Jean-Yves Lucas

La prise en compte des symétries dans CAIA

Volume 3, nº 1-2 (2022), p. 167-177.

<http://roia.centre-mersenne.org/item?id=ROIA_2022__3_1-2_167_0>

Revue Ouverte d'Intelligence Artificielle
Volume 3, nº 1-2, 2022, 167-177

La prise en compte des symétries dans CAIA

Jean-Yves Lucas[a]

[a] EDF R&D, Département OSIRIS 7 boulevard Gaspard Monge 91120 Palaiseau, France
E-mail : jean-yves.lucas@edf.fr.

Résumé. — Le système d'Intelligence Artificielle CAIA, conçu par Jacques Pitrat, est un chercheur artificiel capable de résoudre des problèmes de satisfaction de contraintes. Dans cet article, nous décrivons comment CAIA détecte les symétries du problème à résoudre, et ajoute des contraintes afin de casser ces symétries et réduire ainsi significativement la taille de l'espace de recherche. Une anecdote finale évoque une rencontre entre Jacques Pitrat et Douglas Hofstadter.

Mots-clés. — Système CAIA, problème de satisfaction de contraintes, prise en compte des symétries.

1. Introduction

Au tournant de l'année 2000, j'ai reçu un mail de Jacques Pitrat dans lequel il me demandait de relire un article qu'il venait d'écrire sur la manière dont il avait implémenté le résolveur de problème ALICE [8], de Jean-Louis Laurière, avec son propre système, qui s'appelait MACISTE. Ce résolveur s'appelait déjà MALICE, me semble-t-il, pour Meta-ALICE. MALICE est devenu depuis une brique du système qui s'appelle CAIA (Chercheur Artificiel en Intelligence Artificielle). J'avais soutenu ma thèse [9] sous la direction de Jean-Louis Laurière en 1989, le principal résultat de celle-ci étant le développement d'un système, SIREN (Système Intelligent pour Résoudre des ENoncés), qui résolvait des problèmes combinatoires par programmation automatique, après une phase de propagation de contraintes. Dans SIREN, bien des idées étaient directement inspirées d'ALICE, et j'imagine que c'est pour cette raison que Jacques Pitrat avait pensé à moi pour cette relecture.
C'était un honneur, mais un honneur un peu redoutable, néanmoins, car bien qu'ayant soutenu ma thèse depuis une dizaine d'années, je me sentais encore un peu son étudiant, comme à l'époque où je suivais son enseignement dans le module « méta-connaissances » qu'il donnait en DEA IARFA (Intelligence Artificielle, Reconnaissance des Formes et Applications). Lors de ses cours, nous étions toujours impressionnés par la profondeur des thèmes qu'il abordait, autour des connaissances, des méta-connaissances (avec plusieurs niveaux méta...) et des systèmes réflexifs. Pour tout dire, je ne me voyais guère critiquer son papier, ni sur la forme, ni encore moins sur le fond. Et pourtant, une relecture se doit d'être exigeante, c'est bien le moins

que l'on doive à l'auteur. J'ai donc entamé la lecture de l'article en me demandant si je trouverais une observation un tant soit peu pertinente à faire... Mais il s'avère que je n'ai pas eu de remarque ni d'observation, seulement une question sur un point technique.
Les méthodes de prise en compte des symétries décrites dans l'article m'avaient parues vraiment nouvelles, et simples, et élégantes dans leur principe [(1)]. Afin de bien situer le contexte de ces travaux, je vais décrire brièvement en section 2 le système CAIA dans lequel ils s'intègrent. Ensuite je présenterai la prise en compte des symétries sur un exemple en section 3. Le lecteur voulant une description complète de CAIA, et du traitement des symétries pour la résolution de problèmes combinatoires peut se reporter à l'article plus récent que Jacques Pitrat a écrit sur le sujet [10] disponible sur son blog à l'adresse : `http://jacques.pitrat.pagesperso-orange.fr`.

