Apprentissage de Contraintes

Rémi Coletta, Christian Bessière et Joel Quinqueton

LIRMM-CNRS
161 rue Ada
34392 Montpellier Cedex 5
{coletta,bessiere,jq}@lirmm.fr

Résumé : La programmation par contraintes est une technologie désormais largement utilisée pour résoudre des problèmes combinatoires dans les applications industrielles. Pourtant, l'utiliser requiert une certaine connaissance du paradigme des contraintes. Cet article introduit un cadre pour apprendre automatiquement des réseaux de contraintes à partir d'ensembles d'instances qui sont des solutions acceptables ou des assignations non désirables du problème que nous souhaiterions exprimer. Ce qui peut aider un novice à manipuler ses contraintes. En restreignant le langage des contraintes utilisées pour construire le réseau, cela peut aussi assister un expert dans la recherche d'une modélisation efficace d'un problème donné.

Mots-clés : Programmation par contraintes, Apprentissage automatique.

1 Introduction

Depuis les 30 dernières années, des progrès considérables ont été réalisés en Programmation par contraintes, fournissant ainsi un paradigme efficace pour résoudre des problèmes combinatoires. Son usage s'est répandu dans de nombreux domaines d'application, tels que l'allocation de ressources, l'ordonnancement, etc (Wallace, 1996).

Néanmoins, l'utilisation de ce paradigme reste limitée aux spécialistes de la technologie des contraintes. Modéliser un problème dans le formalisme des contraintes requiert certaines notions de satisfaction de contraintes. Ce qui exclut à un novice la possibilité d'utiliser la programmation par contraintes tout seul, sans l'aide d'un expert. En effet, les utilisateurs savent reconnaître les exemples dans lesquels leurs contraintes sont satisfaites ou violées, mais ils ne savent pas manipuler les contraintes proprement dites. Notre premier but est de soulager l'expert, en fournissant des méthodes semi-automatiques pour acquérir les contraintes de l'utilisateur.

Par ailleurs, même si l'utilisateur réussit à coder son problème dans le formalisme des contraintes, une mauvaise modélisation peut ruiner les performances du meilleur solveur basé sur les meilleurs techniques de filtrage. Par exemple, dans le but de fournir une assistance à la modélisation, certains solveurs (ILOG Solver, Choco) intègrent la possibilité de décrire des contraintes en extension (c.à.d. en énumérant les n-uplets autorisés). Ceci étend considérablement les possibilités de modélisation et facilite la définition de combinaisons complexes entre des variables. Mais la sémantique des contraintes ainsi exprimées est complètement perdue et ne peut donc pas être exploitée pour améliorer leur propagation. Il en résulte une détérioration significative des performances de résolution si de telles contraintes sont trop largement utilisées. Notre second but est donc de permettre à l'expert de définir son propre 'biais' (c.à.d. une limitation des contraintes pouvant être utilisées pour coder le problème ou un sous problème particulièrement compliqué) et de tester la représentabilité du problème dans ce biais. L'expert serait alors capable, par exemple, de tester si une bibliothèque de contraintes optimisées d'un solveur est suffisante pour exprimer de manière efficace un problème actuellement codé en utilisant des contraintes génériques et moins efficaces.

De nombreux travaux ont été réalisés en programmation par contraintes pour identifier des classes polynomiales de réseaux de contraintes (Freuder, 1982; van Beek & Dechter, 1995). Malheureusement, d'une part ces classes polynomiales sont pauvres en terme d'expressivité et ne codent que peu de problèmes réels. D'autre part, ces classes polynomiales ont des propriétés statiques, et ne peuvent pas s'adapter au type de problèmes que l'utilisateur souhaite résoudre : c'est le problème qui doit s'adapter à la classe polynomiale, et non le contraire. L'approche proposée ici se veut plus flexible. Elle ne définit pas de classes a priori dans lesquelles l'expressivité d'un problème doit être testée, mais laisse le soin à l'expert de les définir lui-même. (Nous pouvons imaginer une tactique interactive où l'utilisateur essaye des bibliothèques de contraintes de plus en plus expressives jusqu'à réussir à exprimer son problème.) Précisons que le choix du biais sort du cadre de ce papier. Nous pourrions utiliser une approche basée sur les cas, sélectionnant des bibliothèques en fonction du type de problème (scheduling, time-tabling, etc.) (Little *et al.*, 2002).

Des travaux existants ont déjà essayé d'utiliser des techniques issues d'apprentissage automatique pour aider l'utilisateur à modéliser son problème. Dans (Rossi & Sperduti, 1998), le but n'est pas exactement d'aider l'utilisateur à apprendre un réseau de contraintes, mais de l'aider à apprendre les valuations des n-uplets dans un semi-ring CSP dont les contraintes sont déjà données. Dans (O'Connell *et al.*, 2002), le but général est très proche du nôtre mais les techniques présentées se concentrent sur l'acquisition d'une seule contrainte et mettent l'accent sur les heuristiques pour améliorer le processus d'apprentissage de cette contrainte. Inversement, le but de notre article est principalement de poser le problème général de l'acquisition d'un réseau de contraintes, de le formaliser en termes d'apprentissage automatique, et de circonscrire quelques unes des difficultés que cette approche soulève.

Le reste de cet article est organisé comme suit. La Section 2 donne quelques définitions préliminaires sur les réseaux de contraintes. La Section 3 présente brièvement les techniques d'apprentissage automatique qui vont être utilisées pour notre problème. Dans la Section 4, nous formulons notre problème comme un problème d'apprentissage. La Section 5 présente la technique en détails et donne ses bonnes propriétés. Dans la Section 6, quelques difficultés que cette approche soulève sont présentées et leurs effets possibles sur le processus d'apprentissage sont illustrés au travers de quelques expérimentations. La Section 7 conclut le travail et élargit sur les perspectives.

