Apprentissage interactif de réseau de contraintes

Mathias Paulin

Apprentissage interactif de réseau de contraintes

Mathias Paulin

LIRMM - CNRS (UMR 5506)
161 rue Ada 34392 Montpellier cedex 5 - France
paulin@lirmm.fr

Abstract :

La programmation par contraintes est une technologie désormais largement utilisée pour résoudre des problèmes combinatoires dans les applications industrielles. Cependant, malgré le succès croissant qu'elle connaît, la diffusion de la programmation par contraintes est freinée par le manque d'experts connaissant ce paradigme. Depuis trois ans, une partie de la communauté "contraintes" travaille sur l'acquisition automatique de réseau de contraintes à partir d'instances que l'utilisateur accepte ou n'accepte pas comme solution à son problème. C'est dans cette optique que la plateforme Conacq a été proposée. Néanmoins, jusqu'à présent, cette dernière est passive vis à vis de l'utilisateur, c'est à dire basée sur la capacité de ce dernier à fournir des exemples significatifs de son problème. Dans cet article, nous proposons une amélioration de cette plateforme en développant un Conacq interactif, capable de poser à l'utilisateur des questions dont le but est d'augmenter plus rapidement et de manière conséquente la connaissance de la plateforme. Nous étudierons donc ici différentes approches de génération de questions dans le cadre de l'apprentissage de réseau de contraintes.

Mots-clés : Programmation par contraintes, Apprentissage.

1 Introduction

La programmation par contraintes (CP) connaît un succès croissant depuis qu'elle a montré sa capacité à résoudre des problèmes réels difficiles hors de portée des autres approches. Cependant, malgré ce succès, la diffusion de la CP est principalement freinée pour deux raisons. La première vient du fait que de nombreuses applications réelles (la mise en place des emplois du temps par exemple) qui pourraient être résolues à l'aide de réseaux de contraintes, ne font pas appel à la CP car les experts "métiers" (la personne chargée du planning dans notre exemple) en charge de ces applications ne maîtrisent pas la CP, et sont donc incapables modéliser leur problème sous la forme de réseaux de contraintes. Le second frein à la diffusion de la CP résulte du manque d'experts capables de modéliser efficacement des problèmes très combinatoires qui ne peuvent être

résolus à l'aide de modèles naïfs. Notre article n'a pas pour objectif de s'attaquer au second point, qui relève d'un problème de reformulation, mais présente des solutions capables d'aider un expert "métier", totalement novice en CP, à modéliser son problème sous la forme d'un réseau de contraintes.

Depuis trois ans, une partie de la communauté "contraintes" travaille sur l'acquisition automatique de réseau de contraintes à partir d'instances que l'utilisateur accepte ou n'accepte pas comme solution à son problème (Legtchenko Lallouet 2005). C'est dans cette optique que la plateforme CONACQ a été proposée (Coletta Bessière *et al.* 2003). Néanmoins, jusqu'à présent, cette dernière est passive vis à vis de l'utilisateur, c'est à dire basée sur la capacité de ce dernier à fournir des exemples significatifs de son problème. Dans cet article, nous proposons une approche théorique, validée par des expérimentations simples, permettant d'améliorer cette plateforme en développant un CONACQ interactif, capable de poser à l'utilisateur des questions dont le but est d'augmenter plus rapidement et de manière conséquente la connaissance acquise.

Le reste de cet article est organisé comme suit. La section 2 rappelle quelques définitions préliminaires sur les réseaux de contraintes et résume succinctement le fonctionnement de la plateforme CONACQ . La section 3 présente ensuite une première approche, basée sur les principes d'Active Learning (Cohn Ghahramani *et al.* 1996), consistant à effectuer une sélection judicieuse d'instances générées aléatoirement. Dans la section 4, nous caractérisons la question optimale et montrons comment la construire. La section 5 illustre les deux modules précédemment présentés au travers de quelques expérimentations préliminaires, avant de conclure le travail (section 6).

2 Préliminaires

Dans cette section, nous introduisons les concepts de base utilisés dans l'article. Nous rappelons quelques définitions de la programmation par contraintes avant de présenter succinctement la plateforme CONACQ ainsi que quelques propriétés de l'Espace des Versions.

2.1 Rappel sur la programmation par contraintes

Un réseau de contraintes $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ est défini par un ensemble fini $\mathcal{X} = \{x_1, \ldots, x_n\}$ de n variables prenant chacune une valeur de leur domaine respectif $D_{x_1}, \ldots, D_{x_n}$, éléments de $\mathcal{D}$, et par $\mathcal{C} = \{C_1, \ldots, C_m\}$ une séquence de contraintes sur $\mathcal{X}$. Une contrainte C_i est définie par la séquence $var(C_i)$ des variables sur lesquelles elle porte, et par la relation $sol(C_i)$ qui spécifie les n-uplets autorisés sur $var(C_i)$. L'assignement de valeurs aux variables de $var(C_i)$ satisfait C_i si il appartient à $sol(C_i)$. Une instance e sur $\mathcal{X}$ est un n-uplet $(v_1, \ldots, v_n) \in D_{x_1} \times \ldots \times D_{x_n}$. On dit qu'une instance est une solution du réseau si elle satisfait toutes les contraintes du réseau. Sinon, c'est une non solution. On note $Sol(\mathcal{X}, \mathcal{D}, \mathcal{C})$ l'ensemble des solutions de $(\mathcal{X}, \mathcal{D}, \mathcal{C})$. Lorsque toutes les contraintes de $\mathcal{C}$ mettent en jeu exactement deux variables, nous disons que les contraintes et le réseau sont *binaires*. Dans cet article, nous restreignons notre étude au cas binaire ; La notation C_{ij} signifie dans ce cas que

la contrainte C est placée sur les variables x_i et x_j du problème.

