Finite Domain Constraint Solver Learning

Arnaud Lallouet, Thi-Bich-Hanh Dao, Andrei Legtchenko, Abdelali Ed-Dbali

Finite Domain Constraint Solver Learning

Arnaud Lallouet, Thi-Bich-Hanh Dao, Andreï Legtchenko, AbdelAli Ed-Dbali
Université d'Orléans – LIFO – BP 6759 – 45067 Orléans – France
firstname.name@lifo.univ-orleans.fr

Abstract

In this paper, we present an abstract framework for learning a finite domain constraint solver modeled by a set of operators enforcing a consistency. The behavior of the consistency to be learned is taken as the set of examples on which the learning process is applied. The best possible expression of this operator in a given language is then searched. We instantiate this framework to the learning of bound-consistency in the indexical language of Gnu-Prolog.

1 Introduction

Constraint Satisfaction Problems (or CSPs) have been widely recognized as a powerful tool to model, formulate and solve artificial intelligence problems as well as industrial ones. A common framework to address this task is the combination of search and filtering by a local consistency. Enforcing a local consistency is usually done by the scheduling of monotonic and contracting operators up to reach their greatest common fix point. The consistency is obtained as the common closure of the set of operators. This closure is efficiently computed by a chaotic iteration of the operators [Apt, 1999]. But usually, the task of finding efficient operators which actually define a consistency is considered as one of the smartest skills of the solver's implementor. The essence of the learning framework consists in considering the behavior of the operator enforcing the desired consistency as a set of examples in order to find an adequate representation of this operator in a given language.

2 Theoretical framework

Let V be a set of variables and $D = (D_X)_{X \in V}$ their finite domains. A *constraint* c is a pair (W, T) where $W \subseteq V$ is the arity of the constraints and $T \subseteq \prod_{X \in W} D_X$ is the set of solutions of c. A *CSP* is a set of constraints. For $W \subseteq V$, a *search state* s is a tuple $(s_X)_{X \in W}$ where $\forall X \in W$, $s_X \subseteq D_X$. A *singletonic* search state $(\{v_X\})_{X \in W}$ represents a single tuple. The *search space* is $S_W = \prod_{X \in W} \mathcal{P}(D_X)$. The set S_W ordered by point-wise inclusion $\subseteq$ forms a complete lattice.

A *consistency* for a constraint $c = (W, T)$ is an operator $f : S_W \to S_W$ such that:

- f is monotonic, i.e $s \subseteq s' \Rightarrow f(s) \subseteq f(s')$, in order to ensure the confluence of the reduction mechanism,
- f is contracting, i.e $\forall s \in S_W$, $f(s) \subseteq s$, in order to reduce variable's domains,
- f is correct w.r.t c, i.e $\forall s \in S_W$, every solutions of c which are present in s remain in $f(s)$,
- f represents c, i.e for every singletonic search state s which does not represent a solution of c, $f(s)$ is an empty state (at least one element of the tuple $f(s)$ is the empty set).

Operators which satisfy the first three conditions are called *pre-consistencies* for c. As an example of consistency, if we suppose that each variable domain D_X is ordered by a total ordering $\leq$ and for $A \subseteq D_X$, we denote by $[A]$ the set $\{a \in D_X | \min(A) \leq a \leq \max(A)\}$, then the bound-consistency bc_c is defined by $\forall s \in S_W, \forall X \in W, bc_c(s)_X = s_X \cap [T_X]$, with T_X the projection of T on X.

Let cs_c be the consistency to be learned. Our aim is to build a consistency f which behaves like cs_c as much as possible. Thus f must be contracting, monotonic, *correct w.r.t* cs_c ($\forall s \in S_W$, $cs_c(s) \subseteq f(s)$) and *singleton complete w.r.t* cs_c ($f(s) \subseteq cs_c(s)$ for any singletonic search state s). However, singleton completeness is difficult to get and even not always possible to express in a given language. In order to transform a pre-consistency into a consistency, let us define a consistency id_c such that $\forall s \in S_W$, $id_c(s)$ is an empty state if s is a non-solution singletonic state, and $id_c(s) = s$ otherwise. Thus $f \circ id_c$ and $id_c \circ f$ are consistencies for c if f is a pre-consistency for c. Therefore by adding id_c in the set of operators, processed by a chaotic iteration mechanism [Apt, 1999], we only need to build pre-consistencies for c. On the other hand, the correctness condition must be ensured for every $s \in S_W$ which is generally huge. We show that:

Proposition 1 *If f is a monotonic and contracting operator such that $f(s) = s$ for every singletonic state s which represents a solution of c, then f is a pre-consistency for c.*

Therefore, by considering monotonic operators, we can reduce the search space to a sample set E which is a subset of S_W and which contains all singletonic search states. Let $\mathcal{L}$ be the language in which operators are expressed and l be an operator in this language. In order to find the best possible expression, we shall be able to compare two consistencies. This

is usually done with a distance. Let d be such a distance between two consistencies. The learning problem is formulated as follows:

$$\begin{array}{rl} \texttt{minimize} & d(cs_s, l), \\ \texttt{subject to} & \forall s \in E, cs_c(s) \subseteq l(s) \subseteq s, \end{array}$$

where $E \subseteq S_W$, E contains all singletonic search states of S_W and l is a monotonic operator. Following the machine learning vocabulary, cs_c represents the example space and $\mathcal{L}$ the hypothesis space.