2. CAIA, un chercheur artificiel en Intelligence Artificielle

Jacques Pitrat pensait qu'il était possible de construire un système d'IA qui répondrait à la définition de l'IA forte. Toutefois, il pensait qu'atteindre cet objectif était si difficile qu'il était probablement hors de portée de l'intelligence humaine. Par conséquent, la solution qu'il proposait est de réaliser un bootstrap de l'IA, c'est-à-dire de s'appuyer sur l'IA elle-même pour construire un système d'IA forte. Dans cette optique, un jalon intermédiaire de l'IA est donc de créer un chercheur artificiel en IA. En effet, si l'on arrive à atteindre ce but, ce chercheur artificiel sera à même de progresser seul, de se poser de nouveaux problèmes et d'apprendre à les résoudre, de développer de nouveaux systèmes d'IA, et en particulier un chercheur artificiel plus performant que lui-même. C'est pour avancer dans cette direction que Jacques Pitrat a conçu CAIA, dont le domaine est la résolution de problèmes de satisfaction de contraintes. CAIA est un système d'IA constitué d'une hiérarchie de 5 agents :

- MALICE, qui résout un problème qui lui est posé par l'agent du niveau immédiatement supérieur, MONITOR ;
- MONITOR, qui contrôle le comportement de l'agent MALICE : avance-t-il vers une solution ou tourne-t-il en rond ? Quelle variable instancier pour continuer la résolution ? La propagation de contraintes est-elle inutile et coûteuse ? etc.
- MANAGER, qui gère le comportement de MONITOR : allocation de ressources, proposition d'expériences et génération de nouveaux problèmes ;
- ADVISOR, qui teste le comportement de MANAGER : exécute-t-il des tâches intéressantes ?
- Et finalement ZEUS, appelé par un mécanisme d'interruption, qui vérifie le fonctionnement des 4 autres agents, et relance éventuellement MANAGER.

[(1)] Un certain nombre de travaux ont porté sur la prise en compte des symétries dans les problèmes combinatoires (par exemple [1, 2, 3, 4, 5, 6, 7, 11]), la grande majorité de ces travaux étant postérieure à l'année 2000.

3. Les symétries dans CAIA

Lorsque l'on résout un problème combinatoire, il est essentiel de détecter les symétries de ce problème. Cela permet bien entendu de réduire la combinatoire, en ajoutant des contraintes qui cassent ces symétries. Mais cela permet aussi de démontrer certaines propriétés du problème, avant même de le résoudre effectivement.

Dans CAIA, la résolution du problème est faite par MALICE, et ce, sous le contrôle de MONITOR. La découverte des symétries est faite par MANAGER. L'ajout des contraintes qui permettent de casser ces symétries est fait aussi par MANAGER.

3.1. Découvrir les symétries d'un problème

Je reprends ici un exemple présenté dans le papier mentionné ci-dessus, pour illustrer la manière dont il est traité par CAIA, et en particulier comment sont traitées les symétries. Il s'agit d'un problème de cube magique $3 \times 3 \times 3$. On considère un cube $3 \times 3 \times 3$ constitué de 27 petits cubes comme le montre la figure 3.1.

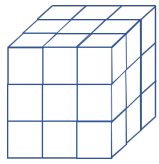

Figure 3.1. Un cube $3 \times 3 \times 3$

Le problème consiste à placer les 27 entiers consécutifs de 1 à 27 dans les 27 petits cubes de telle sorte que la somme des entiers placés dans tous les plans constitués de 9 petites cases soit toujours la même[(2)]. Pour faciliter la visualisation du cube, nous allons séparer dans les représentations graphiques de la suite, les 3 plans verticaux de 9 cases, afin de voir les 27 petits cubes. Le cube $3 \times 3 \times 3$ est alors représenté comme le montre la figure 3.2 avec ses 27 petits cubes numérotés de 1 à 27.

