ **Argument** décrit le type de chaque argument, ici une collection de variables entières. Le champ **Restriction** indique des conditions toujours vérifiées par les arguments de la contrainte, ici le fait que l'attribut `var` doit être utilisé. Le champ **Typical** spécifie des conditions étant le plus souvent satisfaites sur les exemples usuels, ici que l'on a souvent plus de deux variables. Le champ **Symmetries** déclare des symétries valides pour la contrainte, ici une symétrie sur les variables, et une deuxième symétrie sur les valeurs prises par les variables. Le champ **Arg. properties** précise des propriétés de la contrainte, ici le fait qu'en partant de n'importe quelle instance d'une contrainte ALLDIFFERENT qui est vérifiée, retirer n'importe quelle variable donne une nouvelle instance qui est encore satisfaite. Le champ **See also** donne les contraintes impliquant la contrainte ALLDIFFERENT, les contraintes impliquées par la contrainte ALLDIFFERENT, la contrainte correspondant à la négation de la contrainte ALLDIFFERENT, et finalement les contraintes impliquées par la contrainte ALLDIFFERENT lorsqu'une certaine condition est vérifiée.

B) Le développement de l'analyse de flux de données a conduit à nous intéresser à la question de l'extraction de toutes les occurrences (maximales au sens de l'inclusion) d'un patron d'une séquence, ainsi qu'à la génération de séquences contenant ou pas certaines occurrences de différents patrons. Les patrons considérés ici alliaient un aspect qualitatif représenté à l'aide d'expressions régulières, à un aspect quantitatif décrit par une cascade de fonctions arithmétiques. Jusqu'à 2013, moi et mon collègue suédois représentions en dur chaque patron en termes d'un automate à registres. Tout s'est simplifié le jour ou nous avons compris que l'on gagnait à rendre explicite, cf. (iii) en Section 2.1, le sens d'une transition d'un automate à états finis : si quelques rares textes mentionnent bien le fait que l'on puisse attribuer un sens aux transitions d'un automate (par exemple, certaines transitions correspondent au fait que l'on avance dans le processus de reconnaissance d'un patron, alors que d'autres représentent une resynchronisation après un échec), aucun à notre connaissance ne rendait cette information explicite. En passant d'un automate à états finis à un transducteur dans lequel les lettres de l'alphabet de sortie correspondent aux phases de reconnaissance d'un patron, nous avons pu synthétiser, cf. (vi) en Section 2.1, avec plusieurs collègues :

- Un transducteur $\mathcal{T}$ en partant d'une classe d'expressions régulières $\mathcal{R}$ [26].
- Un automate à registres $\mathcal{A}$ en remplaçant les lettres de phases d'un transducteur $\mathcal{T}$ par un jeu d'instruction réduit sur des registres [12]. Ici, le transducteur peut être vu comme un méta-automate permettant d'engendrer plusieurs catégories d'automates adaptés à différents usages.
- Des bornes sur le résultat retourné par l'automate à registre $\mathcal{A}$ en se basant sur des caractéristiques de l'expression régulière $\mathcal{R}$ [6].
- Des invariants entre les deux quantités retournées par deux automates à registres $\mathcal{A}_1$ et $\mathcal{A}_2$ appliqués à une même séquence [5].
- Une matrice de colle $\mathcal{M}$ indexée par les états de deux automates à registres, faisant le lien entre les trois quantités suivantes : (1) le résultat d'un automate à registres $\mathcal{A}$ appliqué à toute une séquence S, (2) le résultat du même automate $\mathcal{A}$ appliqué à un préfixe de la séquence S, et finalement (3) le

résultat de l'automate inverse $\mathcal{A}^r$ appliqué à un suffixe de la séquence S tel que la concaténation du préfixe et du suffixe redonne S [2]. Cette matrice de colle se retrouve de manière implicite sous la forme d'un terme correctif dans nombre d'algorithmes sur des séquences qui combinent une information sur le préfixe d'une séquence S avec une information sur le suffixe de S pour obtenir une information sur la séquence complète S.

La génération de tels objets a été rendue possible en rendant explicite ce qui était là, implicite sous nos yeux depuis si longtemps : en attribuant un sens aux transitions d'un automate à états finis, et en détournant l'usage classique de l'alphabet de sortie d'un transducteur pour y encoder ce sens. Cela a conduit au deuxième volume de catalogue de contraintes qui a été complètement synthétisé et qui rend directement disponibles les transducteurs, les automates à registres, les bornes, les invariants et les matrices de colles sous forme de métadonnées [3]. Ces métadonnées ont été utilisées pour améliorer la performance de modèles à contraintes pour générer des séquences soumises à une conjonction de contraintes [7] sans utiliser le moindre algorithme dédié, cf. (v) en Section 2.1.