3 Learning indexicals

To instantiate our theoretical framework, we have to define strong language biases in order to limit the combinatorial explosion.

The first question is the language in which operators are expressed. The language of indexicals [van Hentenryck *et al.*, 1991] is chosen, motivated by the ease of integration of the user-defined indexicals in Gnu-Prolog [Diaz and Codognet, 2001]. In this language, an operator is written `X in r`, where `X` represents the domain of the variable X and `r` is an expression representing a subset of D_X. If we denote x the unary constraint representing X's domain, then the indexical represents the operator $x \leftarrow x \cap r$.

Then comes the choice of consistency. We learn the bound-consistency, since it allows to limit the example space to intervals instead of arbitrary subsets.

For each variable we learn a reduction indexical and define an indexical for id_c. The reduction indexical for X is of the form `X in minX .. maxX` where `minX`, `maxX` are in some fixed forms. In practice, we use linear, piecewise linear and rational forms. In order to the reduction indexical to be monotonic, the bound `minX` must be anti-monotonic and `maxX` monotonic. This can be ensured by syntactic conditions on the sign of the coefficients for each expression.

The indexicals for id_c could be implemented in two ways: by using Gnu-Prolog indexicals for predefined constraints in which each instance of `min` and `max` is simply replaced by `val`, or by a direct code using `val` and C operators.

As distance between two consistencies, we use the global error on the example space E. By considering that f must be correct w.r.t cs_c, this distance is $\Sigma_{s \in E} |f(s) \backslash cs_c(s)|$.

Example For lack of space, we present here one example. An user defined global constraint is defined by the following conjunction: $X - Y > 3, X - Y < 30, 3 * X - Y > 50, 5 * Y - X < 120, X < 45, Y > 5$. When treated globally as a two dimensional polyhedra, these constraints yield less indexicals than the above decomposition. On the domain $[0..49] \times [0..49]$, our system generates the following operators:

```
X in 6*min(Y)/<20+18 .. -10/<(1 + max(Y))+45
Y in 10/<(max(X)+1)+6 .. -590/<(max(X)+1)+47
```

Reparation operators are implemented from Gnu-Prolog indexicals. Here is the one for X:

```
X in val(Y)+4 .. 44 & 0 .. val(Y)+29 &
  50+val(Y)/>3+1 .. 44 & 5*val(Y)-119 .. 44
```

When trying all reductions on boxes included in $[16, 25] \times [5, 26]$, the learned operators ran in 290 ms, while the non-decomposed constraints ran in 400 ms. All tests have been done on a Pentium4, 2Ghz, 512MB running Linux.

4 Conclusion

Related work Solver learning has been first introduced by [Apt and Monfroy, 1999] in which they automatically generate a set of rules from the tuples defining a constraint. The complexity of the rules generation limits them to small finite domains such as boolean.

The system PROPMINER [Abdennadher and Rigotti, 2002; 2003] is devoted to the learning of Constraint Handling Rules [Früwirth, 1998]. The produced solver is often very readable, especially when a small number of rules are produced. While being less general in theory since we only deal with finite domains, our method works on domains and constraint arities much larger.

In an earlier paper [Dao *et al.*, 2002], we have presented an indexical learning process. We propose here two main improvements, besides a more general theoretical framework: the possibility of using only a sample of the example space while still ensuring the correctness of the learned solver and the reparation method. It follows that our system is able to handle larger constraint arity and larger domains and therefore yields a better solver.

Summary We have presented a general, language-independent framework for finite domain constraint solver learning and an instantiation to the learning of bound-consistency with Gnu-Prolog indexicals.

Acknowledgment We gratefully thank Michel Bergère and the anonymous referees for their remarks on this paper. This work is supported by french CNRS grant 2JE095.

References

[Abdennadher and Rigotti, 2002] Slim Abdennadher and Christophe Rigotti. Automatic generation of rule-based solvers for intensionally defined constraints. *International Journal on Artificial Intelligence Tools*, 11(2):283–302, 2002.

[Abdennadher and Rigotti, 2003] Slim Abdennadher and Christophe Rigotti. Automatic generation of rule-based constraint solvers over finite domains. *Transaction on Computational Logic*, 2003. accepted for publication.

[Apt and Monfroy, 1999] K. R. Apt and E. Monfroy. Automatic generation of constraint propagation algorithms for small finite domains. In Joxan Jaffar, editor, *International Conference on Principles and Practice of Constraint Programming*, volume 1713 of *LNCS*, pages 58–72. Springer, 1999.

[Apt, 1999] K. R. Apt. The essence of constraint propagation. *Theoretical Computer Science*, 221(1-2):179–210, 1999.

[Dao *et al.*, 2002] Thi Bich Hanh Dao, Arnaud Lallouet, Andrei Legtchenko, and Lionel Martin. Indexical-based solver learning. In Pascal van Hentenryck, editor, *International Conference on Principles and Practice of Constraint Programming*, volume 2470 of *LNCS*, pages 541–555. Springer, 2002.

[Diaz and Codognet, 2001] Daniel Diaz and Philippe Codognet. Design and implementation of the Gnu-Prolog system. *Journal of Functional and Logic Programming*, 2001(6), 2001.

[Früwirth, 1998] Thom Früwirth. Theory and practice of Constraint Handling Rules. *Journal of Logic Programming*, 37(1-3):95–138, 1998.

[van Hentenryck *et al.*, 1991] P. van Hentenryck, V. Saraswat, and Y. Deville. Constraint processing in cc(fd). draft, 1991.