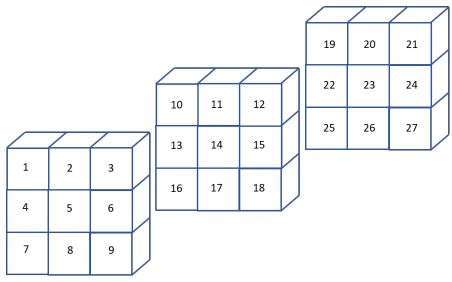

Figure 3.2. Les 27 petits cubes du cube $3 \times 3 \times 3$

[(2)]Notons que ce problème est différent de celui décrit à la page Wikipedia `https://fr.wikipedia.org/wiki/Cube_magique`. Il est soumis à moins de contraintes (15 seulement au lieu de 31), mais celles-ci contiennent plus de variables (9 au lieu de 3).

Les contraintes indiquent que tous les plans constitués de 9 petits cubes doivent avoir la même somme. Dans le langage de MALICE, directement inspiré du langage de description de problèmes ALICE, ce problème peut s'exprimer simplement comme suit :

```
SOIT CSTE N = 27
SOIT ENS E = [1, N]
TROUVER VAL, F BIJ E -> E
AVEC
F(1) + F(2) + F(3) + F(4) + F(5) + F(6) + F(7) + F(8) + F(9) = VAL
. . .
F(1) + F(5) + F(9) + F(10) + F(14) + F(18) + F(19) + F(23) + F(27) = VAL
```

La première ligne définit simplement la constante N égale à 27. La deuxième ligne introduit l'ensemble ENS constitué des entiers de 1 à 27. Cet ensemble va servir à modéliser l'ensemble des 27 petits cubes, c'est-à-dire l'ensemble des variables du problème, mais aussi l'ensemble des 27 valeurs possibles que l'on doit placer dans les petits cubes : c'est une petite astuce de modélisation qui permet de faire l'économie de la définition d'un second ensemble constitué lui aussi des entiers de 1 à 27. La troisième ligne indique le but de la recherche : il faut trouver la valeur qui sera affectée à une variable VAL dont le domaine n'est pas précisé (qui peut donc prendre n'importe quelle valeur située dans l'intervalle $]-\infty, +\infty[$).

Il faut aussi trouver une bijection de E dans E, c'est-à-dire une bijection de l'ensemble des 27 petits cubes dans l'ensemble des 27 valeurs possibles. Le fait de chercher une bijection indique bien que chacune des valeurs de 1 à 27 sera placée une et une seule fois dans l'un des 27 petits cubes. Enfin la section commençant par le mot-clé AVEC permet de décrire les 15 contraintes du problème, chaque contrainte correspond à l'un des 15 plans du grand cube constitués de 9 petits cubes.

Les figures 3.3, 3.4 et 3.5 donnent 3 exemples de plans représentés en bleu. La somme des valeurs affectées aux petits cubes de chacun de ces plans doit avoir la même valeur.