L'apprentissage de réseau de contraintes a pour premier but de soulager l'utilisateur en fournissant à ce dernier des méthodes semi-automatiques pour acquérir les contraintes du problème qu'il cherche à modéliser. Comme point de départ, nous supposons que l'utilisateur connaît l'ensemble $\mathcal{X}$ des variables du problème ainsi que leur domaine $\mathcal{D}$ de valeurs possibles. Il est aussi supposé capable de se prononcer sur la validité d'une instance, il fournit à ce titre E^+ un sous-ensemble des solutions de son problème et E^- un ensemble de non solutions. Par ailleurs, l'objectif de CONACQ consiste à modéliser le problème de l'utilisateur dans un solveur de contraintes. Notre biais d'apprentissage, noté $\mathcal{B}$ est en conséquence une librairie de contraintes issue de ce solveur.

On dit qu'une séquence de contraintes appartient au biais si et seulement si cette séquence n'utilise que des contraintes de la librairie du biais.

Apprendre un réseau de contraintes consiste à rechercher une séquence de contraintes $\mathcal{C}$ appartenant à un biais d'apprentissage $\mathcal{B}$ donné, et dont l'ensemble des solutions est un sur-ensemble de E^+ ne contenant aucun élément de E^-. Le Problème d'Acquisition de Contraintes se définit alors formellement comme suit :

Définition 1 (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 d'apprentissage $\mathcal{B}$, le problème d'acquisition de contraintes consiste à trouver une séquence de contraintes $\mathcal{C}$ telle que :

$$\begin{cases} \mathcal{C} \in \mathcal{B}, \\ \forall e^- \in E^-, e^- \text{ n'est pas solution de } (\mathcal{X}, \mathcal{D}, \mathcal{C}), \text{ et,} \\ \forall e^+ \in E^+, e^+ \text{ est solution de } (\mathcal{X}, \mathcal{D}, \mathcal{C}). \end{cases}$$

2.2 Processus d'apprentissage de la plateforme CONACQ

Présentée en 2003 (Coletta Bessière *et al.* 2003), CONACQ est une plateforme d'apprentissage de réseau de contraintes basée sur l'Espace des Versions (Mitchell 1997). Dans cet article, nous considérons CONACQ comme une boite noire qui, étant donnés deux ensembles E^+ et E^- de données d'entraînement,[1] renvoie une formule SAT, notée $\mathcal{K}$. Le vocabulaire de cette base est un ensemble d'atomes y^r_{ij}, où r est une contrainte de $\mathcal{B}$ et i et j sont tels que x_i et x_j appartiennent à $\mathcal{X}$. Le littéral y^r_{ij} signifie, lorsqu'il est satisfait, que la contrainte r doit être placée entre les variables x_i et x_j du problème. Un résultat notoire de la formulation SAT de CONACQ (Bessière Coletta *et al.* 2005) est : *Pour toute solution de $\mathcal{K}$, le réseau de contraintes associé est consistant avec les données d'entraînement*.

La construction de $\mathcal{K}$ suit le mode opératoire suivant. Pour chaque instance d'entraînement e fournie par l'utilisateur, CONACQ construit l'ensemble $\mathcal{K}_e$ des littéraux correspondant aux contraintes les plus générales qui permettent de rejeter e. La

[1] respectivement ensemble de solutions et de non solutions au problème de l'utilisateur.

mise à jour de la base de connaissance $\mathcal{K}$ diffère ensuite selon la classe d'appartenance de e :

- Si e est une instance négative, alors une seule des contraintes identifiées dans $\mathcal{K}_e$ suffit à expliquer le rejet de e. On ajoute donc à $\mathcal{K}$ la disjonction des éléments de $\mathcal{K}_e$.
- Si e est une instance positive, alors CONACQ est assurée qu'aucune des contraintes identifiées dans $\mathcal{K}_e$ n'appartient au concept cible (sinon, e serait une non solution). En conséquence, $\mathcal{K}$ est mis à jour en ajoutant la conjonction des négations des littéraux de $\mathcal{K}_e$.

L'analyse des instances positives de E^+ permet de simplifier les clauses de rejet des instances négatives de E^- et permet, *in fine*, d'identifier exactement quelles sont les contraintes mises en jeu dans le réseau cible.

L'exemple 1 permet d'illustrer simplement le mode de fonctionnement de la plateforme CONACQ .

Exemple 1
Dans cet exemple, nous cherchons un réseau de contraintes mettant en jeu trois variables x_1, x_2 et x_3 dont les domaines sont $D(x_i) = \{1, 2, 3, 4\}$. Notre biais d'apprentissage est restreint à $\mathcal{B} = ((\{x_1, x_2\}, L), (\{x_2, x_3\}, L))$ où L est la librairie de contraintes arithmétiques binaires $L = \{<, \leq, =, \geq, >, \neq\}$. Soient maintenant $E^- = \{(3, 2, 1), (3, 3, 2)\}$ et $E^+ = \{(1, 3, 2)\}$ les données d'entraînement.