3. Actualité des enseignements de Jacques Pitrat

Le développement de services en ligne s'adaptant au profil de l'utilisateur a conduit les géants de l'internet à remettre les méthodes d'intelligence artificielle sur le devant de la scène. L'exploitation en temps réel de flux de données pour piloter et prendre des décisions a également participé à ce regain d'intérêt. Ironiquement, les analyses récentes sur les limitations actuelles des systèmes d'intelligence artificielle [47, 48] ont redécouvert le fait montré par Jacques Pitrat, cf. (vii) en Section 2.1, à savoir que l'injection explicite de connaissances dédiées est problématique [33, 39].

L'idée de construire un « programme chercheur » faisant, dans le domaine des contraintes, des expériences pour les analyser et pour ainsi détecter des régularités et émettre des hypothèses, et ceci sur plusieurs étapes, a été le dernier travail de Jacques Pitrat [41, 42]. Il va bien au delà des systèmes de configuration d'hyperparamètres dans le cadre de porte-folio de solveurs et d'identifications de catégories de problèmes, et est d'actualité pour au moins deux raisons :

- La construction de programmes de jeux consiste justement à faire jouer la machine contre elle-même sur plusieurs générations pour acquérir un programme pouvant jouer sans injecter de connaissances spécifiques.
- La construction de programmes construisant des conjectures pertinentes est également un domaine connaissant un regain d'intérêt [31, 44]. Dans ce cadre, un problème ouvert concerne le couplage entre, d'une part la génération de conjectures, et d'autre part la preuve automatique de ces conjectures. En effet, d'un côté les outils de preuves actuels sont plutôt des assistants de preuves, et d'un autre côté les méthodes de génération de conjectures fournissent trop peu d'éléments pouvant guider un assistant de preuves. Réussir à concilier la génération de conjectures avec la preuve automatique de ces conjectures semble être un élément clef pour :

– faire progresser l'intelligence artificielle symbolique avec des systèmes qui apprennent et prouvent de nouvelles connaissances dans un processus itéré d'amorçage.
– aller au-delà de la recherche de corrélations et de patrons dont on ne peut prouver la véracité.

4. Réflexion sur la démarche de chercheur et conclusion

Je me suis souvent demandé pourquoi les travaux de Jacques Pitrat ont suscité autant de critiques et n'ont pas eu plus de retentissement même si, par ailleurs, il est bien reconnu comme l'un des pionniers ayant amorcé la recherche en intelligence artificielle en France. Parmi ces raisons on peut citer pêle-mêle :

- Le poids des présupposés inconscients tenus pour acquis et la difficulté de s'en dégager. Je me suis par exemple demandé pourquoi il n'y avait toujours pas de bases de données de coupes pour des sous-structures particulières directement exploitables par des solveurs de différentes technologies.
- La propension à juger rapidement les travaux d'une communauté tout en étant en dehors. Mais chaque groupe scientifique possède son historique, sa feuille de route et sa logique propre ; un regard externe non nuancé n'est souvent pas constructif.
- Le fait de considérer un domaine juste par sa capacité à donner rapidement des solutions pour des problèmes bien délimités, ou bien à fournir des résultats désincarnés dans lesquels tous les tenants et les aboutissants ayant amené à ces conclusions seront gommés. Dans son cours, Jacques Pitrat nous a justement appris que les tenants et aboutissants conduisant à un résultat sont souvent aussi importants que le résultat lui-même, ceci pour au moins deux raisons : d'une part, pour éviter l'effet magique qui perd les gens, et d'autre part pour permettre de transférer la démarche à des situations approchées.
- La promotion du fait de coder comme un but en soit. Mais c'est la conjugaison des aspects métaconnaissances et métaprogrammation qui fournit un levier puissant pour obtenir des systèmes plus généraux.
- L'idée selon laquelle l'humain est seul capable d'intelligence.

Ce qui a été jugé irréaliste un jour change le lendemain. Pour ne citer que quatre exemples, on a bel et bien pu séparer dans une certaine mesure les aspects modélisation d'un problème de la technologie de résolution (CP, MIP, SAT) [35], même si des progrès restent encore à faire. On a pu également résoudre de manière exacte des problèmes ouverts de RCPSP en remplaçant des algorithmes dédiés par des approches ou la synergie entre un algorithme simple de cumul des parties obligatoires [30] expliqué et un solveur SAT a montré une efficacité certaine [46]. Les « ennemis déclarés » d'un temps ont été intégrés dans le solveur OR-Tools de Google qui utilise à la fois les aspects CP, SAP et MIP (voir la présentation CP-SAT à la master class sur les contraintes à CPAIOR2020). Finalement, les contraintes d'automates et les méthodes pour casser des symétries ont été transférées de la CP vers les solveurs MIP [23, 24].