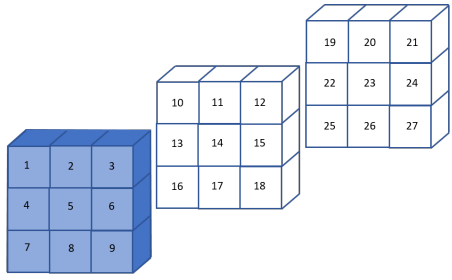

FIGURE 3.3. Le plan vertical antérieur du cube 3x3x3

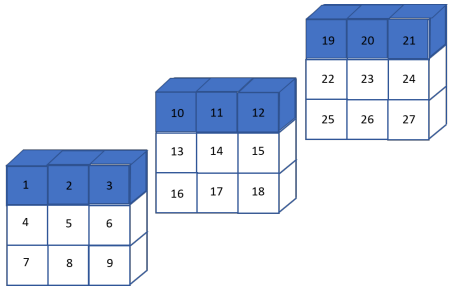

FIGURE 3.4. Le plan horizontal supérieur du cube 3x3x3

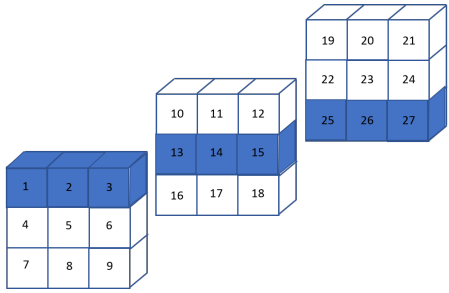

FIGURE 3.5. Un plan diagonal du cube 3x3x3

Avant même que MALICE et MONITOR commencent à résoudre le problème, l'agent MANAGER commence par chercher des symétries du problème. Ce qui me semble particulièrement élégant, c'est que cette recherche se fait par MANAGER en posant à MALICE un (méta-)problème combinatoire . Il s'exprime sous la forme générale suivante : trouver une bijection G, de l'ensemble des variables dans l'ensemble des variables, qui laisse les contraintes du problème inchangées[(3)]. Et naturellement, pour résoudre ce méta-problème, l'agent MALICE va utiliser, non pas une méthode d'énumération exhaustive, mais les mêmes connaissances de résolution que celles qu'il emploie pour résoudre n'importe quel problème combinatoire, et en particulier pour résoudre le problème de base qui lui est posé.

Cette manière de définir les symétries portant sur les variables du problème est à la fois simple et efficace, et elle permet de découvrir toutes les symétries géométriques du problème du cube 3×3×3. MALICE trouve 47 solutions au méta-problème, et identifie ainsi 47 symétries différentes. Certaines ne sont vraiment pas évidentes, comme celle correspondant à la bijection sur les variables donnée dans les tables 3.1 et 3.2 sous la forme de tableaux de 2 lignes, les éléments de la première ligne étant les antécédents et ceux de la seconde les images de la bijection.

[(3)] Dans son papier [10], Jacques Pitrat ne donne pas l'énoncé explicite de ce problème de recherche de symétries. Pour rester dans le cadre du langage de MALICE, on doit supposer l'existence de primitives du langage qui font référence au remplacement de certaines variables par d'autres variables dans une expression telle que le membre gauche ou droit d'une contrainte.

TABLE 3.1. Les 13 premières variables en correspondance dans la symétrie.

1	2	3	4	5	6	7	8	9	10	11	12	13
9	18	27	6	15	24	3	12	21	8	17	26	5

TABLE 3.2. Les 14 variables suivantes en correspondance dans la symétrie.

14	15	16	17	18	19	20	21	22	23	24	25	26	27
14	23	2	11	20	7	16	25	4	13	22	1	10	19

Dans cette bijection, les petits cubes rose clair ont pour image les petits cubes rouge brique (figure 3.6), les petits cubes rouge brique ont pour image les petits cubes verts (figure 3.7).

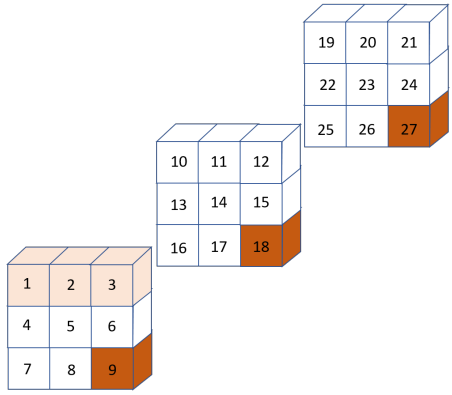

FIGURE 3.6. Des petits cubes en symétrie.

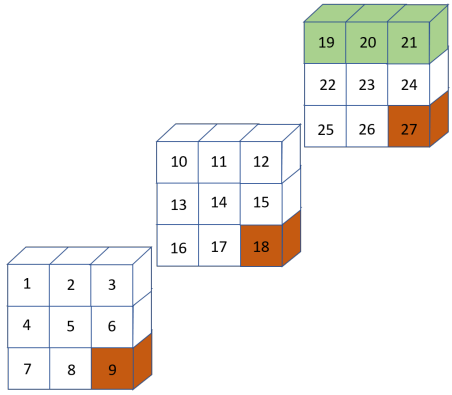

FIGURE 3.7. Des petits cubes en symétrie.

