Using Constraint Satisfaction for Learning Hypotheses in Inductive Logic Programming

Roman Barták

Charles University, Faculty of Mathematics and Physics
Malostranské nám. 2/25, 118 00 Praha 1, Czech Republic
roman.bartak@mff.cuni.cz

Ondřej Kuželka, Filip Železný

Czech Technical University, Faculty of Electrical Engineering
Technická 2, 166 27 Praha 6, Czech Republic
{kuzelon2, zelezny}@fel.cvut.cz

Introduction

Inductive logic programming (ILP) is a subfield of machine learning which uses first-order logic as a uniform representation for examples, background knowledge and hypotheses (Muggleton and De Raedt, 1994). In this paper we deal with a so called *template consistency problem*, which is one of essential tasks in ILP (Gottlob et al 1999). In particular, given learning examples and template T, we are looking for a substitution σ making $T\sigma$ consistent with the examples. Such $T\sigma$ is called a *consistent hypothesis* meaning that it entails all *positive examples* and no *negative example*. Writing variables (constants) in upper (lower) cases, we assume examples expressed as sets of ground function-free atoms, e.g. E^+ = {arc(a,b), arc(b,c), arc(c,a)}. A hypothesis is a set of atoms where all terms are variables, e.g. H = {arc(X,Y), arc(Y,Z), arc(Z,X)}. The set represents a disjunction of atoms, negation is not allowed though a negative literal can be modeled using a special atom. The hypothesis is obtained from a template by applying substitution σ which is basically unification of certain variables in the template. In this paper we propose a constraint model describing which variables in the template are unified to obtain consistent hypothesis. To check the entailment we use a form of *θ-subsumption* (Plotkin 1970) which is a decidable restriction of logical entailment. Hypothesis H *subsumes* example E, if there exists a substitution θ of variables such that $H\theta \subseteq E$. In the above example, substitution θ = {X/a, Y/b, Z/c} implies that H subsumes E^+. The requirement that a negative example E^- is not subsumed by hypothesis H means that there may not exist any substitution θ such that $H\theta \subseteq E^-$.

Constraint satisfaction techniques have been previously used in ILP, though only for subsumption checking (Maloberti, Sebag, 2004). *Constraint satisfaction* (CS) is basically a technology for solving combinatorial problems modeled by a set of variables, each of which has a finite set of possible values (domain), and a set of constraints defining allowed combinations of values for the variables. The task is to find an instantiation of variables satisfying all the constraints. The mainstream constraint satisfaction technology combines search with inference (constraint propagation). We describe here the constraint models for finding unification of variables in template and for checking consistency of the obtained hypothesis and corresponding search strategies.

Constraint Model for Unification

Assume that template T is described by a set of atoms with fresh variables, that is, each variable occurs exactly once in the template and the variables are ordered, for example T = {arc(X_1,X_2), arc(X_3,X_4), arc(X_5,X_6)}. The templates can be incrementally generated from predicates in positive examples (from smaller to larger templates). Recall that we are looking for substitution (unification) σ such that $T\sigma$ is a consistent hypothesis. Our model is based on the observation that if a set of variables is unified then we can take the variable with the smallest index to represent this set and all other variables in the set are mapped to this variable. For example, unification $X_2 = X_3$ can be represented by mapping X_3 to X_2. The proposed constraint model uses index variable I_j for each variable X_i in the template to describe the mapping. The domain of I_i is {1, …,i}. To ensure that each variable is mapped to the first variable in the set of unified variables we use a constraint $\forall i=1,\ldots,n$ *element*(I_i, [I_1,.., I_n], I_i), where *n* is the total count of variables. The semantics of *element*(X,List,Y) is as follows: Y equals to the X-th element of List. For example, [1,1,2] is not a valid list of indexes (it represents $X_1 = X_2$ and $X_2 = X_3$) as it violates *element*(2,[1,1,2],2). The correct representation of this unification should be [1,1,1]. The *element* constraints thus ensure that each set of unifications is represented by exactly one list of indexes.

As we assume the template structure given, we should ensure that no atom will disappear after unifying the variables. For example, index list [1,2,1,2,5,6] satisfies the element constraints and represents hypothesis {arc(X_1,X_2),

arc(X_1,X_2), arc(X_5,X_6)}. However, it is actually hypothesis {arc(X_1,X_2), arc(X_5,X_6)} because the first two atoms are identical. To remove this ambiguity we suggest the following constraint. We collect the tuples of index variables belonging to variables in atoms with the same name, in our example, we obtain pairs (I_1,I_2), (I_3,I_4), (I_5,I_6). For each atom and a corresponding list of variable tuples L, we post a constraint *lex*(L). This constraint ensures that variable tuples in list L must be lexicographically ordered. In our example, it means (I_1,I_2) < (I_3,I_4) < (I_5,I_6) which ensures that no atom *arc* is unified with another atom *arc* in the template (and hence no atom will disappear from T).