Pour conclure, ce sont justement la ténacité de Jacques Pitrat, sa modestie et sa capacité à transmettre à ceux qui ont eu la chance d'assister à ces cours qui font sens : même si le succès est incertain et partiel, c'est le plaisir de développer ses idées au vu des progrès potentiels à la clef qui nous inspire.

Lewis Carroll, *Alice au pays des merveilles* :

– Inutile d'essayer, dit Alice. Qui pourrait croire en l'impossible ?

– Vous péchez, selon moi, par manque d'entraînement, dit la Reine. Quand j'avais votre âge, je m'y exerçais une demi-heure par jour. Eh bien, il m'est arrivé parfois, avant même l'heure du petit déjeuner, de croire jusqu'à six choses impossibles.

Bibliographie

[1] A. Aggoun & N. Beldiceanu, « Extending CHIP in order to Solve Complex Scheduling and Placement Problems », *Mathl. Comput. Modelling* **17** (1993), nº 7, p. 57-73.

[2] E. Arafailova, N. Beldiceanu, M. Carlsson, P. Flener, M. A. F. Rodríguez, J. Pearson & H. Simonis, « Systematic Derivation of Bounds and Glue Constraints for Time-Series Constraints », in *Principles and Practice of Constraint Programming (CP'2016)* (Toulouse, France) (M. Rueher, éd.), LNCS, vol. 9892, Springer-Verlag, 2016, p. 13-29.

[3] E. Arafailova, N. Beldiceanu, R. Douence, M. Carlsson, P. Flener, M. A. F. Rodríguez, J. Pearson & H. Simonis, « Global Constraint Catalog, Volume II, Time-Series Constraints », `https://arxiv.org/abs/1609.08925`, 2018.

[4] E. Arafailova, N. Beldiceanu, R. Douence, P. Flener, M. A. Francisco Rodríguez, J. Pearson & H. Simonis, « Time-Series Constraints: Improvements and Application in CP and MIP Contexts », in *International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'16)* (Banff, Canada) (C.-G. Quimper, éd.), LNCS, vol. 9676, Springer-Verlag, 2016, p. 18-34.

[5] E. Arafailova, N. Beldiceanu & H. Simonis, « Generating Linear Invariants for a Conjunction of Automata Constraints », in *Principles and Practice of Constraint Programming (CP'2017)* (Melbourne, Australia) (C. Beck, éd.), LNCS, vol. 10416, Springer-Verlag, 2017, p. 21-37.

[6] ———, « Deriving Generic Bounds for Time-Series Constraints Based on Regular Expressions Characteristics », *Constraints An Int. J.* **23** (2018), nº 1, p. 44-86.

[7] ———, « Invariants for time-series constraints », *Constraints An Int. J.* **25** (2020), nº 3-4, p. 71-120.

[8] N. Beldiceanu, « An example of introduction of global constraints in CHIP: Application to block theory problems », Tech. Report TR-LP-49, ECRC, May 1990.

[9] ———, « Global Constraints as Graph Properties on a Structured Network of Elementary Constraints of the Same Type », in *Principles and Practice of Constraint Programming (CP'2000)* (R. Dechter, éd.), LNCS, vol. 1894, Springer-Verlag, 2000, Preprint available as SICS Tech Report T2000-01, `soda.swedish-ict.se/2284/2/SICS-T--2000-01--SE.pdf`, p. 52-66.

[10] N. Beldiceanu & M. Carlsson, « Revisiting the Lexicographic Ordering Constraint », Tech. Report T2002-17, Swedish Institute of Computer Science, 2002, Available at `http://www.diva-portal.org/smash/get/diva2:1041533/FULLTEXT01.pdf`.

[11] N. Beldiceanu, M. Carlsson, R. Debruyne & T. Petit, « Reformulation of Global Constraints Based on Constraint Checkers », *Constraints An Int. J.* **10** (2005), nº 4, p. 339-362.

[12] N. Beldiceanu, M. Carlsson, R. Douence & H. Simonis, « Using finite transducers for describing and synthesising structural time-series constraints », *Constraints An Int. J.* **21** (2016), nº 1, p. 22-40.