Pour la tranche horizontale du milieu, la situation est plus difficile à représenter. Disons pour simplifier qu'il s'agit d'une permutation d'un quart de tour des cubes de ce plan médian (le cube central est sa propre image). Ainsi par exemple les petits cubes marron ont pour image les petits cubes jaune clair (figure 3.8).

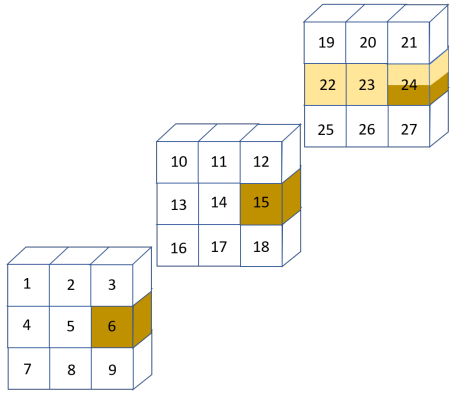

FIGURE 3.8. Des petits cubes en symétrie.

Détecter les symétries est véritablement difficile, car en réalité ce n'est pas une bijection des variables vers les variables qu'il faut construire, ce sont deux bijections : l'une des variables vers les variables, l'autre des contraintes vers les contraintes, cette dernière devant respecter la première bijection sur les variables. Illustrons cela par un exemple simple : Soit 6 variables V1, V2, V3, V4, V5 et V6 ayant toutes le même domaine de valeurs possibles, et soit les 3 contraintes à satisfaire :
(1) $V1 + V2 = V3$
(2) $V4 + V5 = V6$
(3) $V1 + V2 + V3 + V4 + V5 + V6 > 20$
Il existe une première symétrie évidente qui correspond à la bijection suivante sur les variables :

$$V1 \rightarrow V2, \quad V2 \rightarrow V1, \quad V4 \rightarrow V5, \quad V5 \rightarrow V4$$

$V3$ et $V6$ étant leur propre image. Ici la bijection sur les contraintes est la fonction identité : $(1) \rightarrow (1), (2) \rightarrow (2), (3) \rightarrow (3)$
Mais il existe une autre symétrie, qui correspond à la bijection suivante sur les variables :

$$V1 \rightarrow V4, \quad V4 \rightarrow V1, \quad V2 \rightarrow V5, \quad V5 \rightarrow V2, \quad V3 \rightarrow V6, \quad V6 \rightarrow V3$$

Ici la bijection sur les contraintes est : $(1) \rightarrow (2), (2) \rightarrow (1), (3) \rightarrow (3)$
Il existe d'ailleurs une troisième symétrie correspondant à la bijection :
$V1 \rightarrow V5, V5 \rightarrow V1, V2 \rightarrow V4, V4 \rightarrow V2, V3 \rightarrow V6, V6 \rightarrow V3$
avec la même bijection sur les contraintes que la symétrie précédente.
On voit donc qu'il faut au minimum implémenter un algorithme d'unification pour

analyser les contraintes dans le but d'identifier une bijection sur celles-ci après permutation des variables du problème. Les contraintes sont passées à MALICE par MANAGER[4]. MANAGER peut ainsi générer les solutions symétriques mais cela lui permet aussi, lorsqu'il trouve une nouvelle contrainte, de générer les contraintes symétriques de cette nouvelle contrainte, grâce aux bijections sur les variables.

3.2. Casser les symétries d'un problème

Il ne suffit pas de détecter les symétries d'un problème, encore faut-il casser ces symétries pour réduire l'espace de recherche. Dans le système CAIA, c'est fait en ajoutant des contraintes. Dans l'exemple de la symétrie du cube $3 \times 3 \times 3$ décrite ci-dessus, on voit que la variable $V1$ a pour image $V9$, $V2$ a pour image $V18$, etc. Il suffit donc pour casser la symétrie, d'ajouter la contrainte : $V1 < V9$ (ou $V2 < V18$, etc.) Dans ce cas, un seul ajout de contrainte suffit, car toutes les variables ont le même domaine de valeurs possibles. Néanmoins dans le cas général, la contrainte d'élimination est plus complexe. Dans le cas d'une bijection qui est une permutation symétrique sur 3 variables (U, V, W) (X, Y, Z), la contrainte s'exprime de la manière suivante :

$$(U < X) \vee (U = X \wedge V < Y) \vee (U = X \wedge V = Y \wedge W < Z) \vee (U = X \wedge V = Y \wedge W = Z)$$