2 Préliminaires

Définition 1 (Réseau de Contraintes) *On définit un réseau de contraintes comme un triplet* $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ *où :*
- $\mathcal{X} = \{X_1, \ldots, X_n\}$ *est un ensemble de variables.*
- $\mathcal{D} = \{D_{X_1}, \ldots, D_{X_n}\}$ *est l'ensemble de leur domaine : chaque variable* X_i *prend sa valeur dans le domaine* D_{X_i}.
- $\mathcal{C} = (C_1, \ldots, C_m)$ *est une séquence de contraintes sur* $\mathcal{X}$ *et* $\mathcal{D}$*, où une contrainte* C_i *est définie par la séquence* $var(C_i)$ *des variables sur lesquelles elle porte, et la relation* $rel(C_i)$ *qui spécifie les n-uplets autorisés sur* $var(C_i)$.

L'ensemble des contraintes est défini comme une séquence dans le seul but de simplifier les notations à venir.

Définition 2 (Instance) *Soit* $Y = \{Y_1, \cdots, Y_k\}$ *un sous-ensemble de* $\mathcal{X}$. *Une* instance e_Y *sur* Y *est un n-uplet* $(v_1, \ldots, v_k) \in D_{Y_1} \times \ldots \times D_{Y_k}$. *Cette instance est partielle si* $Y \neq \mathcal{X}$*, complète sinon (notée* e*). Une instance* e_Y *sur* Y viole *la contrainte* C_i *ssi* $var(C_i) \subseteq Y$ *et* $e_Y[var(C_i)] \notin rel(C_i)$.

Définition 3 (Solution) *Une instance complète sur un ensemble* $\mathcal{X}$ *de variables est une* solution *du réseau de contraintes* $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ *ssi elle ne viole aucune contrainte. Sinon, c'est une* non solution. *On note* $Sol(\mathcal{X}, \mathcal{D}, \mathcal{C})$ *l'ensemble des solutions de* $(\mathcal{X}, \mathcal{D}, \mathcal{C})$.

3 Problème de base

Comme point de départ, nous supposons que l'utilisateur connaît les variables et leur domaine de valeurs possibles. Il est aussi supposé capable de se prononcer sur la validité d'une instance. Ainsi, les données disponibles sont l'ensemble $\mathcal{X}$ des variables du problème, leur domaine $\mathcal{D}$, un sous-ensemble E^+ des solutions du problème et un ensemble E^- de non solutions.

De plus, du point de vue de l'expert, le but est de coder le problème efficacement, en utilisant seulement des contraintes efficaces entre ces variables (c.à.d. une bibliothèque de contraintes avec des caractéristiques de propagation efficaces.) Des indications peuvent aussi être fournies à propos de la localisation des contraintes, en définissant les variables entre lesquelles des contraintes doivent être cherchées (apprises) ou en restreignant l'arité de ces contraintes. Ces limitations sémantiques et structurelles définissent le biais inductif :

Définition 4 (Biais) *Étant donnés un ensemble* $\mathcal{X}$ *de variables et l'ensemble* $\mathcal{D}$ *de leur domaine, un* biais $\mathcal{B}$ *sur* $(\mathcal{X}, \mathcal{D})$ *est une séquence* $(B_1, \ldots, B_m)$ *de biais locaux, où un biais local* B_i *est défini par une séquence* $var(B_i) \subseteq \mathcal{X}$ *de variables, et un ensemble* $L(B_i)$ *de relations possibles sur* $var(B_i)$.

L'ensemble $L(B_i)$ des relations autorisées sur un ensemble de variables $var(B_i)$ peut être n'importe quelle bibliothèque de contraintes d'arité $|var(B_i)|$.

Définition 5 (Appartenance à un Biais) *Étant donnés un ensemble $\mathcal{X}$ de variables et l'ensemble $\mathcal{D}$ de leur domaines, une séquence de contraintes $\mathcal{C} = (C_1, \ldots, C_m)$* appartient *à un biais $\mathcal{B} = (B_1, \ldots, B_m)$ sur $(\mathcal{X}, \mathcal{D})$ si $\forall C_i \in \mathcal{C}, var(C_i) = var(B_i)$ et $rel(C_i) \in L(B_i)$. Nous notons $\mathcal{C} \in \mathcal{B}$.*

Le problème consiste en la recherche d'une séquence de contraintes $\mathcal{C}$ appartenant à un biais donné $\mathcal{B}$, et dont l'ensemble des solutions est un sur-ensemble de E^+ ne contenant aucun élément de E^-.

Définition 6 (Problème d'Acquisition de Contraintes) *Étant donnés un ensemble de variables $\mathcal{X}$, leur domaine $\mathcal{D}$, deux ensembles E^+ et E^- d'instances sur $\mathcal{X}$, et un biais $\mathcal{B}$ sur $(\mathcal{X}, \mathcal{D})$, le* problème d'acquisition de contraintes *consiste à trouver une séquence de contraintes $\mathcal{C}$ telle que :*

$\mathcal{C} \in \mathcal{B}$,

$\forall e^- \in E^-$, e^- n'est pas solution de $(\mathcal{X}, \mathcal{D}, \mathcal{C})$, et,

$\forall e^+ \in E^+$, e^+ est solution de $(\mathcal{X}, \mathcal{D}, \mathcal{C})$.

Si (E^+, E^-), appelées les *données d'entraînement*, sont fournies par une interaction avec l'utilisateur, alors le problème d'acquisition de contraintes est appelé phase de modélisation du problème de l'utilisateur, sinon, c'est un test automatique de reformulation.

Exemple 1 Signalons que si $E^+ \cup E^- = D_{X_1} \times \cdots \times D_{X_n}$, et $\mathcal{B}$ est un biais sur $(\mathcal{X}, \mathcal{D})$ contenant $n(n-1)/2$ biais locaux tels que pour chaque paire de variables (X_i, X_j), $\exists B_i \in \mathcal{B}$ avec $var(B_i) = (X_i, X_j)$, et $L(B_i) = \mathcal{P}(D_{X_i} \times D_{X_j})$,[1] alors le problème de l'acquisition de contraintes répond au problème de la représentabilité d'une relation $\rho = E^+$ dans un réseau de contraintes binaires (Montanari, 1974).