[13] N. Beldiceanu, M. Carlsson & J.-X. Rampon, « Global Constraint Catalog, 2nd Edition (revision a) », Tech. Report T2012-03, Swedish Institute of Computer Science, 2012, Available at `http://soda.swedish-ict.se/5195/1/T2012-03.pdf`.

[14] N. Beldiceanu & T. Petit, « Cost Evaluation of Soft Global Constraints », in *International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'2004)* (Nice, France) (J.-C. Régin & M. Rueher, éds.), LNCS, vol. 3011, Springer-Verlag, April 2004, p. 80-95.

[15] N. Beldiceanu & H. Simonis, « A Constraint Seeker: Finding and Ranking Global Constraints from Examples », in *Principles and Practice of Constraint Programming (CP'2011)* (Perugia, Italy) (J. H. Lee, éd.), LNCS, vol. 6876, Springer-Verlag, 2011, p. 12-25.

[16] ———, « A Model Seeker: Extracting Global Constraint Models from Positive Examples », in *Principles and Practice of Constraint Programming (CP'2012)* (Quebec City, Canada) (M. Milano, éd.), LNCS, vol. 7514, Springer-Verlag, 2012, p. 141-157.

[17] C. Bessière, G. Katsirelos, N. Narodytska, C.-G. Quimper & T. Walsh, « Decompositions of *all different*, *global cardinality* and related constraints », in *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009* (C. Boutilier, éd.), 2009, p. 419-424.

[18] N. Bleuzen-Guernalec & A. Colmerauer, « Optimal Narrowing of a Block of Sortings in Optimal Time », *Constraints An Int. J.* **5** (2000), nº 1/2, p. 85-118.

[19] M. Carlsson & N. Beldiceanu, « From Constraints to Finite Automata to Filtering Algorithms », in *Programming Languages and Systems, 13th European Symposium on Programming, ESOP 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, March 29 - April 2, 2004, Proceedings*, 2004, p. 94-108.

[20] M. Carlsson, N. Beldiceanu & J. Martin, « A Geometric Constraint over k-Dimensional Objects and Shapes Subject to Business Rules », in *Principles and Practice of Constraint Programming (CP'2008)* (P. J. Stuckey, éd.), LNCS, vol. 5202, Springer-Verlag, 2008, p. 220-234.

[21] P. Codognet & D. Diaz, « A Simple and Efficient Boolean Solver for Constraint Logic Programming », *J. Autom. Reason.* **17** (1996), nº 1, p. 97-129.

[22] M.-C. Costa, « Persistency in maximum cardinality bipartite matchings », *Operation Research Letters* **15** (1994), p. 143-149.

[23] M.-C. Coté, B. Gendron & L.-M. Rousseau, « Modeling the Regular Constraint with Integer Programming », in *International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'2007)* (Brussels, Belgium) (P. Van Hentenryck & L. Wolsey, éds.), LNCS, vol. 4150, Springer-Verlag, May 2007, p. 29-43.

[24] M. Fischetti, L. Liberti, D. Salvagnin & T. Walsh, « Orbital shrinking: Theory and applications », *Discret. Appl. Math.* **222** (2017), p. 109-123.

[25] M. A. Francisco Rodríguez, P. Flener & J. Pearson, « Implied constraints for automaton constraints », in *GCAI 2015*, EasyChair Epic Series in Computing, 2015, Available at `http://www.easychair.org/publications/paper/Implied_Constraints_for_Automaton_Constraints`.

[26] M. A. Francisco Rodríguez, P. Flener & J. Pearson, « Automatic Generation of Descriptions of Time-Series Constraints », in *29th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2017, Boston, MA, USA, November 6-8, 2017*, IEEE Computer Society, 2017, p. 102-109.

[27] A. M. Frisch, B. Hnich, Z. Kiziltan, I. Miguel & T. Walsh, « Global Constraints for Lexicographic Orderings », in *Principles and Practice of Constraint Programming (CP'2002)* (P. Van Hentenryck, éd.), LNCS, vol. 2470, Springer-Verlag, 2002, p. 93-108.

[28] W.-J. van Hoeve & I. Katriel, « Global Constraints », in *Handbook of Constraint Programming* (F. Rossi, P. van Beek & T. Walsh, éds.), Elsevier, 2006, p. 169-208.

[29] D. Hofstadter & E. Sander, *L'Analogie, coeur de la pensée*, Odile Jacob, 2013.

[30] A. Lahrichi, « Ordonnancements: la notion de partie obligatoire et son application aux problèmes cumulatifs », Thèse, Paris 6 University, France, Mai 1979.

