Representing Biases for Inductive Logic Programming

Birgit Tausend

Fakultät Informatik, Universität Stuttgart, Breitwiesenstr. 20-22, D-70565 Stuttgart

Abstract. As each of the four main approaches to a declarative bias represention in Inductive Logic Programming (ILP), the representation by parameterized languages or by clause sets, the grammar-based and the scheme-based representation, fails in representing all language biases in ILP systems, we present a unifying representation language MILES-CTL for these biases by extending the scheme-based approach.

1 Introduction

Describing an inductive learning system at the knowledge level [Die86] requires to characterize the amount of new knowledge added by induction. As the hypotheses vary with respect to the bias, characterizing the new knowledge means to describe the bias. Among others, the bias involves the hypothesis language. For example, the language bias in Inductive Logic Programming (ILP) [Mug93] restricts the hypotheses to subsets of Horn logic. Comparing and combining several language biases in ILP is often difficult as they are uniquely not represented. None of the four approaches to bias representation in ILP described in section 3 enables the represention of all language biases in ILP.

As a unifying language for the representation of all biases is useful for the comparison and experiments with known and new biases, we show in section 4 how the scheme based approach can be extended, resulting in a representation language MILES-CTL. A scheme in MILES-CTL is called clause template and describes a set of clauses of the hypothesis language.

2 Bias in ILP Systems

Inductive logic programming aims to induce logic programs from examples. Given a set of examples $E = E^+ \cup E^-$ and the background knowledge B, the task is to find a logic program H that is necessary, i.e. $B \not\vdash E^+$, sufficient, i.e. $B \wedge H \vdash E^+$, and consistent, i.e. $B \wedge H \wedge E^- \not\vdash \Box$.

Apart from these conditions, the outcome of an ILP system depends on several additional factors, the so called bias. For example, the language bias aims to exclude unsuitable hypotheses from the hypothesis language L_H. The problem we focus on in this paper is to find a representation for the language bias in order to represent a wide range of very different restrictions, e.g. the restriction to linked, generative, constrained, constant free or function free clauses, to clauses with unique variables or a restricted number of new variables, or restrictions on connection paths established by shared variables as well as restrictions concerning the type of predicates or arguments, the determinism, the functionality and the restriction on commutative literals.

3 Approaches to the Representation of Biases

The language bias in ILP is represented either not declaratively or by one of the four main approaches, the representation by parameterized languages, clause sets, grammars or schemes.

The first approach, the representation by parameterized languages, is used to represent series of growing languages in CLINT [DR91]. The languages of a series share common features, e.g. containing only clauses with at most i new variables in the body. Some of the features have parameters varying for the language in the series. As the common features are represented by conditions that must hold for the hypothesis clauses, this approach results in a very concise description of the hypothesis language. Extension can be done by simply adding further conditions. However, fine-grained biases, e.g. argument or predicate types, that vary for small subsets of the hypothesis clauses are not easy to represent as the definition of series 3 in [DR91] shows.

The second approach is to describe the hypothesis language by an antecedent description grammar, as in GRENDEL [Coh93]. The hypothesis language includes all clauses that can be derived from the starting symbol. A derivation can be restricted by annotations added to a grammar rule. This approach shares the features of the representation by parametrized languages except that the bias is not alway easy to understand as the annotations may be arbitray Prolog clauses.

The third approach is to provide a set of schemes containing scheme variables, e.g. the predicate names of literals or their arguments. The hypothesis language includes all clauses covered by a scheme. For example, the rule models in MOBAL [KW92] are based on Horn clauses with predicate names replaced by scheme variables. Graphs as in [BJ93], [Tau93] are a closely related kind of schemes that abstract not only from the predidcate name. A clause is covered by a graph if each literal is covered by a vertex, and the edges correspond to connection paths established by shared variables.

The last representation of biases in ILP systems is the representation by clause sets [BG93]. Abstraction is achieved by using sets of alternative clauses, literals or terms. Each combination of clauses in a clause set can be used as a hypothesis. In a literal sets, each subset unless the empty set can be selected as part of the body of a hypothesis clause.

4 Representing the Language Bias in MILES-CTL

The unifying representation MILES-CTL we developed for the language bias is implemented as part of the system MILES [ST93], a modular system for simulation and experiments in ILP. MILES-CTL is based on schemes, in particular on rule models, because schemes support the representation of a wide range of bias constituents, and are close to Horn clauses. In contrast to clause sets that share the last feature, they result in a better abstraction and are easy to extend.

However, using rule models as in MOBAL [KW92] requires that a scheme for each clause with different arguments has to be available as the arguments are fixed. The set of schemes needed even increases if the hypothesis languages to be represented are not function free. To avoid the definition of a very large set of schemes, a new abstraction step is introduced in the representation language.

Similar to predicate variables, a literal scheme may contain a scheme variable for the arguments.

As a literal scheme consisting of a scheme variable $P1$ to represent the predicate and $A1$ for the arguments covers arbitrary literals, further restrictions reducing the set of covered literals have to be added to a literal scheme, e.g. the type of the predicate or the arguments, the mode, the maximum number of new variables, or some terms that have to occur in the literal. This information is represented by slots and fillers as in frames or records. Each information consists of a unqiue identifier followed by either a constant, a variable or a constrained variable, i.e. $\langle Identifier\rangle : \langle Constant\rangle$, $\langle Identifier\rangle : \langle SchemeVariable\rangle$, or $\langle Identifier\rangle : \langle SchemeVariable\rangle || Conditions$.

