

TopLog: ILP using a logic program declarative bias

Stephen H. Muggleton, José C. A. Santos and Alireza Tamaddon-Nezhad

Department of Computing, Imperial College, London
{shm, jcs06, atn}@doc.ic.ac.uk

Abstract. This paper introduces a new Inductive Logic Programming (ILP) framework called Top Directed Hypothesis Derivation (TDHD). In this framework each hypothesised clause must be derivable from a given logic program called top theory (\top). The top theory can be viewed as a declarative bias which defines the hypothesis space. This replaces the metalogical mode statements which are used in many ILP systems. Firstly we present a theoretical framework for TDHD and show that a standard SLD derivation can be used to efficiently derive hypotheses from \top . Secondly, we present a prototype implementation of TDHD within a new ILP system called TopLog. Thirdly, we show that the accuracy and efficiency of TopLog, on several benchmark datasets, is comparable with the accuracy and efficiency of a state of the art ILP system like Aleph.

1 Introduction

In this paper we introduce a new approach to providing declarative bias called Top-Directed Hypothesis Derivation (TDHD). The approach extends the use of the \perp clause in Mode-Directed Inverse Entailment (MDIE) [5]. In Inverse Entailment \perp is constructed for a single, arbitrarily chosen training example. Refinement graph search is then constrained by the requirement that all hypothesised clauses considered must subsume \perp . In TDHD we further restrict the search associated with each training example by requiring that each hypothesised clause must also be entailed by a given logic program, \top .

The \top theory can be viewed as a form of first-order declarative bias which defines the hypothesis space, since each hypothesised clause must be derivable from \top . The use of the \top theory in TopLog is also comparable to grammar-based declarative biases [2, 4]. However, compared with a grammar-based declarative bias, \top has all the expressive power of a logic program, and can be efficiently reasoned with using standard logic programming techniques.

The SPECTRE system [1] employs an approach related to the use of \top . SPECTRE also relies on an overly general logic program as a starting point. However, unlike the TopLog system described in this paper, SPECTRE proceeds by successively unfolding clauses in the initial theory. TDHD is also related to Explanation-Based Generalisation (EBG) [3]. However, like SPECTRE, EBG does not make the key MDHD distinction between the \top theory and background knowledge. Moreover, EBG is viewed as a form of deductive learning, while the clauses generated by TDHD represent inductive hypotheses.

This paper is arranged as follows: Section 2 provides a theoretical framework for TDHD. This framework is then used as the basis for Hypothesis Generation in TopLog as described in Section 3. Experiments comparing speed and accuracy of TopLog and Aleph are given in Section 4. In Section 5 we conclude and discuss further work.

2 Theoretical framework

MDIE was introduced in [5] as the basis for Progol. The input to an MDIE system is the vector $S_{MDIE} = \langle M, B, E \rangle$ where M is a set of mode statements, B is a logic program representing the background knowledge and E is set of examples. M can be viewed as a set of metalogical statements used to define the hypothesis language \mathcal{L}_M . The aim of the system is to find consistent hypothesised clauses H such that for each clause $h \in H$ there is at least one positive example $e \in E$ such that $B, h \models e$.

The input to an TDHD system is the vector $S_{TDHD} = \langle NT, \top, B, E \rangle$ where NT is a set of “non-terminal” predicate symbols, \top is a logic program representing the declarative bias over the hypothesis space, B is a logic program representing the background knowledge and E is a set of examples.

The following three conditions hold for clauses in \top : (a) each clause in \top must contain at least one occurrence of an element of NT while clauses in B and E must not contain any occurrences of elements of NT , (b) any predicate appearing in the head of some clause in \top must not occur in the body of any clause in B and (c) the head of the first clause in \top is the target predicate and the head predicates for other clauses in \top must be in NT .

The aim of a TDHD system is to find a set of consistent hypothesised clauses H , containing no occurrence of NT , such that for each clause $h \in H$ there is at least one positive example $e \in E$ such that the following two conditions hold: (1) $\top \models h$ and (2) $B, h \models e$.

Due to space limitation we omit the proof of the following theorem.