[31] C. E. Larson & N. Van Cleemput, « Automated conjecturing I: Fajtlowicz's Dalmatian heuristic revisited », *Artif. Intell.* **231** (2016), p. 17-38.

[32] J.-L. Laurière, « A Language and a Program for Stating and Solving Combinatorial Problems », *Artificial Intelligence* **10** (1978), nº 1, p. 29-127.

[33] J. Laurière, « Toward Efficiency Through Generality », in *Proceedings of the Sixth International Joint Conference on Artificial Intelligence, IJCAI 79, Tokyo, Japan, August 20-23, 1979, 2 Volumes* (B. G. Buchanan, éd.), William Kaufmann, 1979, p. 519-521.

[34] M. J. Maher, « Open Contractible Global Constraints », in *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009* (C. Boutilier, éd.), 2009, p. 578-583.

[35] N. Nethercote, P. J. Stuckey, R. Becket, S. Brand, G. J. Duck & G. Tack, « MiniZinc: Towards a Standard CP Modelling Language », in *Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings* (C. Bessière, éd.), Lecture Notes in Computer Science, vol. 4741, Springer, 2007, p. 529-543.

[36] W. J. Older, G. M. Swinkels & M. H. van Emden, « Getting to the Real Problem: Experience with BNR Prolog in OR », in *3rd Int. Conf. on the Practical Application of Prolog (PAP'95)*, Alinmead Software Ltd., 1995, p. 465-478.

[37] G. Pesant, « A Regular Language Membership Constraint for Finite Sequences of Variables », in *Principles and Practice of Constraint Programming (CP'2004)* (M. G. Wallace, éd.), LNCS, vol. 3258, Springer-Verlag, 2004, p. 482-495.

[38] J. Pitrat, *Métaconnaissance, Futur de l'Intelligence Artificielle*, Hermès, 1993.

[39] ———, « AI Systems Are Dumb Because AI Researchers Are Too Clever », *ACM Comput. Surv.* **27** (1995), nº 3, p. 349-350.

[40] ———, *Penser autrement l'informatique*, Hermès, 2000.

[41] ———, « MALICE, notre collègue », in *Colloque Métaconnaissance de Berder*, September 2001, p. 4-19.

[42] ———, « A Step toward an Artificial Artificial Intelligence Scientist », Tech. report, Université Paris VI, France, 2008.

[43] ———, *Artificial Beings*, Wiley InterScience, April 2009.

[44] G. Raayoni, G. Pisha, Y. Manor, U. Mendlovic, D. Haviv, Y. Hadad & I. Kaminer, « The Ramanujan Machine: Automatically Generated Conjectures on Fundamental Constants », *Nature* **590** (2021), p. 67-73.

[45] J.-C. Régin, « A Filtering Algorithm for Constraints of Difference in CSP », in *12th National Conference on Artificial Intelligence (AAAI-94)*, 1994, p. 362-367.

[46] A. Schutt, T. Feydy, P. J. Stuckey & M. G. Wallace, « Explaining the cumulative propagator », *Constraints An Int. J.* **16** (2011), nº 3, p. 250-282.

[47] G. J. Stein, « Machine Learning & Robotics: My (biased) 2019 State of the Field », `http://cachestocaches.com/2019/12/my-state-of-the-field/`, Accessed: 2021-07-28.

[48] R. Sutton, « The Bitter Lesson », `http://incompleteideas.net/IncIdeas/BitterLesson.html`, Accessed: 2021-07-28.

[49] L. G. Valiant, « The complexity of computing the permanent », *Theoretical Computer Science* **8** (1979), p. 189-201.

[50] P. Van Hentenryck, « Scheduling and Packing in the Constraint Language cc(FD) », in *Intelligent Scheduling* (M. Zweben & M. Fox, éds.), Morgan Kaufmann Publishers, 1994.

[51] A. Zanarini & G. Pesant, « Solution Counting Algorithms for Constraint-Centered Search Heuristics », in *Principles and Practice of Constraint Programming (CP'2007)* (C. Bessière, éd.), LNCS, vol. 4741, Springer-Verlag, 2007, p. 743-757.

Abstract. — In this article, I show how Jacques Pitrat's teachings and work have influenced my career and have fed my thoughts and questions in the field of constraint programming. After a brief reminder of the key ideas I adopted, I show how I used these ideas throughout my career and indicate why these ideas are still relevant today.

Keywords. — Constraints, Metaknowledge, Constraint programming.

Manuscrit reçu le 29 mars 2021, révisé le 28 juillet 2021, accepté le 15 septembre 2021.