Dans ce cas, l'agent MANAGER ajoute aussi la contrainte plus faible, mais plus facile à prendre en compte : $(U \leqslant X)$ puisque tous les termes du OU excluent la possibilité que U soit supérieure à X[5]. Il convient de remarquer qu'une seule contrainte peut casser plusieurs symétries, dans l'exemple du cube $3 \times 3 \times 3$, 14 contraintes suffisent pour casser les 47 symétries du problème.

3.3. Limites de la découverte de symétries par MANAGER

MANAGER ne trouve pas toutes les symétries d'un problème. Par exemple dans le problème du cube $3 \times 3 \times 3$, il existe une symétrie non pas géométrique comme celles portant sur des permutations de variables, mais arithmétique, c'est-à-dire portant sur les valeurs : il suffit, pour chaque variable, de remplacer dans chaque solution, la valeur V de cette variable par la valeur $28 - V$. MANAGER ne la trouve pas, en particulier car il faut connaître la valeur qui sera affectée à la variable VAL, c'est-à-dire 126 (c'est la seule valeur possible de VAL), afin de montrer qu'il existe une bijection sur les contraintes, en l'espèce que chaque contrainte reste inchangée si on remplace chaque variable V par l'expression $28 - V$. En effet, la contrainte :

$$V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8 + V9 = \text{VAL}$$

devient :

$$V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8 + V9 = 126$$

[4] Jacques Pitrat mentionne dans son article que lorsque l'agent MANAGER demande un travail à l'agent MALICE, il le fait toujours via l'agent MONITOR.

[5] Dans le cadre de la programmation par contraintes, cette contrainte peut s'exprimer comme une contrainte d'ordre lexicographique entre deux vecteurs de variables.

En effectuant les remplacements V par (28-V), on obtient :

$$(28 - V1) + (28 - V2) + (28 - V3) + (28 - V4) + (28 - V5) + ((28 - V6) + (28 - V7) + (28 - V8) + (28 - V9) = 126$$

soit :

$$9 \times 28 - 126 = V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8 + V9$$
$$252 - 126 = V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8 + V9$$

La contrainte est bien inchangée. D'autres symétries locales dans certains problèmes ne sont pas détectées non plus. Comme Jacques Pitrat l'écrit dans le papier cité au début de cet article [10], trouver toutes les symétries d'un problème est un méta-problème très difficile !

4. Pour conclure...

Pour autant, les idées que Jacques Pitrat a mises en œuvre il y a plus de vingt ans pour caractériser et prendre en compte les symétries d'un problème combinatoire étaient, et sont encore, à bien des égards, en avance sur d'autres travaux relatifs au même domaine. Cela ne surprendra pas ceux qui ont eu le plaisir de le connaître et d'apprécier son enseignement et ses travaux de recherche.