Comme le montre le tableau 1, l'analyse des instances négatives implique une mise à jour de $\mathcal{K}$ au moyen de clauses disjonctives alors que les instances positives permettent d'augmenter $\mathcal{K}$ par conjonction de littéraux mis à Faux.

instance	$\mathcal{K}_e$	$\mathcal{K}$
$e_1^- = (3, 2, 1)$	$\{y_{12}^{\leq}; y_{23}^{\leq}\}$	$(y_{12}^{\leq} \vee y_{23}^{\leq})$
$e_2^- = (3, 3, 2)$	$\{y_{12}^{\neq}; y_{23}^{\leq}\}$	$(y_{12}^{\leq} \vee y_{23}^{\leq}) \wedge (y_{12}^{\neq} \vee y_{23}^{\leq})$
$e_1^+ = (1, 3, 2)$	$\{y_{12}^{>}; y_{23}^{<}\}$	$(y_{12}^{\leq}) \wedge (y_{12}^{\neq}) \wedge (\neg y_{12}^{>}) \wedge (\neg y_{23}^{\leq})$

Table 1: Évolution de la base de connaissance de CONACQ pour l'exemple 1.

Après l'analyse de e_1^-, $(y_{12}^{\leq} = 1)$ est une solution pour $\mathcal{K}$. $P_1 = \{(x_1 \leq x_2, x_2 = x_3)\}$ est alors un réseau consistant. Une solution pour $\mathcal{K}$ après e_2^- peut être $(y_{12}^{\leq} = y_{12}^{\neq} = 1)$, on extirpe alors $P_2 = \{(x_1 < x_2, x_2 = x_3)\}$ réseau de contraintes consistant. Le réseau $P_3 = \{(x_1 < x_2, x_2 \geq x_3)\}$ est enfin consistant avec $E = E^+ \cup E^-$ après analyse de e_1^+.

2.3 Espace des Versions

Comme nous l'avons précisé précédemment, CONACQ est basée sur l'Espace des Versions (Mitchell 1997), un paradigme d'apprentissage automatique présentant de

bonnes propriétés (incrémentalité, commutativité, relativement à l'ordre des données d'entraînement).

Définition 2 (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 un ordre partiel $\leq_G$, un espace des versions 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, celui des éléments maximaux.

Définition 3
Étant donné un ensemble E d'instances, on note S_{ij}^E l'état de la borne spécifique S relative au couple (x_i, x_j) après l'analyse de E.

Dans le cadre de l'apprentissage de réseau de contraintes, une hypothèse h est une séquence de contraintes appartenant à $\mathcal{B}$ et l'Espace des Versions est l'ensemble de tous les réseaux de contraintes consistants du problème d'acquisition de contraintes. Par ailleurs, la propriété des solutions la formule $\mathcal{K}$ produite par la plateforme CONACQ (*c.f.* 2.2) et la définition 2 permettent de déduire que l'ensemble $Sol(\mathcal{K})$ des solutions de la base $\mathcal{K}$ matche exactement l'Espace des Versions après analyse des données d'entrainement.

3 Une première approche de questionnement

À l'heure actuelle, CONACQ attend passivement les instances fournies par l'utilisateur. Cette approche est cependant perfectible, dans la mesure où elle repose sur la capacité pour l'utilisateur de fournir des instances significatives de son problème. Dans cette section, nous présentons une première technique de questionnement basée sur la sélection judicieuse d'instances générées aléatoirement.

3.1 Principe général

L'attente passive de CONACQ vis à vis des données d'entraînement constitue pour le moment un défaut majeur pour la plateforme. En effet, cela suppose de la part de l'utilisateur la capacité de produire des exemples significatifs de son problème. En pratique, le manque d'efficacité du processus d'apprentissage résulte de deux causes principales :

1. L'utilisateur est limité par ses propres connaissances du problème.[2] Les exemples qu'il fournit sont en général relativement similaires (peu de valeurs diffèrent

[2] Car le problème est difficile à résoudre ; c'est d'ailleurs pour cette raison que l'utilisateur souhaite utiliser CONACQ .

d'une instance à l'autre). Dans ce cas leur analyse augmente peu, voire pas, la connaissance acquise.

2. Durant le processus d'apprentissage, deux instances apparemment très différentes peuvent être sémantiquement identiques pour un biais d'apprentissage donné. Par exemple, $e_1 = (1, 2, 3, 4)$ et $e_2 = (2, 3, 4, 5)$ produisent la même connaissance si on utilise la librairie arithmétique binaire $L = \{<, \leq, =, \geq, >, \neq\}$.

Pour éviter ces problèmes, nous proposons un premier générateur de questions. Notre approche consiste à doter la plateforme d'un générateur d'instances aléatoires que CONACQ pourra utiliser de la manière suivante : il présentera à l'utilisateur des instances e générées aléatoirement afin que ce dernier statue sur leur validité. La réponse obtenue servira alors à augmenter la connaissance acquise. Doté d'un tel générateur, la plateforme pourrait alors combattre les problèmes liés au biais de l'utilisateur (point 1).

Pour combattre les problèmes de similarité sémantique (point 2), nous ajoutons la fonctionnalité suivante : Avant de demander à l'utilisateur de statuer sur la validité des instances générées aléatoirement, nous les présentons d'abord à CONACQ afin que ce dernier réalise une sélection parmi ces instances. Pour chaque instance e générée, CONACQ estime le gain de connaissance qu'apporterait e à la connaissance actuelle $\mathcal{K}$. e n'est alors présentée que dans le cas où la réponse de l'utilisateur apporte un gain de connaissance. Avec cette approche, nous mettons en place une sélection des questions dans le but d'améliorer sans cesse la connaissance acquise. Notre démarche s'inscrit donc dans le cadre de l'Active Learning décrit dans (Cohn Ghahramani *et al.* 1996).