4 Quelles techniques en Apprentissage Automatique ?

L'induction de concepts est un paradigme bien connu en apprentissage automatique. Le problème sous-jacent peut se formuler de la manière suivante : étant donnés un ensemble H d'hypothèses, deux ensembles de données (E^+ d'instances positives et E^- d'instances négatives), trouver une hypothèse h consistante avec ces données d'entraînement, c.à.d. qui rejette toutes les instances négatives et accepte toutes les positives. Le concept fournissant les données d'entraînement est appelé le concept cible.

Dans notre cadre, ce concept cible est le réseau de contraintes inconnu que nous cherchons (un qui soit consistant avec toutes les informations données par l'utilisateur dans l'ensemble d'entraînement). Donc, dans notre vocabulaire :

- Une hypothèse h est une séquence de contraintes,
- H est l'ensemble des séquences de contraintes possibles appartenant à $\mathcal{B}$,
- Le concept cible est la séquence de contraintes $\mathcal{C}$ que nous cherchons,

[1] E étant un ensemble, $\mathcal{P}(E)$ est l'ensemble des sous-ensembles de E.

– Une instance positive est une solution de $(\mathcal{X}, \mathcal{D}, \mathcal{C})$, une instance négative est une non solution de $(\mathcal{X}, \mathcal{D}, \mathcal{C})$.

A partir de maintenant, nous utiliserons indifféremment les termes issus du raisonnement par contraintes ou de l'apprentissage automatique pour faire référence à ces notions.

Il existe une multitude de techniques en Apprentissage Automatique, depuis les arbres de décision jusqu'aux réseaux de neurones en passant par les algorithmes génétiques. Nous proposons ici une méthode basée sur l'Espace des Versions (Mitchell, 1997), qui présente un certain nombre de bonnes propriétés dont les plus intéressantes dans notre perspective sont : il fournit deux approximations (une borne supérieure et une borne inférieure), son calcul est incrémental relativement aux données d'entraînement, et le résultat ne dépend pas de l'ordre des instances d'entraînement (commutativité). Cette dernière propriété est essentielle dans un processus interagissant avec l'utilisateur.

Nous allons présenter brièvement les Espaces des Versions. Ils sont basés sur un ordre partiel de l'ensemble H des hypothèses.

Définition 7 (Relation de Généralisation $\leq_G$) *Étant donné $(\mathcal{X}, \mathcal{D})$ un ensemble de variables et leur domaine, une hypothèse h_1 est moins générale que ou équivalente à une hypothèse h_2 (ce que l'on note $h_1 \leq_G h_2$) ssi l'ensemble des solutions de $(\mathcal{X}, \mathcal{D}, h_1)$ est un sous-ensemble de celui de $(\mathcal{X}, \mathcal{D}, h_2)$.*

Un espace des versions ne fournit pas seulement une hypothèse consistante, mais tout le sous-ensemble de H consistant avec les données d'entraînement :

Définition 8 (Espace des Versions) *Étant donné $(\mathcal{X}, \mathcal{D})$ un ensemble de variables et leur domaine, E^+ et E^- deux ensembles d'entraînement et H un ensemble d'hypothèses, l'*espace des versions *est l'ensemble :*

$$V = \{h \in H / E^+ \subseteq Sol(\mathcal{X}, \mathcal{D}, h), E^- \cap Sol(\mathcal{X}, \mathcal{D}, h) = \emptyset\}$$

Grâce à sa bonne propriété d'incrémentalité relativement aux données d'entraînement, un espace des versions est appris en traitant une par une les instances d'entraînement de E^+ et E^-. De plus, grâce à l'ordre partiel $\leq_G$, un espace des versions V est complètement caractérisé par deux bornes : la borne spécifique S qui est l'ensemble des éléments minimaux de V (selon $\leq_G$), et la borne générale G, l'ensemble de ses éléments maximaux.

Propriété 1 *Étant donné un espace des versions V, et ses bornes S et G, $\forall h \in V, \exists s \in S$ et $\exists g \in G / s \leq_G h \leq_G g$.*

Dans le cas général, $|V|$ est exponentiel en la taille de la donnée. Mais, grâce à la Proposition 1, le problème de l'acquisition de l'espace des versions se réduit strictement à calculer ses bornes S et G consistantes avec (E^+, E^-).

Étant donnés un ensemble d'hypothèses H sur $(\mathcal{X}, \mathcal{D})$ et les données d'entraînement (E^+, E^-), s'il n'existe pas d'hypothèses $h \in H$ consistante avec (E^+, E^-), alors l'acquisition de l'espace des versions va se finir dans un état dans lequel il existe $s \in S$ et $g \in G$ tels que $s \neq g$ et $g \leq_G s$. Cet état est appelé l' *effondrement* de l'espace des versions.

5 Apprendre un espace des versions de contraintes

Dans cette section, nous décrivons le processus d'apprentissage de l'espace des versions correspondant au problème d'acquisition de contraintes sur $(\mathcal{X}, \mathcal{D})$ avec les ensembles (E^+, E^-) de solutions et non solutions, et le biais $\mathcal{B}$ sur $(\mathcal{X}, \mathcal{D})$.

Définissons tout d'abord les notions et notations que nous allons utiliser au niveau d'une seule contrainte. Nous projetons la relation de généralisation $\leq_G$ au niveau d'une contrainte. Afin d'être en accord avec l'espace des versions nous définissons l'ensemble $L^H(B_i) = L(B_i) \cup \{\bot, \top\}$, où $\bot$ est la relation vide et $\top$ est la relation universelle. (Notons que l'on peut ajouter la relation universelle à n'importe quelle bibliothèque de contraintes, et ce sans perte de généralité.) Ainsi, $\leq_g$ est un ordre partiel sur $L^H(B_i)$ tel que $\forall r_1, r_2 \in L^H(B_i), r_1 \leq_g r_2 \Leftrightarrow r_1 \subseteq r_2$. Étant donnés $L_1 \subseteq L^H(B_i)$ et $L_2 \subseteq L^H(B_i)$, par abus de notation, on notera $L_1 \leq_g L_2$ ssi $\forall r_1 \in L_1, \forall r_2 \in L_2, r_1 \leq_g r_2$.

