Apprentissage de spécifications de CSP

Matthieu Lopez*, Lionel Martin*

*LIFO, Université d'Orléans,
Batiment IIIA, Rue Léonard de Vinci, B.P. 6759, F-45067, ORLEANS Cedex 2
prenom.nom@univ-orleans.fr,
http://www.univ-orleans.fr/lifo/

1 Introduction

Les problèmes de satisfaction de contraintes (CSP) permettent de modéliser une grande variété de problèmes cependant le niveau d'expertise requis pour les modéliser est trop élevé pour des non-experts ou bien pour des informaticiens généralistes (Freuder (1997)). De nombreux travaux ont cherché à simplifier l'utilisation des CSP en proposant par exemple des langages de spécification haut-niveau ou encore l'apprentissage de modèle. Notre objectif est de proposer une manière d'obtenir automatiquement une spécification d'un CSP en partant de solutions et de non-solutions du problème. Une spécification d'un problème est une formalisation plus abstraite qu'un modèle. Considérons le problème des n-reines qui consiste à placer n reines sur un plateau de taille $n \times n$ tel qu'aucune reine n'en attaque une autre. Il existe des modèles pour les 8-reines, 12 reines, et des spécifications pour le problème général des n-reines. Nous proposons (fig. 1) un langage de spécification où `COMPARATOR` est un atome avec que des *entrées* et `VARIABLE_GENERATOR` au moins une *sortie*. Le but de ce langage n'est pas de fournir une simplification à l'utilisateur, mais plutôt de fournir une cible pour l'apprentissage.

2 Apprendre des spécifications de CSP

Nous nous sommes intéressés aux algorithmes de type *separate-and-conquer* (Furnkranz (1999)). Dans ce cas, les méthodes usuelles en ILP échoue car d'une part de l'espace de recherche est en général trop grand, d'autre part, la recherche *Top-down* est aveugle en ne considérant que le critère de couverture des exemples. Nous nous sommes donc intéressée à restreindre la taille de l'espace de recherche en générant pour un exemple graine s, plusieurs treillis de petite taille. Nous définissons les extrémités des treillis (clauses $\top$ et $\bot$) dans la suite.

Clause $\top$: Nous proposons une approche incrémentale visant à ajouter progressivement des `VARIABLE_GENERATOR` par couche. Les atomes de profondeur 1 ajoutés sont ceux n'ayant que des sorties. Puis à profondeur i, on ne peut ajouter que des atomes dont les entrées sont des variables déjà introduites. On commence par ajouter un atome pour chaque entrée d'un prédicat, puis on augmente ce chiffre [1] jusqu'à l'apprentissage d'une règle discriminante.

Clause $\bot$: En partant de la clause $\top$, nous construisons des clauses en ajoutant un maximum de `COMPARATOR` aux atomes de $\top$ tel qu'elles couvrent l'exemple s. Pour se faire, pour

1. Sauf pour les prédicats correspondant à des fonctions

rule	: :=	$\forall$ *variables* : *body* $\rightarrow$ *head*	Exemple (coloration de graphe) :
variables	: :=	*vs* $\in$ DOMAIN	$\forall X, Y \in S, \forall A, B \in C$:
		| *variables*, *variables*	$node(X) \wedge node(Y)$,
vs	: :=	VARIABLE | *vs*,*vs*	$color(X, A) \wedge color(Y, B)$
body	: :=	VARIABLE_GENERATOR	$\rightarrow A \neq B \vee \neg adj(X, Y)$
		| $\neg$ VARIABLE_GENERATOR	$\wedge$
		| *body* $\wedge$ *body*	$\forall X \in S, \forall A, B \in C$:
head	: :=	COMPARATOR | $\neg$ COMPARATOR |	$node(X) \wedge node(Y)$
		head $\vee$ *head*	$color(X, A) \wedge color(X, B)$
			$\rightarrow A = B$

FIG. 1 – *Langage pour spécifier des CSP et un exemple sur la coloration de graphe*

Benchmarks	# de règles apprises	temps (s)	taille (avg) des règles	taille (avg) des clauses $\perp$	taille avec Aleph
Coloring graph	1	0.26	13	14	26
Schooltimetable	2	5.43	15	16	33
Jobshop	4	167.74	40	57	121
N-queens	6	22067.51	127	233	489

FIG. 2 – *Résultats sur différents jeux de données issus de problèmes académiques en CSP*

chaque substitution de $\top$ vers s, nous ajoutons tous les comparateurs vrais dans s. Si la clause obtenue rejette tous les exemples négatifs, elle servira comme clause $\perp$. Pour réduire, le nombre de clauses candidates, nous utilisons un *near-miss* (Winston (1970)) de s pour cibler uniquement les substitutions pertinentes.

Recherche : Pour raffiner nos hypothèses, nous proposons d'utiliser une heuristique maximisant le nombre d'exemples couverts et minimisant le nombre de substitutions associées à la couverture d'un exemple négatif.

Expérimentations et Résultat : Nous avons testé notre technique sur différents jeux de données de CSP dont les exemples sont générés aléatoirement. Les résultats sont résumés dans la figure 2. L'algorithme réussit à chaque fois à apprendre une théorie discriminant exactement les positifs des négatifs du jeux d'apprentissage. Dans les deux premiers jeux de données, les règles apprises sont celles attendues mais dans les deux derniers certaines sont trop spécifiques à nos exemples. Pour finir, nous pouvons remarquer la taille réduite de nos clauses $\perp$ par rapport à celle construite par aleph [2] nous permettant ainsi d'utiliser raisonnablement notre heuristique coûteuse.

Références

Freuder, E. C. (1997). In pursuit of the holy grail. *Constraints 2*(1), 57–61.

Furnkranz, J. (1999). Separate-and-conquer rule learning. *Artificial Intelligence Review 13*.

Winston, P. H. (1970). Learning structural descriptions from examples. Technical report.

2. voir `http ://www.comlab.ox.ac.uk/activities/machinelearning/Aleph/aleph.html`