5. Post scriptum

J'aimerais ajouter une petite anecdote qui me fait sourire chaque fois que j'y repense. Quand j'étais doctorant, Jacques Pitrat et Jean-Louis Laurière avaient l'habitude d'organiser une réunion mensuelle à Jussieu où les membres de leurs équipes et tous leurs thésards étaient conviés (nous étions donc habituellement une bonne quinzaine de personnes). Trois doctorants, à tour de rôle, présentaient l'état d'avancement de leur thèse, les difficultés rencontrées, ce qui marchait bien ou moins bien, etc. Avant ces présentations, Jacques Pitrat nous faisait souvent une revue rapide des articles qu'il avait lus, des travaux dont il avait entendu parler, des personnes qu'il avait rencontrées. Et il se trouve qu'à l'une de ces occasions, il nous a dit qu'il avait rencontré Douglas Hofstadter qui était en visite en France, et outre les discussions qu'il avait eues avec lui concernant l'Intelligence Artificielle, il nous a raconté l'anecdote suivante : au cours d'un repas au restaurant, Douglas Hofstadter avait pris une feuille de papier et avait écrit à main levée le nom de M. Pitrat, mais ce qui était surprenant, c'est qu'en tournant la feuille de 180°, on y lisait en fait son prénom, Jacques ! C'était bien dans la manière d'Hofstadter, tous ceux qui ont lu « Gödel, Escher et Bach, les Brins d'une Guirlande Eternelle » se souviennent comme il aimait jouer avec les notions de fond et de forme, d'envers et d'endroit, en particulier en commentant les dessins d'Escher. Cette anecdote m'avait fortement impressionné. J'aime dessiner, et je ne voyais pas trop comment il était possible d'écrire un nom dans un sens qui devient un prénom dans l'autre sens, à l'envers ! En bon doctorant de Jean-Louis Laurière, j'ai bien sûr immédiatement « propagé les contraintes » c'est-à-dire que j'ai compté le nombre de

lettres. Le nom Pitrat compte six lettres, le prénom Jacques en compte sept, ce qui n'a fait qu'augmenter ma perplexité. Comment un nom de six lettres peut-il se transformer en prénom de 7 lettres lorsqu'il est lu à l'envers ? Bien entendu, certaines lettres lues à l'envers forment aussi des lettres, un b à l'envers forme un q, un d à l'envers est un p, un o reste un o, etc. Mais s'il y a bien un p dans Pitrat, il n'y a pas de d dans Jacques... Bref, cette énigme m'a titillé. Le premier exposé a commencé, je ne me souviens plus qui présentait, mais qu'il veuille bien accepter rétrospectivement mes excuses, j'avoue ne pas avoir écouté sa présentation. J'ai été ce jour-là un étudiant dissipé, j'ai passé une heure à chercher comment écrire « Jacques » à l'endroit qui se lise « Pitrat » à l'envers, et vice-versa. Cela m'a pris pratiquement toute la durée de l'exposé, et j'ai finalement abouti à la solution présentée sur les figures 5.1 et 5.2.

Figure 5.1. Le nom

Figure 5.2. Le prénom

Ce n'était pas parfait, et j'imagine bien que la solution d'Hofstadter était plus élégante (je n'ai jamais eu l'occasion de la voir, hélas), mais je n'étais tout de même pas mécontent d'être arrivé à quelque chose. Après chaque exposé, il y avait une pause d'une dizaine de minutes. J'en ai profité pour montrer aux doctorants assis à côté de moi ce à quoi j'avais abouti, et tous m'ont dit : « il faut absolument montrer ça à M. Pitrat ! » A dire vrai, je n'en avais vraiment pas envie : l'idée que, peut-être, M. Pitrat allait me regarder avec un air déçu et me dirait « mais vous griffonnez au lieu d'écouter les exposés ? » n'était pas faite pour me rassurer... Mais bon, un peu poussé par mes petits camarades, me voilà à côté de Jacques Pitrat qui discutait avec quelques personnes. Quelqu'un dit : « M. Pitrat, regardez ce que Jean-Yves a dessiné... ». Il se retourne, je lui tends ma feuille de papier... Je ne savais pas trop comment il allait réagir, je me disais que peut-être, il parlerait du dessin d'Hofstadter, ou que nous disserterions sur la possibilité ou de la nécessité pour une machine intelligente de créer des dessins, ou d'aboutir aussi à une solution à ce petit problème, ou même d'avoir l'idée de se lancer un tel défi, écrire à l'endroit et à l'envers... Et bien pas du tout ! Jacques Pitrat a pris ma feuille, l'a observée quelques secondes à l'endroit, puis à l'envers, puis à nouveau à l'endroit, puis il me l'a rendue en me disant « Ah oui, ah bien, c'est facile alors... » et il s'est retourné et a repris sa conversation. Je crois que cela ne l'intéressait plus. Imaginez ma déconfiture. Non je n'avais pas trouvé ça facile ! Cela m'avait pris une heure ! Je pense que M. Pitrat, me faisant trop crédit au fond, avait cru que j'avais fait ce dessin en quelques secondes, à la fin de l'exposé, à main levée, comme Douglas Hofstadter !