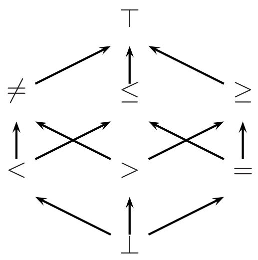


FIG. 1 – $L^H(B_i)$

Exemple 2 Soit $L(B_i) = \{<, \leq, =, \geq, >, \neq\}$ un biais local donné. La Fig. 1 décrit l'ordre partiel $(L^H(B_i), \leq_g)$, qui dans ce cas est un treillis.

En restant au niveau d'une seule contrainte, nous introduisons un espace des versions local à chaque contrainte (c.à.d. à chaque biais local B_i). Puisque $\leq_g$ est un ordre partiel tout comme $\leq_G$, l'espace des versions local hérite de la propriété 1, il est donc lui aussi complètement caractérisé par ses bornes spécifique et générale.

Définition 9 (Bornes locales) *$L(S_i)$ (resp. $L(G_i)$) désigne l'ensemble des relations de $L^H(B_i)$ qui appartiennent à une hypothèse de S (resp. G) :*

$$L(S_i) = \{r \in L^H(B_i) / \exists s \in S : (var(Bi), r) \in s\}$$

$$L(G_i) = \{r \in L^H(B_i) / \exists g \in G : (var(Bi), r) \in g\}$$

S_i et G_i désignent, quant à elles, des ensembles de contraintes :

$$S_i = \{(var(B_i), r),\ \textit{où}\ r \in L(S_i)\}; \qquad G_i = \{(var(B_i), r),\ \textit{où}\ r \in L(G_i)\}$$

5.1 Le processus d'apprentissage

Nous sommes désormais prêts pour décrire l'algorithme CONACQ qui prend en entrée un biais $\mathcal{B}$, deux ensembles d'entraînement E^+ et E^- et retourne l'espace des versions correspondant V. Nous présentons étape par étape les différents cas possibles quand une instance des données d'entraînement est traitée.

5.1.1 Instances de E^+

Une instance positive e^+ doit être une solution de tous les réseaux $(\mathcal{X}, \mathcal{D}, h)$ où $h \in V$. Ainsi,

$$\forall h \in V, \forall B_i \in h, e^+[var(B_i)] \in rel(B_i)$$

Si l'on se reporte à l'espace des versions de chaque biais local B_i, on obtient la propriété suivante :

Propriété 2 (Propriété de projection des S_i locaux) *Chaque borne spécifique locale S_i doit accepter toutes les instances positives. $L(S_i)$ est donc l'ensemble des relations minimales au sens de $\leq_g$ acceptant tout E^+ :*

$$L(S_i) = min_{\leq_g}\{r \in L^H(B_i)/\forall e^+ \in E^+, e^+[var(B_i)] \in r\}$$

Corollaire 1 *La borne spécifique globale S est le produit cartésien des bornes spécifiques locales, c.à.d. l'ensemble des hypothèses, où chaque contrainte prend sa valeur dans $L(Si)$:*

$$S = \bigtimes_{i \in 1..m} S_i$$

Grâce à la propriété 2, quand une instance positive e^+ est reçue, chaque biais local B_i peut être traité individuellement (ligne 2 de l'Algorithme CONACQ). Si la borne spécifique actuelle d'une contrainte accepte déjà cette instance positive, il n'y a rien à faire pour cette contrainte (ligne 3), sinon la borne remonte aux relations les plus spécifiques de l'espace des versions local (c.à.d. les relations de $L^H(B_i)$ comprise entre $L(S_i)$ et $L(G_i)$) acceptant e^+ (ligne 4). S'il n'existe pas de telles relations, cela signifie qu'aucune hypothèse ne peut accepter cette instance positive. Alors, l'algorithme se termine dans un état d'effondrement (ligne 5).

5.1.2 Instances de E^-

Une instance négative e^- doit être une non solution pour tous les réseaux $(\mathcal{X}, \mathcal{D}, h)$ où $h \in V$. Ainsi,

$$\forall h \in V, \exists B_i \in h/e^-[var(B_i)] \notin rel(B_i)$$

Puisqu'une contrainte violée est suffisante (au lieu de toutes les contraintes satisfaites dans le cas d'une instance positive), G_i n'a pas la propriété de projection[2] exhibée sur S :

$$L(G_i) \neq max_{\leq_g}\{r \in L^H(B_i)/\forall e^- \in E^-, e^-[var(B_i)] \notin r\}$$

On a seulement : $\forall e^- \in E^-, \exists i/\forall r \in L(G_i), e^-[var(B_i)] \notin r$. Mais, il se peut qu'il y ait incertitude sur les contraintes à l'origine du rejet. Et stocker uniquement les bornes locales G_i est insuffisant pour exprimer cette incertitude.

L'approche traditionnelle de l'Espace des Versions consiste à stocker l'ensemble des bornes globales G. Mais ceci peut requérir un espace exponentiel (Hirsh, 1992). Dans le but de rester polynomial, nous ne construisons pas cet ensemble mais codons chaque instance négative e^- par une clause Cl. Chaque contrainte pouvant être à l'origine du

[2] sinon la contrainte définie sur $rel(B_i)$ rejetterait à elle seule toutes les instances de E^-

rejet de e^- sera réifiée en méta-variable de la clause Cl. Sémantiquement, la clause Cl représentera donc la disjonction des différentes explications possibles du rejet de l'instance négative e^-.

Lorsqu'on reçoit une instance négative e^-, on crée une nouvelle clause Cl initialement vide et on examine chaque biais local B_i séparément (lignes 12-14). Ceux dont toutes les valeurs (relations) de leur borne spécifique $L(S_i)$ acceptent déjà e^- sont ignorés (ligne 15). En effet S_i étant la borne inférieure de l'espace des versions local à B_i et S_i ne faisant que se généraliser au cours du temps (voir section 5.1.1), on sait qu'à toute étape de l'apprentissage, la contrainte définie sur $rel(B_i)$ ne sera pas violée par e^-.