Theorem 1. *Given $S_{TDHD} = \langle NT, \top, B, E \rangle$ assumptions (1) and (2) hold only if for each positive example $e \in E$ there exists an SLD refutation R of $\neg e$ from \top, B , such that R can be re-ordered to give $R' = D_h R_e$ where D_h is an SLD derivation of a hypothesis h for which (1) and (2) hold.*

According to Theorem 1, implicit hypotheses can be extracted from the refutations of a positive example $e \in E$. Let us now consider a simple example.

Example 1. Let $S_{TDHD} = \langle NT, \top, B, E \rangle$ where NT, B, e and \top are defined as follows:

$$\begin{array}{l} NT = \{\$body\} \\ B = b_1 = \text{pet(lassy)} \leftarrow \\ e = \text{nice(lassy)} \leftarrow \end{array} \quad \top = \begin{cases} \top_1 : \text{nice}(X) \leftarrow \$body(X) \\ \top_2 : \$body(X) \leftarrow \text{pet}(X) \\ \top_3 : \$body(X) \leftarrow \text{friend}(X) \end{cases}$$

Given the linear refutation $R = \langle \neg e, \top_1, \top_2, b_1 \rangle$, we now construct the re-ordered refutation $R' = D_h R_e$ where $D_h = \langle \top_1, \top_2 \rangle$ derives the clause $h = \text{nice}(X) \leftarrow \text{pet}(X)$ for which (1) and (2) hold.

3 System Description

TopLog is a prototype ILP system developed by the authors to implement the TDHD described in section 2. It is fully implemented in Prolog and is ensured to run at least in YAP, SWI and Sicstus Prolog. It is publicly available at <http://www.doc.ic.ac.uk/~jcs06> and may be freely used for academic purposes.

3.1 From mode declarations to \top theory

As the user of TopLog may not be familiar with specifying a search bias in the form of a logic program, TopLog has a module to build a general \top theory automatically from user specified mode declarations. In this way input compatibility is ensured with existing ILP systems. Below is a simplified example of user specified mode declarations and the automatically constructed \top theory.

$$\begin{array}{l} \text{modeh(mammal(+animal)).} \\ \text{modeb(has_milk(+animal)).} \\ \text{modeb(has_eggs(+animal)).} \end{array} \quad \top = \begin{cases} \top_1 : \text{mammal}(X) \leftarrow \text{\$body}(X). \\ \top_2 : \text{\$body}(X) \leftarrow \text{\%emptybody} \\ \top_3 : \text{\$body}(X) \leftarrow \text{has_milk}(X), \text{\$body}(X). \\ \top_4 : \text{\$body}(X) \leftarrow \text{has_eggs}(X), \text{\$body}(X). \end{cases}$$

Fig. 1. Mode declarations and a \top theory automatically constructed from it

The above illustrated \top theory is extremely simplified. The actual implementation has stricter control rules like: variables may only bind with others of the same type, a newly added literal must have its input variables already bound.

It is worth pointing out that the user could directly write a \top theory specific for the problem, potentially restricting the search better than the generic \top theory built automatically from the mode declarations.

3.2 TopLog Learning Algorithm

The TopLog learning algorithm consists of three major steps: 1) hypotheses derivation for each positive example, 2) coverage computation for all unique hypotheses, H , derived in previous step, 3) construct the final theory, T , as the subset of H that maximizes a given score function (e.g. compression).

Hypotheses derivation In TopLog, contrary to MDIE ILP systems, there is no construction of the bottom clause but rather an example guided generalization, deriving all hypotheses that entail a given example with respect to the background knowledge.

The hypothesis derivation procedure is composed of two distinct steps. In the first step an example is proved from the background knowledge and the \top theory. That is, the \top theory is executed having the example matching the head of its start clause (i.e. \top_1). This execution yields a proof consisting of a sequence of clauses from the \top theory and background knowledge.

For instance, using the \top theory from figure 1 and $B = b_1 = \text{has_milk}(\text{dog})$ to derive refutations for example $e = \text{mammal}(\text{dog})$, the following two refutations would be yielded: $r_1 = \langle \neg e, \top_1, \top_2 \rangle$ and $r_2 = \langle \neg e, \top_1, \top_3, b_1, \top_2 \rangle$.