En utilisant l'Espace des Versions (c.f. section 2.3), à chaque étape du processus d'apprentissage, la propriété suivante est vérifiée : Chaque instance rejetée par une contrainte de la borne générale G de l'espace des versions est rejetée par tous les réseaux de contraintes encore possibles. De même, toute instance acceptée par la borne spécifique S est acceptée par tous les réseaux de contraintes encore disponibles. Enfin, toute instance non acceptée par S mais non rejetée par G est seulement acceptée par une partie des réseaux de contraintes encore disponibles (et rejetée par les autres).

Nous déduisons de cette propriété que les instances intéressantes pour CONACQ sont celles qui sont situées entre les bornes S et G de l'Espace des Versions.

Illustrons maintenant le principe de sélection à l'aide de l'exemple 2.

Exemple 2
Dans cet exemple, nous cherchons un réseau de contraintes mettant en jeu trois variables $\mathcal{X} = \{x_1, x_2, x_3\}$ où $D(x_1) = D(x_2) = D(x_3) = \{1, 2, 3, 4, 5, 6\}$. Nous utilisons pour cela le biais d'apprentissage $\mathcal{B} = ((\{x_1, x_2\}, L), (\{x_2, x_3\}, L))$, où L est la librairie $\{<, \leq, =, \geq, >, \neq\}$. Soient $e_1^+ = (2, 2, 5)$ et $e_2^- = (1, 3, 2)$ les deux premières instances d'entraînement. Après les avoir analysées, le réseau de contraintes le plus spécifique construit met en jeu deux contraintes $P_S = \{(x_1 = x_2, x_2 < x_3)\}$. Il existe cependant deux possibilités pour le rejet de e_2^-, le réseau de contraintes issu de la borne générale G s'exprime donc de la façon suivante : $P_G = \{(x_1 \geq x_2), (x_2 \leq x_3)\}$.

Soient maintenant $i_1 = (4, 4, 6)$, $i_2 = (5, 6, 1)$ et $i_3 = (5, 5, 5)$ trois instances produites par notre générateur aléatoire. La première instance i_1 ne sera pas présentée[3] à l'utilisateur car elle est déjà acceptée par le réseau le plus spécifique P_S. Il est tout autant inutile de demander à l'utilisateur de statuer sur la validité de i_2 car cette dernière est déjà rejetée par P_G. Enfin, $i_3 = (5, 5, 5)$ constitue une instance intéressante qui doit être présentée à l'utilisateur, dans la mesure où la connaissance actuelle ne permet pas de statuer sur sa validité. Si l'utilisateur considère que i_3 est une solution à son problème, la borne spécifique S sera mise à jour $P_{S'} = \{(x_1 = x_2, x_2 \leq x_3)\}$. Sinon ($i_3$ est une non solution), la borne générale G sera abaissée à $P_{G'} = \{(x_1 \geq x_2, x_2 \leq x_3), (x_2 < x_3)\}$.

3.2 Formalisation

Dans cette section, nous présentons une formalisation du processus de sélection des instances générées aléatoirement. Soit e_q une instance ainsi générée et soit Cl_q la clause codant pour son rejet, telle que Cl_q a été simplifiée par propagation unitaire de la connaissance $\mathcal{K}$. Si $Cl_q = \emptyset$, alors e_q est dores et déjà acceptée par S, elle sera donc filtrée par le module de sélection, c'est à dire qu'elle ne sera pas présentée à l'utilisateur. Si il existe une clause $Cl \in \mathcal{K}$ telle que $Cl_q \supseteq Cl$, alors e_q est déjà rejetée par la borne spécifique G et sera là encore filtrée. Toutes les autres instances générées sont intéressantes en vue d'une augmentation de la connaissance de CONACQ .

Il convient cependant de noter que deux cas se distinguent plus particulièrement : $|Cl_q| = 1$ et $Cl_q \subset Cl$. La méthode du *near-miss* présentée dans (Smith Rosenbloom 1990) traite du cas où $|Cl_q| = 1$. Dans le cas où $Cl_q \subset Cl$, la mise à jour de Cl dépend de la classification de e_q par l'utilisateur. Si e_q est déclarée négative, Cl prend pour nouvelle valeur Cl_q. Dans le cas contraire, Cl devient $Cl \backslash Clq$.

Exemple 3
Dans cet exemple, et à l'aide du tableau 2, nous illustrons nos précédents propos en décrivant le processus de sélection effectuée par CONACQ lors de l'analyse des instances de l'exemple 2. Il convient de remarquer que Cl_q se vide par propagation unitaire de $\mathcal{K}$ pour i_1, et $Cl_q \supseteq Cl$ pour i_2. Enfin, la clause Cl_q construite pour l'analyse de la troisième instance permet de conclure que i_3 est une question intéressante qui peut être présentée à l'utilisateur.