Pour toutes les autres contraintes (potentiellement suspectées d'être à l'origine du rejet de e^-), on calcule A_i le sous-ensemble des relations comprises entre $L(S_i)$ et $L(G_i)$ et minimales au sens de $\leq_g$ qui acceptent $e^-[var(B_i)]$, c.à.d. la borne au dessus de laquelle B_i ne doit pas se généraliser si c'est elle qui s'avère être à l'origine du rejet de e^- (ligne 16). Si l'ensemble ainsi construit est vide, c'est que toutes les valeurs encore possibles de la contrainte définie sur $rel(B_i)$ rejettent déjà e^-. Toute hypothèse de l'espace des versions est donc consistante avec e^-, il n'y a rien à faire pour e^-, on passe à l'instance suivante (ligne 17). Sinon (A_i est non vide), on rajoute la méta-variable $(L(G_i) <_g A_i)$ à la clause Cl (ligne 18). La sémantique de cette méta-variable est *"si B_i est coupable du rejet de e^-, alors il faut abaisser (spécialiser) G_i"*.

Pour rejeter e^-, une séquence de contraintes h doit satisfaire au moins une méta-variable de la clause Cl codant e^-. Ce que nous noterons $h \models Cl$. L'ensemble de ces clauses, caractérisant tout l'ensemble E^-, est noté $\mathcal{K}$. Ci-après, nous présentons la maintenance de cette base de clauses.

5.1.3 Maintenance de la base de clauses

i Maintenance des clauses quand des positifs sont ajoutés.

Si une clause Cl contient la méta-variable $(L(G_i) <_g A_i)$ et si, suite à la réception d'instances positives, la borne $L(S_i)$ remonte de sorte à comporter des relations qui ne sont plus inférieures (au sens de $<_g$) à A_i (ligne 6), alors de telles relations ne constituent plus d'explication possible du rejet de l'instance négative codée par Cl et doivent donc être effacées. Soit A'_i le sous-ensemble des relations de A_i qui restent supérieures à la nouvelle borne S_i. Si A'_i est non vide, la nouvelle explication devient $(L(G_i) <_g A'_i)$. Sinon la méta-variable $(L(G_i) <_g A_i)$ est effacée de Cl (lignes 7-9).

ii Équivalence "clause vide/effrondement".

Quand une clause Cl, codant une instance négative e^-, est vide, soit durant sa construction (ligne 19), soit après l'ajout d'instances positives (ligne 10), il n'y a plus d'explication possible pour le rejet de l' instance négative e^-, alors l'espace des versions s'effondre.

iii Les clauses à une méta-variable.

Les clauses à une méta-variable représentent la seule possibilité du rejet de l'instance négative qu'elle code. Elles doivent être reportées sur l'espace des versions local puis effacées de $\mathcal{K}$. Soit $\{(L(G_i) <_g A_i)\}$ une telle clause : $L(G_i)$ doit descendre (se spécialiser) de sorte à devenir plus spécifique que A_i (lignes 11 et 20).

iv Subsumption d'instances négatives.

Une instance négative e_1^- (encodée par Cl_1) subsume e_2^- (encodée par Cl_2) ssi $Cl_1 \subseteq Cl_2$. En effet, si Cl_1 est satisfaite, alors Cl_2 aussi. Il est donc inutile de stocker Cl_2. Les lignes 21-22 de l'Algo. CONACQ maintiennent la minimalité de la base de clauses $\mathcal{K}$.

Algorithme 1: L' algorithme CONACQ

$\mathcal{K} \leftarrow \emptyset$
foreach B_i **do** $L(S_i) \leftarrow \{\bot\};\ L(G_i) \leftarrow \{\top\}$
foreach *training instance e* **do**
1 **if** $e \in E^+$ **then**
2 **foreach** B_i **do**
3 **if** $\exists r \in L(S_i)/e[var(B_i)] \notin r$ **then**
4 $L(S_i) \leftarrow min_{\leq_g}\{r/S_i \leq_g r \leq_g G_i$ and $e[var(B_i)] \in r\}$
5 **if** $L(S_i) = \emptyset$ **then** "collapsing"
6 **foreach** $Cl/(L(G_i) <_g A_i) \in Cl$ *and* $L(S_i) \nleq_g A_i$ **do**
7 $Cl \leftarrow Cl \setminus (L(G_i) <_g A_i)$
8 $A_i' \leftarrow \{r \in A_i/r <_g L(S_i)\}$
9 **if** $A_i' \neq \emptyset$ **then** $Cl \leftarrow Cl \cup (L(G_i) <_g A_i')$
10 **if** $Cl = \emptyset$ **then** "collapsing"
11 **if** $Cl = \{(L(G_i) <_g A_i)\}$ **then**
$L(G_i) \leftarrow max_{\leq_g}\{r/S_i \leq_g r \leq_g G_i$ and $A_i \nleq_g r\}$; $\mathcal{K} \leftarrow \mathcal{K} \setminus Cl$
12 **if** $e \in E^-$ **then**
13 $Cl \leftarrow \emptyset$; $reject \leftarrow false$
14 **foreach** B_i ***while*** $\neg reject$ **do**
15 **if** $\exists r \in L(S_i)/e[var(B_i)] \notin r$ **then**
16 $A_i \leftarrow min_{\leq_g}\{r/S_i \leq_g r \leq_g G_i$ and $e[var(B_i)] \in r\}$
17 **if** $A_i = \emptyset$ **then** $reject \leftarrow true$
18 **else** $Cl \leftarrow Cl \cup (L(G_i) <_g A_i)$
if $\neg reject$ **then**
19 **if** $Cl = \emptyset$ **then** "collapsing"
20 **if** $Cl = \{(L(G_i) <_g A_i)\}$ **then**
$L(G_i) \leftarrow max_{\leq_g}\{r/S_i \leq_g r \leq_g G_i$ and $A_i \nleq_g r\}$; $\mathcal{K} \leftarrow \mathcal{K} \setminus Cl$
21 **else if** $\nexists Cl'/Cl' \subseteq Cl$ **then** $\mathcal{K} \leftarrow \mathcal{K} \cup Cl$
22 **foreach** $Cl''/Cl \subset Cl''$ **do** $\mathcal{K} \leftarrow \mathcal{K} \setminus Cl''$