A list of informations on the literals results in a literal template. The information on the predicate and the arguments is mandatory, the other elements are optional. The following information may occur in a literal template.

predicate:	the name of the predicate in the literal,
arguments:	the set of arguments with their poisitions,
terms:	the set of terms that occur in the arguments,
determinate_terms:	the set of determinate terms,
variables:	the set of variables that occur in the arguments,
new_variables:	the set of new variables,
unique_variables:	yes, if the variables of the literal must be unique,
generative:	yes, if all variables of the head occur in the body literals,
arity:	the arity of the predicate in *pred*,
argument_types:	the set of argument types followed by their position,
depth:	the depth of the literal,
mode:	the mode declaration,
predicate_type:	the type of the predicate in *pred*,
commutative:	yes, if commutative variants of the literal are enabled,
solutions:	the number of solutions.

These slots are sufficient with respect to represent the constituents of biases in ILP. The representation can be exteded by adding further slots if needed.

A literal template covers a literal, if its scheme variables can be instantiated by the corresponding information on the literal such that the conditions hold, and if all constants in the template are equal to the corresponding information on the literal. For example, the literal template

$$\begin{bmatrix} predicate : P2 || (P2 \neq P1), \\ arguments : A2, \\ predicate\_type : comp \\ terms : T2 || (\{100 : \langle 2\rangle\} \subseteq T2) \\ arity : 3 \end{bmatrix}$$

covers all literals the arity of which is 3, the predicate type is *comp*, the predicate is not equal to a predicate of another literal referred to by $P1$, and the second argument is 100.

The conditions in MILES-CTL mainly restrict numeric informations to a upper or lower bound, or describe elements that have to be included in an information of the literal templates represented by sets, or require that relations to other literals hold, e.g. sharing the predicate name.

Clause templates representing a set of hypothesis clauses consists of a literal template for the head literal and a template for each body literal, i.e. $\langle HeadLitTemp\rangle \leftarrow \langle BodyLitTemp_1\rangle, \ldots, \langle BodyLitTemp_n\rangle$. For example,

$$T: \begin{bmatrix} predicate: P1, \\ arguments: A1 \end{bmatrix} \leftarrow \begin{bmatrix} predicate: P2, \\ arguments: A2, \\ predicate\_type: comp \\ new\_variables: N2 \| (|N2| \leq 1) \end{bmatrix},$$

$$\begin{bmatrix} predicate: P3, \\ arguments: A3, \\ arity: R3 \| (R3 \leq 3) \\ new\_variables: \emptyset \end{bmatrix}.$$

is a clause template covering clauses the head literal of which is arbitrary, the second literal contains at most one new variable and has the predicate type *comp*, and the third literal is arity of 3 and does not contain new variables.

A set of clause templates represents the all Horn clauses that are in the hypothesis language L_H, i.e. the language bias is represented in MILES-CTL by describing the set of hypothesis clauses.

5 Conclusion

MILES-CTL is a representation based on schemes that enables the representation of the biases used in ILP systems, and is easy to extend in order to represent further biases. As editing a large set of clause templates may result in errors and inconsistencies, we are implementing a tool to support users in this task.

Acknowledgements This work has been partially supported by ESPRIT BRA 6020 ILP. I would like to thank Katharina Morik, Irene Stahl, and Steffo Weber for comments on drafts of this paper.

References

[BG93] F. Bergadano and D. Gunetti. Learning clauses by tracing derivations. Technical report, University of Torino, 1993.

[BJ93] P. Brazdil and A. Jorge. Exploiting algorithm sketches in ILP. In *Third International Workshop on ILP* , Tech. Rep., IJS-DP-6707. J. Stefan Inst., 1993.

[Coh93] W.W. Cohen. Rapid prototyping of ILP systems using explicit bias. In *IJCAI-93 Workshop on Inductive Logic Programming*, 1993.

[Die86] T.G. Dietterich. Learning at the knowledge level. *Machine Learning*, 1(3):287–316, 1986.

[DR91] L. De Raedt. *Interactive Concept-Learning*. PhD thesis, Katholieke Universiteit Leuven, 1991.

[KW92] J.U. Kietz and S. Wrobel. Controlling the complexity of learning in logic through syntactic and task-oriented models. In S. Muggleton, editor, *Inductive Logic Programming*. Academic Press, 1992.

[Mug93] S. Muggleton. Inductive logic programming: Derivations, successes and shortcoming. In *Machine Learning: ECML-93,* . Springer, 1993.

[ST93] I. Stahl, I. and B. Tausend. MILES – a Modular Inductive Logic Programming Experimentation System. Deliverable STU1.2, ESPRIT BRA 6020: ILP. 1993.

[Tau93] B. Tausend. A unifying representation for language restrictions.In *Third International Workshop on ILP* , Tech. Rep., IJS-DP-6707. J. Stefan Inst., 1993.

This article was processed using the LaTeX macro package with LLNCS style