Pour terminer cette section, il convient de noter que la taille de la clause $|Cl_q|$ correspond à la distance qui sépare e_q de la borne spécifique S. En conséquence, si $|Cl_q|$ est grande alors e_q est très proche de la borne générale dans l'Espace des Versions. Dans ce cas, le gain de connaissance pour CONACQ sera très important si e_q est une solution au problème de l'utilisateur. En revanche, le gain sera réduit si e_q est une non solution. Symétriquement, si $|Cl_q|$ est réduite, l'apport de connaissance sera important si e_q est une instance négative du problème, mais réduit dans le cas contraire. En se basant sur la taille des clauses Cl_q, nous pouvons envisager une sélection des instances aléatoires

[3]Sauf si nous cherchons un effondrement de l'Espace de Versions afin de montrer que le biais d'apprentissage $\mathcal{B}$ n'est pas assez expressif pour le problème étudié.

Instance	Analyse de CONACQ	Cl_q resolved by $\mathcal{K}$
$e_1^+ = (2,2,5)$	$\mathcal{K} = (\neg y_{12}^{\neq}) \wedge (\neg y_{23}^{>})$	–
$e_2^- = (1,3,2)$	$Cl = (y_{12}^{>} \vee y_{23}^{<})$	–
$i_1 = (4,4,6)$	$Cl_q = (y_{12}^{\neq} \vee y_{23}^{>})$	$\emptyset$
$i_2 = (5,6,1)$	$Cl_q = (y_{12}^{>} \vee y_{23}^{<})$	$(y_{12}^{>} \vee y_{23}^{<}) = Cl$
$i_3 = (5,5,5)$	$Cl_q = (y_{12}^{\neq} \vee y_{23}^{\neq})$	$(y_{23}^{\neq})$

Table 2: Sélection effectuée par CONACQ pour les instances de l'exemple 2.

selon leur positionnement dans l'espace de versions ; nous sommes dans ce cas dans le cadre d'une généralisation du *near-miss*. Il convient par ailleurs de noter que de telles heuristiques ont été étudiées (O'Connell O'Sullivan *et al.* 2002) dans le cadre de l'apprentissage d'une contrainte unique.

4 Optimalité du questionnement

La section précédente a permis de dégager une première approche de questionnement. Cependant, le gain de connaissance apporté par une question varie selon la position de cette dernière dans l'Espace des Versions et selon la réponse donnée par l'utilisateur. Dans notre recherche d'optimisation, il convient maintenant d'identifier la meilleure question à poser au cours du processus d'apprentissage, c'est à dire celle dont le gain de connaissance est maximum quelque soit la réponse de l'utilisateur.

Notre objectif dans cette section consiste donc à caractériser une question optimale ainsi qu'à évaluer la complexité relative à sa construction.

4.1 Caractérisation d'une question optimale

À chaque étape du processus d'apprentissage, la base SAT $\mathcal{K}$ renferme l'ensemble des connaissances acquises. Soit $Cl = (y_{ij}^{r} \vee \ldots \vee y_{i'j'}^{r'})$ une clause de $\mathcal{K}$ codant le rejet d'une instance négative e (on suppose ici que Cl a été réduite par propagation unitaire). Cl code l'ensemble des contraintes qui rejètent e, et dont au moins une doit faire partie du réseau cible si nous voulons que ce dernier soit consistant avec les données d'entraînement.

L'objectif de l'interaction avec l'utilisateur consiste à trouver, parmi toutes ces contraintes, celle qui appartient réellement au réseau cible. Si nous utilisons la technique du *near-miss* (Smith Rosenbloom 1990), nous devons poser $|Cl|$ questions dans le pire des cas. Nous proposons ici une autre technique nous permettant de réduire le nombre d'interactions à $log(|Cl|)$ questions. Il convient de noter que notre idée est inspirée de celle de Mitchell (Mitchell 1997), qui garantit un nombre logarithmique de questions pour converger.

Propriété 1 (Caractérisation d'une question optimale)
Étant donnée $\mathcal{K}$ une base SAT représentant un ensemble de connaissances et $Cl \in \mathcal{K}$ une clause codant une non solution des données d'entraînement. Une instance e_q constitue une question optimale relativement à Cl si la clause Cl_q codant le rejet de e_q est telle que :

$$Cl_q \subset Cl \quad \textit{et} \quad |Cl_q| = \frac{|Cl|}{2}$$

Preuve. Soit une instance e_q telle que la clause Cl_q codant pour son rejet soit telle que $Cl_q \subset Cl$ et $|Cl_q| = |Cl|/2$. Si $e_q \in E^-$, alors Cl_q sera ajoutée à la base SAT $\mathcal{K}$. Comme $Cl_q \subset Cl$, Cl sera subsumée par Cl_q. Dans le cas contraire ($e_q \in E^+$), les littéraux y^r_{ij} appartenant à Cl_q seront négatés dans $\mathcal{K}$, réduisant ainsi, par propagation unitaire, Cl à $Cl \setminus Cl_q$. Quelque soit la classe d'appartenance de e_q (déterminée par l'utilisateur), e_q nous assure une division par 2 de la taille de Cl. En conséquence, e_q correspond bien à une question optimale puisque le gain de connaissance obtenu ne dépend pas de la réponse de l'utilisateur. □

Exemple 4
Pour illustrer nos propos, nous considérons dans cet exemple un ensemble de variables $\mathcal{X} = \{x_1, x_2, x_3, x_4, x_5\}$ avec $D(x_i) = \{1, 2, 3, 4\} \forall i \in [1..5]$. Le biais d'apprentissage $\mathcal{B} = (\{\{x_i, x_{i+1}\}, L) \forall i \in [1..4]\})$, où $L = \{<, \leq, =, \geq, >, \neq\}$.

instance	CONACQ 's output
$e_1^+ = (1, 2, 4, 3, 1)$	$\mathcal{K} = (\neg y^{\geq}_{12}) \wedge (\neg y^{\geq}_{23}) \wedge (\neg y^{\leq}_{34}) \wedge (\neg y^{\leq}_{45})$
$e_2^- = (2, 2, 2, 2, 2)$	$Cl = (y^{\neq}_{12} \vee y^{\neq}_{23} \vee y^{\neq}_{34} \vee y^{\neq}_{45})$

Table 3: Analyse des instances pour l'exemple 4.