Exemple 3 Dans la Figure 2, nous décrivons le processus d'apprentissage sur une seule contrainte C_{12} et où le biais B_{12} est le biais de la Figure 1. Initialement $L(S_{12}) = \{\bot\}$ et $L(G_{12}) = \{\top\}$. Quand l'instance positive $e_1^+ = (1, 1)$ est reçue, $L(S_{12})$ monte dans

E	X_1	X_2
e_1^+	1	1
e_2^-	2	1
e_3^-	0	3

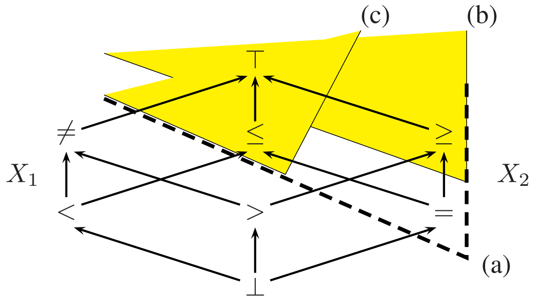
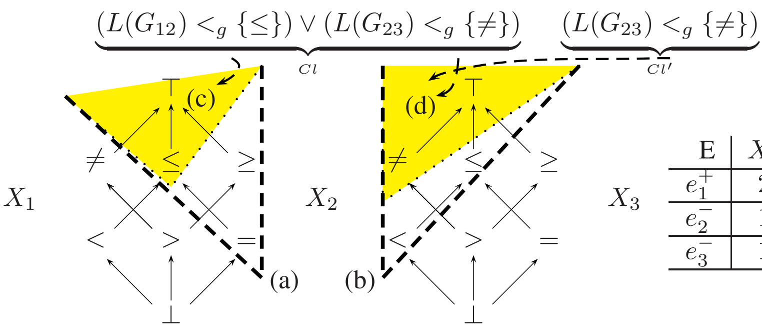


FIG. 2 – Un exemple local

$L^H(B_{12})$ aux relations les plus spécifiques acceptant ce n-uplet (a), ici elle est unique $\{=\}$. Nous savons que la relation finale de la contrainte C_{12} sera plus générale ou équivalente à "$=$". Quand l'instance négative $e_2^- = (2, 1)$ est reçue, il est nécessaire de faire descendre $L(G_{12})$ pour qu'il rejette ce n-uplet (b). A cette étape : $L(S_{12}) = \{=\}$ et $L(G_{12}) = \{\leq\}$. L'instance négative $e_3^- = (0, 3)$ interdit la relation "$\leq$" (c). On a alors $L(S_{12}) = L(G_{12}) = \{=\}$: on parle de convergence locale de la contrainte C_{12}.



E	X_1	X_2	X_3
e_1^+	2	2	5
e_2^-	1	3	2
e_3^-	1	1	0

FIG. 3 – Un exemple global

Exemple 4 Dans la Figure 3, nous décrivons le processus d'apprentissage sur un problème possédant 3 variables X_1, X_2 et X_3. Le biais utilisé est (B_{12}, B_{23}), où $var(B_{12}) = \{X_1, X_2\}$, $var(B_{23}) = \{X_2, X_3\}$ et $L^H(B_{12}) = L^H(B_{23})$ identiques au $L^H(B_i)$ de la Fig. 1. A la réception de $e_1^+ = (2, 2, 5)$, à cause de la propriété de projection de S, S_{12} **et** S_{23} montent ($L(S_{12}) \leftarrow \{=\}$ et $L(S_{23}) \leftarrow \{<\}$) (a) et (b).

Mais G ne possède pas cette propriété de projection : quand $e_2^- = (1, 3, 2)$ est reçue, soit C_{12} doit rejeter le n-uplet $(1, 3)$ **ou** C_{23} doit rejeter $(3, 2)$. Nous construisons donc la clause $Cl = (L(G_{12}) <_g \{\leq\}) \vee (L(G_{23}) <_g \{\neq\})$ pour stocker ces alternatives (c) ou (d). Quand l'instance négative $e_3^- = (1, 1, 0)$ est reçue, nous savons que la contraintes C_{12} n'est pas en cause dans son rejet, car $e_3^-[var(C_{12})] = (1, 1)$ est un n-uplet autorisé de $L(S_{12}) = \{=\}$. La seule explication du rejet de e_3^- est $Cl' = (L(G_{23}) <_g \{\neq\})$ (d). Notons que $Cl' \subset Cl$. L'explication du rejet de e_3^- (d) en est aussi une valide pour e_2^- : Cl est alors subsumée par Cl', et est donc effacée, ainsi que l'explication possible (c). Après ces trois instances, on a pour B_{12} : $L(S_{12}) = \{=\}$ et $L(G_{12}) = \{\top\}$ et pour B_{23} : $L(S_{23}) = \{<\}$ et $L(G_{23}) = \{\leq\}$.

5.2 Exactitude de l'algorithme

Le théorème de représentabilité (Mitchell, 1997) dit que si cn le réseau cible appartient à H (l'espace des hypothèses) et si S et G sont les bornes minimale et maximale de l'espace des versions relativement à l'ordre de généralisation $\leq_G$, alors à chaque étape de l'entraînement, cn est tel que $\exists s \in S, \exists g \in G / s \leq_G cn \leq_G g$.