Constraint Model for Subsumption Checks

We realize the subsumption check using the idea proposed in (Maloberti, Sebag, 2004). The main difference of our approach is in using n-ary constraints and standard constraint satisfaction techniques instead of binary constraints and ad-hoc implementation. For a given predicate symbol *p* with arity *k* we collect all *k*-tuples of values from atoms of this predicate. Let {arc(a,b), arc(b,c), arc(c,a)} be the example then the set of tuples for predicate symbol arc/2 is {(a,b), (b,c), (c,a)}. These tuples of values define the *k*-ary tabular constraint. We post this constraint over all *k*-tuples of variables from the set {X_1,… X_n} that correspond to predicate symbol *p*. The constraint for hypothesis {arc(X_1,X_2), arc(X_3,X_4), arc(X_5,X_6)} is posted over the pairs of variables (X_1,X_2), (X_3,X_4), (X_5,X_6). Any solution of this constraint satisfaction problem defines a substitution θ such that $H\theta \subseteq E$.

The remaining question is how to connect the index variables describing unification of variables in the template with the subsumption model. For each example E_j, we plug a set $X_{j,1}$,…, $X_{j,n}$ of fresh variables into H, where *n* is the number of variables in the template and H has identical structure to template T. Note that each example requires a separate set of variables X as substitution θ can be different for each example (standardization apart). We unify these variables according to the index list I_1,…, I_n via the constraint $\forall i=1,\ldots,n$ *element*(I_i, [$X_{j,1}$,.., $X_{j,n}$], $X_{j,i}$).

Search Strategies

To find a hypothesis subsuming all positive examples, we can post the above described *element* and *lex* constraints for the index variables and for each positive example we post the tabular constraints over the set of fresh variables connected to the index variables via the *element* constraints. Instantiation of all the variables describes which variables in the hypothesis are unified and validates the subsumption check. Clearly, if we instantiate first the index variables, the subsumption checks for individual examples become independent (the models for positive examples share the index variables only). Moreover, the easiest way to instantiate the index variables is to assign value j to variable I_j. This is always possible if there are no additional restrictions and it will produce the most general hypothesis. However, there are also negative examples which may *force unification* of some variables.

We include the negative examples in the following way. First, we post all above mentioned constraints for index variables and for positive examples. Then we take a negative example E^- and perform the subsumption check using the constraint model from the previous section. Assume that for hypothesis H containing variables X_1,…, X_n, $H\theta \subseteq E^-$ turns out to hold. As the negative example is required not to be subsumed by H, we need to break the substitution θ. This can be done by selecting a pair of variables X_i and X_j such that $X_i\theta \neq X_j\theta$ and forcing unification of these variables by adding a constraint $I_i = I_j$. Frequently, there are several such pairs. If the selected pair is found wrong (that is, unifying the pair makes it impossible to subsume some positive example), we try another one. We repeat the above process while some negative example is subsumed. After this step, we obtain a set of unifications ensuring that no negative example is subsumed. Finally, we instantiate the index variables and the variables from the positive examples to validate that all positive examples are subsumed. In case of failure, we backtrack to the negative examples and try a different set of unifications. During this process, the information is propagated through the constraints, which helps in early detection of wrong decisions.

Summary

The paper suggests using constraint satisfaction techniques to solve the template consistency problem, i.e., to find suitable unifications of variables in template in ILP and to perform the incurred subsumption checks.

Acknowledgement. The research is supported by the Czech Science Foundation under the project 201/08/0509.

References

Gottlob, G., Leone, N., and Scarcello, F. 1999. On the complexity of some inductive logic programming problems. *New Generation Computing*, 17, 53-75, Omsha.

Maloberti, J. and Sebag, M. 2004. Fast Theta-Subsumption with Constraint Satisfaction Algorithms. *Machine Learning*, 55, 137–174. Kluwer Academic Publishers.

Muggleton, S. and De Raedt, L. 1994. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19, 629–679.

Plotkin, G., 1970. A note on inductive generalization. In B. Meltzer, & D. Michie (Eds.), *Machine Intelligence*, 5, 153–163. Edinburgh University Press.