In the second step, Theorem 1 is applied to r_1 and r_2 deriving, respectively, the clauses $h_1 = \text{mammal}(X)$ from $\langle \top_1, \top_2 \rangle$ and $h_2 = \text{mammal}(X) \leftarrow \text{has_milk}(X)$ from $\langle \top_1, \top_3, \top_2 \rangle$.

Coverage computation Each $h \in H$ is individually tested with all the examples (positives and negatives) to compute its coverage (i.e. the examples it entails). The positive examples that were used to derive h do not need to be tested for entailment as it is guaranteed by the hypothesis derivation procedure that h entails them.

Constructing the final theory The final theory to be constructed, T , is a subset H' of H that maximizes a given score function (e.g. compression, coverage, accuracy). Each $h \in H$ has associated the set of examples from which it was derived, Eg_h , and the set of examples which it entails, Ec_h .

The compression score function (the default) evaluates T as the weighted sum of the examples it covers (positive examples have weights > 0 and negative examples < 0) minus number of literals in T . This is the minimum description length principle and is analogous to Progol’s and Aleph’s compression measure. T is constructed using a greedy approach where at each step the hypothesis, if any, that maximizes current T' score is added to the next round.

Efficient cross-validation Prior to N fold cross-validation all possible hypotheses are derived and their coverage is computed on all examples. This is the most time consuming step. Then, examples are randomly assigned a fold and N theories are built each using a distinct combination of $N - 1$ folds as training and one fold as testing.

Hypotheses generated exclusively from examples in the test set are not eligible for the theory construction step. Also, the merit of an hypothesis is evaluated only taking into account the hypothesis coverage on examples belonging to the training folds. At the end of cross-validation, N fold average training and test accuracies and standard deviations are reported.

It is not possible to do efficient cross-validation with Aleph or Progol as no relationship exists between hypotheses and the examples that generated it.

4 Experimental Evaluation

Materials In order to empirically evaluate TopLog we used four datasets: mutagenesis [7], carcinogenesis [6], alzheimers-amine [9] and DSSTox [10] mainly because they are well known to the ILP community and are good examples of practical problems where relational knowledge is important.

In these datasets the purpose is to characterize an active molecule (for the problem at hand). It is given examples of molecules that exhibit the property (i.e. positives) and examples of molecules that do not exhibit the property (i.e. negatives) together with background knowledge.

The task of the ILP system is to find a theory that entails as most of the positive examples while entailing as few of the negative examples as possible.

Methods TopLog was compared with Aleph because Aleph is a MDIE ILP system and is also implemented in YAP Prolog. The experiments were performed on a Intel Core 2 Duo @ 2.13 GHz with 2Gb of RAM using Ubuntu Linux with kernel version 2.60.3. Aleph current version (5.0) is publicly available at [8]. The four datasets may be freely downloaded from TopLog’s webpage as well. Both TopLog and Aleph were executed on the latest YAP (version 5.1.3).

Aleph and TopLog were executed with as close as possible settings to ensure a fair test. Clause length (i.e. maximum literals in the body of an hypothesis) was set to 4 (except in DSSTox where it was set to 10), noise set to 100%, evaluation function set to compression, and number of search nodes (i.e. hypotheses) per example set to 1000.

Aleph was called both with *induce* and *induce_max* settings. The difference is that *induce* (the default), after finding a compressive clause for an example, retracts all positive examples covered by that clause while *induce_max* does not. TopLog also does not retract any example during the search and thus one should compare TopLog times with *induce_max* times rather than *induce*.

Results and Discussion The table below the time column has the running time, in CPU seconds, the ILP system took to build the model in the training data (Train column) and the total time it took to build the models for the ten folds (CV column). We distinguish between the two times to highlight the benefits of the efficient cross validation in TopLog. The accuracy column has the average (over the ten folds) percentage of correct predictions made by the model with the respective standard deviation.