Afin d'assurer l'exactitude de l'algorithme CONACQ, nous devons seulement prouver que S et G sont les bornes minimale et maximale de V. Nous donnons juste une idée de la preuve. Pour S, c'est facile à prouver grâce à sa propriété de projection. Le S global est le produit cartésien des S_i locaux (Corollaire 1). Et par construction, chaque S_i est la borne minimale de V sur $var(B_i)$. Pour G, c'est un peu plus complexe car, du fait de l'absence de la propriété de projection, G n'est pas équivalent au produit cartésien des G_i locaux. Souvenons nous que pour des raisons de complexité en espace, nous ne stockons pas G en extension mais seulement sa caractérisation sous forme de clauses $\mathcal{K}$. Pour être dans G, une hypothèse h doit rejeter toutes les instances négatives, c.à.d. $h \models \mathcal{K}$. C'est un impliquant [3] de $\mathcal{K}$. De plus, h doit être aussi général que possible relativement à $\leq_G$, ce qui signifie qu'il doit satisfaire aussi peu de méta-variables que possible. C'est donc un impliquant premier de $\mathcal{K}$:

$$G = \{g \in \bigtimes_{i \in 1..m} G_i / g \models^\star \mathcal{K}\}$$

Désormais une hypothèse $h \in G$ doit être telle que $E^- \cap Sol(\mathcal{X}, \mathcal{D}, h) = \emptyset$, et $\not\exists h' \neq h / h \leq_G h'$ et $E^- \cap Sol(\mathcal{X}, \mathcal{D}, h') = \emptyset$. Supposons qu'il existe un tel h'. Puisqu'il rejette toutes les instances négatives, il vient $h' \models \mathcal{K}$ par construction de $\mathcal{K}$. Et alors, h ne sera pas un impliquant premier, ce qui contredit l'hypothèse.

Puisque nos S et G ont les propriétés requises, le corollaire suivant s'applique :

Corollaire 2 **(Mitchell, 1997)** *Étant donné un réseau cible cn et un espace H d'hypothèses, l'espace des versions s'effondre ssi $cn \notin H$, ce qui signifie que le biais sélectionné n'est pas suffisant.*

6 Expérimentations

Nous avons fait quelques expériences préliminaires pour évaluer les capacités d'apprentissage de notre approche. Si le but est de trouver un réseau de contraintes exprimant le problème qu'un utilisateur nous fournit par l'intermédiaire de données d'entraînement, nous pourrions avoir envie de limiter le nombre d'instances pour obtenir un réseau satisfaisant, ou de minimiser le coût de calcul du processus d'apprentissage.

Nous avons procédé à des expériences avec un oracle, qui joue le rôle de l'utilisateur, et un apprenti. L'oracle possède un réseau généré aléatoirement (la cible) noté $< 50, 8, C >$, avec 50 variables ayant pour domaine $\{1, ..8\}$, et un nombre C de

[3] Un terme i est un impliquant de la base de clause A ssi $i \models A$. C'est un impliquant premier, noté $i \models^\star A$, ssi pour tout autre impliquant i', $i' \not\subset i$.

contraintes. Chaque contrainte est choisie aléatoirement dans le biais $\{<, =, >, \leq, \neq, \geq\}$. L'oracle fournit à l'apprenti des solutions et des non solutions. L'apprenti apprend un espace des versions avec l'algorithme CONACQ.

6.1 Ordre des instances

Dans l'expérience suivante, nous essayons d'évaluer la taille de l'espace des versions en fonction de l'ordre dans lequel les instances d'un ensemble E de taille 100 sont données par l'oracle à l'apprenti pour apprendre un réseau $< 50, 8, 50 >$. E contient 10 instances positives et 90 négatives.

Le Tableau 1 présente le temps requis par l'apprenti pour acquérir l'espace des versions produit V en fonction de la date d'arrivée des 10 instances positives : au tout début (a), au milieu du processus (b), ou en fin d'interaction (c).

TAB. 1 – Effet de l'ordre des instances positives

Date d'introduction des positifs	0 (a)	50 (b)	90 (c)
Temps calcul (en sec.)	3.3	5.1	8.6
$log(\|V\|)$	2,234	2,234	2,234

Nous observons que 'le plus tôt sera le mieux' semble être une bonne heuristique pour les instances positives. En effet, la borne spécifique S monte rapidement dans l'espace des hypothèses avec les instances positives, réduisant ainsi la taille de l'espace des versions. De ce fait, le temps CPU utilisé est aussi réduit quand les instances positives arrivent au début. Mais nous pouvons voir que la taille finale de l'espace des versions n'est pas affecté par l'ordre des instances. Ceci est dû à la propriété de commutativité de l'espace des versions. (Voir la Section 4.)

6.2 Instances partielles

Dans certains cas, l'utilisateur peut rejeter une instance en justifiant par une sous-instance négative. Prenons l'exemple d'une agence immobilière, le client (oracle) rejetterait un appartement en précisant *"le séjour ne me convient pas"*. L'agent immobilier (apprenti) sait que la violation porte sur des variables du séjour, ce qui constitue une aide précieuse. L'aide apportée par de telles justifications des rejets peut être mesurée en fournissant à notre apprenti des instances partielles. Dans l'expérience suivante (Tableau 2), l'oracle fournit à l'apprenti 90 instances négatives partielles (après 10 positives complètes). Nous avons essayé avec des instances partielles comportant 2, 5 ou 10 variables instanciées, et avons reporté la taille de l'espace des versions et celle de la base de clauses après réception des 100 instances.

TAB. 2 – Effet des instances partielles

Nb de variables des instances partielles	50	10	5	2
$log(\|V\|)$	2,234	2,233	2,225	2,144
$\|K\|$ (10^4 méta-variables)	7.6	6.1	3.2	0

Nous observons que les instances partielles accélèrent le processus de convergence de l'espace des versions. Plus ces instances partielles sont petites, plus elles sont utiles. Ceci ouvre une direction prometteuse pour aider le processus d'apprentissage : demander à l'utilisateur de justifier pourquoi il rejette certaines instances pourrait participer à réduire la longueur du dialogue entre le système et l'utilisateur. Aspect qui s'avèrera crucial quand l'apprentissage se fait par interaction avec un utilisateur humain.