J'ai eu la chance de suivre les cours de Jean-Louis Laurière et de Jacques Pitrat, et de profiter pendant ma thèse de leurs conseils, remarques, commentaires... Ils m'ont appris beaucoup, bien sûr, mais, au-delà de leur enseignement, chacun d'eux m'a inculqué une idée forte. De Jean-Louis Laurière, j'ai appris que si une idée est bonne, alors il faut s'y accrocher, ne pas transiger avec, et si elle est vraiment bonne, elle tiendra bon. Ce jour-là, Jacques Pitrat m'a appris un autre beau principe : il ne faut s'intéresser qu'à ce qui est nouveau, et difficile.

Bibliographie

[1] D. Cohen, P. Jeavons, C. Jefferson & K. Petrie, « Constraint Symmetry and Solution Symmetry », in *AAAI 2006 - Proceedings of the 28th National Conference on Artificial Intelligence, 16-20 July 2006, Boston, Massachusetts, USA*, 2006.

[2] D. Cohen, P. Jeavons, C. Jefferson, K. E. Petrie & B. M. Smith, « Symmetry Definitions for Constraint Satisfaction Problems », in *Principles and Practice of Constraint Programming – CP 2005* (P. van Beck, éd.), Springer, 2005, p. 17-31.

[3] P. Flener, « Topic 5: Symmetry », in *Combinatorial Optimisation and Constraint Programming (Course 1DL441)*, Université d'Uppsala, Suède, 2020, `https://user.it.uu.se/~pierref/courses/COCP/slides/T05-Symmetry.pdf`.

[4] E. C. Freuder, « Eliminating Interchangeable Values in Constraint Satisfaction Problems », in *AAAI 1991 – Proceedings of the 9th National Conference on Artificial Intelligence, 14-19 July 1991, Anaheim, California, USA*, 1991, p. 227-233.

[5] H. Gelernter, « A Note on Syntactic Symmetry and the Manipulation of Formal Systems by Machine », *Information and control* **2** (1959), p. 80-89.

[6] I. P. Gent, W. Harvey & T. Kelsey, « Groups and Constraints: Symmetry Breaking during Search », in *Principles and Practice of Constraint Programming – CP 2002* (P. van Hentenryck, éd.), Springer, 2002, p. 415-430.

[7] I. P. Gent, K. Petrie & J.-F. Puget, « Symmetry in constraint programming », in *Handbook of constraint programming – Chapter 10*, Elsevier, 2006.

[8] J.-L. Laurière, « Un langage et un programme pour énoncer et résoudre des problèmes combinatoires », Thèse, Université Paris 6, 1976.

[9] J.-Y. Lucas, « Génération automatique de programmes par règles et compilation de base de règles : application à un système expert de diagnostic de signaux courants de Foucault », Thèse, Université Pierre et Marie Curie, 1989.

[10] J. Pitrat, « A Step toward An Artificial Artificial Intelligence Scientist », `http://jacques.pitrat.pagesperso-orange.fr`, 2008.

[11] J.-F. Puget, « Breaking Symmetries in All Different Problems », in *Proceedings of IJCAI'05*, 2005, p. 272-277.

Abstract. — CAIA is an AI system designed by Jacques Pitrat. It is an artificial researcher capable of solving constraint satisfaction problems. In this paper, we describe how CAIA detects the symmetries of the problem and adds constraints in order to break these symmetries. As a result, the size of the search space is significantly reduced. A final anecdote evokes a meeting between Jacques Pitrat and Douglas Hofstadter.

Keywords. — CAIA System, Constraint Satisfaction Problem, Symmetry Handling.

Manuscrit reçu le 30 mars 2021, révisé le 10 novembre 2021, accepté le 30 novembre 2021.