Le tableau 3 renferme les deux premières instances fournies par l'utilisateur ainsi que l'analyse correspondante effectuée par CONACQ . Après l'analyse de ces deux instances, le réseau de contraintes le plus spécifique établi par CONACQ est $P_S = (x_1 < x_2, x_2 < x_3, x_3 > x_4, x_4 > x_5)$ et l'ensemble des réseaux de contraintes permettant d'expliquer le rejet de e_2^- est $P_G = \{(x_1 \neq x_2), (x_2 \neq x_3), (x_3 \neq x_4), (x_4 \neq x_5)\}$.

Notre objectif est donc maintenant de déterminer quelle est la cause réelle du rejet de e_2^- en utilisant le procédé décrit dans nos propos précédents. La clause correspondant à e_2^- est $Cl = (y^{\neq}_{12} \vee y^{\neq}_{23} \vee y^{\neq}_{34} \vee y^{\neq}_{45})$. Nous allons dans un premier temps chercher à savoir si $(x_1 \neq x_2)$ et $(x_2 \neq x_3)$ appartiennent au réseau cible. La question optimale correspond en conséquence à $Cl_q = (y^{\neq}_{12} \vee y^{\neq}_{23})$.

Parmi nos hypothèses de travail, nous partons du principe que l'utilisateur ne maîtrise pas les paradigmes de la programmation par contraintes.[4] En accord avec le vocabulaire

[4]C'est d'ailleurs pour cela qu'il utilise la plateforme pour modéliser son problème sous la forme d'un réseau de contraintes.

défini par D. Angluin (Angluin 2004), nous supposons par conséquent que l'utilisateur n'est pas capable de répondre à des questions d'**équivalence** : *"Ce réseau de contraintes est-il équivalent au réseau recherché?"*. Nous supposons seulement qu'il est capable de répondre à des questions d'**appartenance**, c'est à dire des questions de type : *"Cette instance est-elle une solution à votre problème?"*

Forts de cette constatation, formuler une question optimale revient à extirper une instance e_q dont Cl_q serait la clause codant son éventuel rejet. Pour cela, il faut déduire de Cl_q un réseau de contraintes P, résoudre ce dernier, prendre e_q parmi l'ensemble des solutions de P, puis enfin présenter à l'utilisateur la question d'appartenance *"e_q est-elle une solution à votre problème?"*.

4.2 Construction d'une question optimale

Dans cette partie, à l'aide de l'algorithme 1, nous présentons une formalisation du processus permettant de créer une question optimale à partir d'une clause Cl_q.

Algorithm 1: Construction d'une question optimale

Entrée: $Cl_q, E, \mathcal{X}, \mathcal{D}$
Sortie: e_q : question optimale relativement à Cl_q

$C \leftarrow \emptyset$
$Cl \leftarrow$ clause dont est issue Cl_q
$e \leftarrow$ instance dont Cl code le rejet
pour $(x_i, x_j) \in (\mathcal{X})^2$ **faire**
1 **si** (i,j) *non mis en jeu dans* Cl_q **alors** $C_{ij} \leftarrow S_{ij}^{E}$
2 **si** (i,j) *mis en jeu dans* Cl_q **alors** $C_{ij} \leftarrow S_{ij}^{\{e\}}$
$C \leftarrow C \cup C_{ij}$
$P \leftarrow (\mathcal{X}, \mathcal{D}, C)$
3 $Sol \leftarrow$`résoudre`(P)
retourner *un élément de* Sol

Le fonctionnement de l'algorithme 1 est le suivant : Si Cl est une clause de $\mathcal{K}$ codant une instance négative $e \in E^-$. Soit Cl_q une clause issue de Cl correspondant aux spécifications données dans la section précédente. Nous cherchons à produire un réseau de contraintes $P = (\mathcal{X}, \mathcal{D}, C)$ dont l'ensemble $Sol(\mathcal{X}, \mathcal{D}, C)$ des solutions est tel que $\forall e_q \in Sol(\mathcal{X}, \mathcal{D}, C)$, si e_q est rejetée par l'utilisateur, alors la clause codant son rejet correspond à la clause Cl_q.

Nous voulons pour cela que seules les contraintes codées par les littéraux de Cl_q soient potentiellement violées. En conséquence, il nous faut placer chaque contrainte C_{ij} non impliquée dans Cl_q à sa borne spécifique dans l'Espace des Versions. Nous avons donc $rel(C_{ij}) = S_{ij}^{E}$ (ligne 1). De façon symétrique, nous cherchons à ce que

toutes les contraintes impliquées dans Cl_q soient potentiellement violées. Nous forçons en conséquence (ligne 2) $rel(C_{ij}) = S_{ij}^{\{e\}}$ (i.e. borne spécifique de l'Espace des Versions lorsque e est la seule instance étudiée).