Dataset	Aleph with induce			Aleph with induce_max			TopLog		
	CV Accuracy	Times		CV Accuracy	Times		CV Accuracy	Times	
Mutagenesis	77.2%±9.2%	0.4s	4s	68.6%±11.4%	2s	17s	70.2%±11.9%	0.4s	0.5s
Carcinogenesis	60.9%±8.2%	6s	54s	65.1%±8.6%	29s	245s	64.8%±6.9%	7.0s	7.4s
Alzheimers	67.2%±5.0%	5s	40s	72.6%±6.2%	18s	156s	70.4%±5.6%	17s	16s
DSSTox	70.5%±6.5%	30s	253s	71.3%±3.4%	82s	684s	71.7%±5.6%	3.4s	3.6s

Table 1. Accuracy and time comparison between Aleph and TopLog

If we only consider the training time, TopLog is always faster than Aleph with the *induce_max* setting. Comparing with the *induce* setting the advantage is not clear (e.g. in Alzheimers Aleph is much faster than TopLog but in DSSTox the reverse occurs). Considering cross validation then TopLog is clearly faster. Although this may seem a side point, built-in efficient cross validation is important in practical applications in order to assess properly the model accuracy. The accuracies are identical with none being statistically significantly different.

5 Conclusions and Future work

The key innovation of the TDHD framework is the introduction of a first order \top theory. We prove that SLD derivation can be used to efficiently derive hypotheses from \top . A new general ILP system, TopLog, is described implementing TDHD.

The empirical comparison demonstrates that the new approach is competitive, both in predictive accuracy and speed, with a state of the art system like Aleph. Below we discuss future work and some limitations of TopLog.

Parallelization Due to the way hypotheses are built in TopLog, where the construction of the set of hypotheses that covers one example is independent of the construction of the hypotheses set for the other examples, it is straightforward to parallelize TopLog main algorithm by dividing the number of examples by the number of processors available.

Sample hypothesis space Currently the hypothesis space is searched according to the \top theory automatically constructed from the user mode declarations. Although this approach seems to work well in practice, for some problems possible clusters of interesting hypotheses may not be considered.

Acknowledgments

We thank James Cussens for illuminating discussions on the TDHD framework and Vítor Santos Costa for a great Prolog system and prompt help in fixing YAP's problems allowing us to stretch it to the limits. The first author thanks the Royal Academy of Engineering and Microsoft for funding his present 5 year Research Chair. The second author was supported by a Wellcome Trust Ph.D. scholarship. The third author was supported by the BBSRC grant BB/C519670/1. We are indebted to three anonymous referees for valuable comments.

References

1. H. Boström and P. Idestam-Almquist. Specialisation of logic programs by pruning SLD-trees. In S. Wrobel, editor, *Proceedings of the Fourth Inductive Logic Programming Workshop (ILP94)*, pages 31–48, Bonn, 1994. GDM-studien Nr. 237.
2. W. Cohen. Grammatically biased learning: Learning logic programs using an explicit antecedent description language. *Artificial Intelligence*, 68:303–366, 1994.
3. S.T. Kedar-Cabelli and L.T. McCarty. Explanation-based generalization as resolution theorem proving. In P. Langley, editor, *Proc. of the Fourth Int. Workshop on Machine Learning*, pages 383–389, Los Altos, 1987. Morgan Kaufmann.
4. S. Džeroski and L. Todorovski. Discovering dynamics: From inductive logic programming to machine discovery. *Journal of Int. Inf. Systems*, 4(1):89–108, 1995.
5. S.H. Muggleton. Inverse entailment and Progol. *NGC*, 13:245–286, 1995.
6. A. Srinivasan, R.D. King S.H. Muggleton, and M. Sternberg. Carcinogenesis predictions using ILP. In *Proceedings of the Seventh International Workshop on ILP*, pages 273–287. Springer-Verlag, Berlin, 1997. LNAI 1297.
7. A. Srinivasan, S. Muggleton, R. King, and M. Sternberg. Mutagenesis: ILP experiments in a non-determinate biological domain. In S. Wrobel, editor, *Proceedings of the Fourth International Inductive Logic Programming Workshop*. Gesellschaft für Mathematik und Datenverarbeitung MBH, 1994. GMD-Studien Nr 237.
8. Ashwin Srinivasan. *The Aleph Manual*. University of Oxford, 2007.
9. R.D. King, A. Srinivasan, and M.J.E. Sternberg. Relating chemical activity to structure: an examination of ILP successes. *New Gen. Comp.*, 13:411–433, 1995.
10. A.M. Richard and C.R. Williams. Distributed structure-searchable toxicity DSSTox public database network: A proposal. *Mutation Research*, 499:27–52. 2000.