6.3 Contraintes non représentables et effondrement

Nous avons vu que notre algorithme apprend un réseau exprimé dans un biais donné. Un biais trop restrictif conduit à l'effondrement car il ne peut pas exprimer le réseau cible. Ceci ce passe quand certaines contraintes du réseau cible sont non représentables par un biais donné. Dans l'expérience suivante, (Tableau 3), nous analysons l'effet sur la vitesse d'effondrement de la proportion de contraintes non représentables dans le réseau cible. Pour chaque proportion de contraintes non représentables, nous apprenons 100 réseaux $< 50, 10, 50 >$ différents. (Non représentable signifie, dans ce cas, une contrainte aléatoire n'appartenant pas au biais $\{<, =, >, \leq, \neq, \geq\}$.) Nous comptons le nombre de fois où l'espace des versions s'est effondré après 1000 instances. Notons que cette expérience porte davantage sur la reformulation automatique que sur la modélisation interactive.

TAB. 3 – Effet des contraintes non représentables

Ratio de contraintes non représentables	10%	25%	50%
% d'effondrement	47	78	100

Nous observons que plus il y a de contraintes non représentables, plus vite l'espace des versions s'effondre. Il faut noter que nous avons observé quelques réseaux de contraintes possédant peu de contraintes non représentables qui ne se sont pas effondrés même après plusieurs millions d'instances. Ce qui semble violer le Corollaire 2. En fait, cela correspond au cas où les contraintes non représentables deviennent représentables par propagation des autres contraintes. Imaginons trois variables X_1, X_2, et X_3 reliées par les contraintes suivantes dans un réseau de contraintes : $X_1 = X_2$, $X_2 = X_3$, et $(X_1 = X_3) \vee (X_1 = X_3 + 5)$. Il est évident que la contrainte entre X_1 et X_3 est non représentable dans le biais $\{<, =, >, \leq, \neq, \geq\}$ et chaque réseau la contenant serait supposé s'effondrer. Ce n'est pas le cas dans notre exemple car nous pouvons retreindre la contrainte entre X_1 et X_3 à $(X_1 = X_3)$ en utilisant la transitivité.

Il est important de noter que si nous avions utilisé une technique apprenant chaque contrainte séparément sur ce réseau, l'espace se serait effondré en essayant d'apprendre la contrainte entre X_1 et X_3.

6.3.1 Contraintes Implicites

Ce phénomène général des contraintes, qui peuvent être inférées à partir d'autres contraintes, a un autre effet : il empêche l'espace des versions de converger autant que si chaque contrainte était apprise séparément. Prenons à nouveau un exemple avec trois variables X_1, X_2, et X_3, reliées dans un réseau cible par les contraintes $X_1 = X_2$, $X_2 = X_3$, et $X_1 = X_3$. Imaginons qu'à une certaine étape du processus d'apprentissage, l'espace des versions local à (X_1, X_3), ne contienne plus que $X_1 = X_3$ et $X_1 \leq X_3$. Si les données d'entraînement ne contiennent que des instances complètes, il est impossible de faire converger à la contrainte $X_1 = X_3$ car chaque instance qui permettrait de retirer $X_1 \leq X_3$ de l'espace des versions (c.a.d., $((X_1, 1), (X_2, 2), (X_3, 2))$) sera aussi rejetée par $X_1 = X_2$ ou par $X_2 = X_3$. Ainsi, la frontière G ne détectera jamais la culpabilité de $X_1 \leq X_3$. Ce phénomène peut donc conduire à une taille de l'espace des versions plus grande qu'elle ne devrait être en réalité.

Appliquer différents niveaux de consistance locale semble être un moyen pour améliorer la diminution de l'espace des versions, en ajoutant les contraintes implicites au réseau appris. (Dans le précédent exemple, la chemin consistance aurait suffit pour déduire que la seule relation valable entre X_1 et X_3 était $X_1 = X_3$.)

7 Conclusion

Nous avons proposé une méthode originale pour apprendre des réseaux de contraintes à partir d'instances qui doivent ou ne doivent pas être des solutions. La technique utilisée est basée sur les espaces des versions, un paradigme d'apprentissage automatique qui présente de bonnes propriétés (c.à.d. incrementalité, commutativité, relativement à l'ordre des données d'entraînement.) qui vont être essentielles dans un processus interagissant avec un utilisateur. Même si cet article est principalement une description du processus d'acquisition d'un réseau de contraintes à partir d'instances, nous pouvons facilement prévoir les diverses applications qu'il pourrait avoir : assister un novice dans la modélisation de son problème, ou aider un expert à tester si une bibliothèque de contraintes donnée avec de bonnes caractéristiques calculatoires peut coder le problème.

Dans la dernière section de cet article nous avons présenté des expériences préliminaires qui montrent que notre approche soulève des questions importantes, telles que la vitesse du processus d'apprentissage ou les contraintes implicites.

Références

FREUDER E. (1982). A sufficient condition for backtrack-free search. *Journal of the ACM*, **29**(1), 24–32.

HIRSH H. (1992). Polynomial-time learning with version spaces. In *National Conference on Artificial Intelligence*, p. 117–122.

LITTLE J., GEBRUERS C., BRIDGE D. & FREUDER E. (2002). Capturing constraint programming experience : A case-based approach. In *CP-02 Workshop on Reformulating Constraint Satisfaction Problems*.

MITCHELL T. (1997). Concept learning and the general-to-specific ordering. In *Machine Learning*, chapter 2, p. 20–51. McGraw Hill.

MONTANARI U. (1974). Networks of constraints : Fundamental properties and applications to picture processing. *Information Sciences*, **7**(2), 95–132.

O'CONNELL S., O'SULLIVAN B. & FREUDER E. (2002). Strategies for interactive constraint acquisition. In *CP-02 Workshop on User-Interaction in Constraint Satisfaction*.

ROSSI F. & SPERDUTI A. (1998). Learning solution preferences in constraint problems. *Journal of experimental and theoretical computer science*, **10**.

VAN BEEK P. & DECHTER R. (1995). On the minimality and global consistency of row-convex constraint networks. *Journal of the ACM*, **42**(3), 543–561.

WALLACE M. (1996). Practical applications of constraint programming. *Constraints*, **1**(1–2), 139–168.