Si l'on note maintenant $C = (C_{ij}, \ldots, C_{i'j'})$ la séquence des contraintes C_{ij} ainsi identifiées, il suffit de lancer (ligne 3) une résolution de $P = (\mathcal{X}, \mathcal{D}, C)$, puis de présenter à l'utilisateur une instance $e_q \in Sol(\mathcal{X}, \mathcal{D}, C)$.

Propriété 2 (Complétude et correction de l'algorithme 1)
L'algorithme 1 est correct et complet : L'instance e_q qu'il calcule et renvoie correspond à la question optimale relativement aux données d'entrée.

Preuve. Soit Cl la clause codant pour une instance négative e de E, Cl_q la clause codant une question optimale et respectant la propriété 1, et e_q l'instance renvoyée par l'algorithme 1. Supposons que e_q soit une instance négative du problème, et notons Cl_{e_q} la clause codant son rejet. Par définition du réseau de contraintes P construit, e_q ne viole aucune des contraintes portant sur les couples (i, j) n'étant pas mis en jeu dans Cl_q (puisque ces contraintes on été placées à S_{ij}^E). Étudions maintenant les contraintes C_{ij} posées sur les couples mis en jeu dans Cl_q. Pour chacune d'elle, $C_{ij} = S_{ij}^{\{e\}}$, le tuple (v_i, v_j) est par conséquent sémantiquement équivalent au couple (v'_i, v'_j) de e. Les littéraux y_{ij}^r de Cl_{e_q} sont donc les mêmes que ceux de Cl_q, ce qui achève de démontrer que $Cl_{e_q} = Cl_q$. □

Enfin, il convient de noter que l'algorithme 1 nécessite la résolution d'un réseau de contraintes (ligne 3) afin d'en extirper une solution. La construction d'une question optimale se révèle donc être un procédé NP-complet, contrairement au cadre de l'attribut/valeur décrit par Mitchell.

Exemple 5
Afin d'illustrer le processus de construction présenté ci-dessus, nous reprenons l'étude de l'exemple 4 afin d'extirper la question optimale correspondant à la clause $Cl_q = (y_{12}^{\neq} \vee y_{23}^{\neq})$.

(x_1, x_2) et (x_2, x_3) sont les seuls couples mis en jeu dans Cl_q. En conséquence, les contraintes relatives aux autres couples sont déterminées à l'aide de $S_{34}^E = S_{45}^E = " > "$. $(x_3 > x_4)$ et $(x_4 > x_5)$ sont donc ajoutées au réseau P en cours de construction.

Nous cherchons ensuite à ce que seules les contraintes $(x_1 \neq x_2)$ et $(x_2 \neq x_3)$ soient potentiellement violées. Nous utilisons pour cela $S_{12}^{\{e\}} = S_{23}^{\{e\}} = " = "$. Nous obtenons donc finalement le réseau de contraintes $P = \{(x_1 = x_2, x_2 = x_3, x_3 > x_4, x_4 > x_5)\}$.

L'instance $e_q = (3, 3, 3, 2, 1)$ est une solution du réseau de contraintes ainsi construit. Il ne reste plus qu'à la présenter à l'utilisateur. Si ce dernier la classifie négativement, nous serons assurés qu'au moins une des deux contraintes $(x_1 \neq x_2)$ ou $(x_2 \neq x_3)$ appartient au réseau cible. Dans le cas contraire ($e_q \in E^+$), nous serons assurés que ces deux contraintes n'appartiennent pas au réseau cible.

5 Expérimentations

Afin de comparer l'impact des deux approches que nous avons proposées dans cet article, nous avons mis en place une plateforme expérimentale d'apprentissage interactif de contraintes. Pour implémenter cette dernière, notre choix s'est porté sur le langage JAVA, nous permettant ainsi d'utiliser le solveur CHOCO (Choco) pour les aspects contraintes et la librairie SAT4J (Sat4J) pour les aspects *satisfaction de formules booléennes*.

Lors de nos expérimentations, nous avons utilisé la librairie de contraintes $L = \{<, \leq, =, \geq, >, \neq\}$ et généré aléatoirement des réseaux de contraintes cibles mettant en jeu uniquement des contraintes de L. Le tableau 4 recense les résultats obtenus lors de nos expérimentations, et durant lesquelles nous avons fait varier le nombre $|\mathcal{X}|$ de variables du problème, le nombre $|\mathcal{C}|$ de contraintes appartenant au réseau cible ainsi que le nombre $\#q$ d'instances ou questions posées puis analysées par CONACQ . Les trois autres colonnes présentent les résultats obtenus après $\#q$ interactions pour les différentes techniques d'apprentissage présentées dans cet article : CONACQ *standard*, la sélection des questions et enfin l'apprentissage par questions optimales. $\#y^r_{ij}$ représente le nombre de littéraux de la base $\mathcal{K}$ non encore assignés à la fin du processus d'apprentissage, $\#sol$ le nombre de solutions pour $\mathcal{K}$, et $t(q)$ le temps moyen nécessaire à la génération d'une instance.

Le nombre $\#sol$ de solutions de la base $\mathcal{K}$ correspond au nombre de réseaux consistants avec les données d'entraînement, et donc à la taille de l'Espace des Versions. Nous pouvons par conséquent considérer que plus $\#sol$ est élevé, moins la connaissance acquise par CONACQ est précise. Par ailleurs, le nombre $\#y^r_{ij}$ de littéraux non assignés nous permet d'obtenir une borne sur le nombre de solutions : $\#sol \leq 2^{\#y^r_{ij}}$. Cette donnée est par conséquent intéressante lorsque le temps de calcul du nombre de solutions devient trop important.

Paramètres			CONACQ *standard*			*Sélection des questions*			*Questionnement optimal*		
$\|\mathcal{X}\|$	$\|\mathcal{C}\|$	$\#q$	$\#y^r_{ij}$	$\#sol$	$t(q)$	$\#y^r_{ij}$	$\#sol$	$t(q)$	$\#y^r_{ij}$	$\#sol$	$t(q)$
3	1	5	4	12	$0.10s$	3	6	$0.12s$	2	4	$0.16s$
3	2	10	6	21	$0.10s$	2	3	$0.15s$	1	2	$0.21s$
3	3	10	8	35	$0.10s$	6	24	$0.19s$	1	2	$0.32s$
5	6	10	30	–	$0.10s$	18	–	$0.33s$	4	12	$0.57s$
5	6	20	26	–	$0.10s$	7	28	$0.47s$	2	4	$0.89s$
8	10	20	62	–	$0.10s$	46	–	$0.58s$	29	–	$1.09s$
8	10	60	34	–	$0.10s$	28	–	$0.76s$	12	–	$1.72s$

Table 4: Comparaison entre les performances de CONACQ *standard*, la méthode de sélection des questions et l'apprentissage par questionnement optimal.

Les résultats de ces expérimentations nous permettent de formuler une première observation : La sélection des questions permet toujours d'obtenir des meilleurs résultats que l'apprentissage réalisé par CONACQ *standard*. De même, la connaissance acquise

par questionnement optimal est toujours plus précise que celle acquise par sélection des questions.

Nous notons par ailleurs que cette amélioration de connaissance a un impact sur le temps nécessaire à la génération d'une question : Le questionnement optimal est en effet plus lent que la méthode de sélection des questions, elle même plus lente que CONACQ *standard*. De plus, lorsque la taille du problème grandit, le temps nécessaire à CONACQ pour produire des instances aléatoires reste constant. Celui nécessaire à la sélection des questions semble quant à lui augmenter de manière linéaire, tandis que le temps de construction des questions optimales semble augmenter avec un facteur exponentiel qui pourrait s'expliquer par l'appel caché à la résolution de problèmes NP-complets.

Nous notons aussi que, pour une taille de problème donnée, le temps de génération d'une question augmente avec le nombre d'interactions, qui peut s'expliquer par le fait que plus la connaissance acquise s'affine, plus il est difficile de la consolider, et donc de générer une instance qu'on ne sait pas déjà classifier.

6 Conclusion et perspectives

Dans cet article, nous avons adapté au paradigme de l'apprentissage de réseau de contraintes les techniques d'Active Learning et d'apprentissage interactif par questions d'appartenance. Nous avons aussi dégagé une première méthode de questionnement basée sur la sélection judicieuse d'instances générées automatiquement, une approche facile à mettre en oeuvre. Cependant, notre étude montre qu'on ne maîtrise que partiellement le gain de connaissance fourni par la réponse à cette question. Nous avons ensuite caractérisé la stratégie d'interactions optimale et montré sa complexité théorique. Enfin, des résultats expérimentaux préliminaires ont permis de valider l'intérêt de ces deux approches.

En vue d'une utilisation sur des applications réelles, notre étude va continuer en étendant notre approche à d'autres biais d'apprentissage constitués de contraintes n-aires. Nous proposerons par ailleurs des alternatives à notre approche dans le cas où l'utilisateur n'arrive pas à se prononcer sur la validité d'une instance, par exemple lorsque cette dernière sera trop complexe à analyser.

References

D. Angluin *Queries revisited* Theoretical Computer Science, Volume 313, pages 175-194, Février 2004.

C. Bessière, R. Coletta, F. Koriche, B. O'Sullivan *A SAT-Based formulation of Conacq* Note Coconut 119, LIRMM – Université Montpellier II, Février 2005.

D. A. Cohn, Z. Ghahramani, M. I. Jordan *Active Learning with Statistical Models.* Journal of Artificial Intelligence Research 4, pages 129-145, Mars 1996.

R. Coletta, C. Bessière, B. O'Sullivan, E.C. Freuder, S. O'Connell, J. Quinqueton *Semi-automatic modeling by constraint acquisition.* Proceedings of CP-2003, Short paper, LNCS 2833, Springer Kinsale, Cork, Ireland., pages 812-816, Septembre 2003.

CHOCO *http://choco.sourceforge.net/* The Choco constraint programming system.

A. Legtchenko, A. Lallouet *Acquisition des contraintes ouvertes par apprentissage de solveurs.* To appear in CAP'05, Juin 2005.
T. Mitchell *Concept learning and the general-to-specific ordering.* Machine Learning, chapitre 2, pages 20-51. McGraw Hill, 1997.
SAT4J *http://www.sat4j.org/* A satisfiability library for Java.
B.D. Smith et P.S. Rosenbloom *Incremental non-backtracking focusing: A polynomially bounded generalization algorithm for version spaces.* National Conference on Artificial Intelligence (AAAI-90), pages 848-853, 1990.
S. O'Connell, B. O'Sullivan, E.C. Freuder *Strategies for interactive constraint acquisition.* Proceedings of CP-2002 Workshop on User-Interaction in Constraint Satisfaction, pages 62-76, Septembre 2002.