Improving the efficiency of ILP systems using an Incremental Language Level Search

Rui Camacho
LIACC, Rua do Campo Alegre, 823, 4150 Porto, Portugal
FEUP, Rua Dr Roberto Frias, 4200-465 Porto, Portugal
rcamacho@fe.up.pt
http://www.fe.up.pt/∼rcamacho

Abstract

We propose and evaluate a technique to improve the efficiency of an ILP system. The technique avoids the generation of useless hypotheses. It defines a language bias coupled with a search strategy and is called Incremental Language Level Search (ILLS). The techniques have been encoded in the ILP system IndLog. The proposal leads to substantial efficiency improvements in a set of ILP datasets referenced on the literature.

1 Introduction

Inductive Logic Programming (ILP) has achieved considerable success in a wide range of domains. It is recognised however that efficiency is a major obstacle to the use of ILP systems in applications requiring large amounts of data. Relational Data Mining applications are an example where efficiency is an important issue. In this paper we address the problem of efficiency in ILP systems by proposing a technique to improve it. Our proposal is called Incremental Language Level Search (ILLS) and is a language bias coupled with a search strategy. It is a correct language bias as defined by De Raedt (De Raedt, 1992). ILLS imposes a partition (and a meta-order) in the subsumption lattice (the search space of hypothesis). Language levels are defined based on the repetitions of predicate symbols in the clauses and since in most of the target theories of common applications the repetitions of literals in the clauses is very small or non - existent the most likely clauses are investigated first. A very useful admissible pruning rule is a corollary of this technique. Our proposal is a general technique that can be implemented in any ILP system to improve performance.

A typical ILP system carries out a search through an ordered hypothesis space. During the search hypotheses are generated and their quality estimated against the given examples. Improvements in efficiency may be obtained in avoiding to generate useless hypothesis or/and improving their evaluation.

Avoiding to generate useless hypotheses may be achieved with the specification of language bias limiting therefore the size of the search space (Nédellec et al., 1996). One my consider a procedure where there is a sequence of "space boundaries" that are increased if the system does not find an acceptable hypothesis within the current space. This technique is generally called *shift of bias* (De Raedt, 1992). Our ILLS proposal is a an example of a *shift of bias* technique. Another approach considers the study of refinement operators that allow to efficiently navigate through a hypothesis space (van der Laag and Nienhuys-Cheng, 1998).

The problem of efficient testing of candidate hypotheses has been tackled by two lines of research: improving sequential execution of an ILP system and; improving hypothesis evaluation by a parallel or a distributed execution. We first consider the sequential execution.

Sebag and Rouveirol (Srinivasan, 1999; Sebag and Rouveirol, 1997) (and later Botta *et al.* (Botta et al., 1999)) suggest the use of stochastic matching to speedup hypothesis evaluation. These approaches reduce the evaluation effort at the cost of being correct only with high probability. Another approach is proposed by Costa *et al.*. They carried out a study on exact transformations of queries when evaluating hypotheses (see (Santos Costa et al., 2000) and (Costa et al., 2002 to appear)). In (Santos Costa et al., 2000), the authors illustrated that query execution was a very high percentage of total running

time. Some exact transformations on the generated hypotheses were proposed that make them more efficient to execute on a Prolog engine.

There is a great number of common literals in a clause and its refinements and even with the refinements of its refinements. A *query pack* technique (Blockeel et al., 2002) takes advantage of the great number of shared literals among a clause and its refinements. Queries with a great lot of common computations are grouped in sets called *query packs* that are executed in a way to avoid redundant computations.

Lazy evaluation of examples (Camacho, 2000) as used in IndLog produces considerable speedups. The technique consists in avoiding or postponing the evaluation of each hypothesis against all the examples. Two kinds of lazy evaluation of examples is distinguished: negatives and positives. The rationale supporting lazy evaluation of negative examples is as follows. The purpose of evaluating a hypothesis on the negative examples is to determine if it is consistent, that is to find out if it covers less than "*noise* number" of negative examples. If the hypothesis covers more than "*noise* number" of negative examples it has to be refined (it is too general). In lazy evaluation of negative examples the system stops as soon as the clause covers "*noise* number" + 1 negative examples since that is enough to determine the lack of consistency of the clause. In the lazy evaluation of positives the system stops the evaluation of the clause as soon as it covers 1 more than the best positive cover so far. That number is enough to avoid pruning it away. The complete positive coverage is done only for consistent clauses. In most applications the negative examples overwhelm the positives and lazy evaluation of negatives may be very useful. These techniques prevent the use of negatives or positives counts in the heuristic functions but in most cases the speedups of using these techniques overcome the use of more powerful heuristics.

One last line of research followed by Dehasp and De Raedt (Dehaspe and De Raedt, 1995) and by Ohawada *et al.* (Ohwada et al., 2000), and enphasised by David Page (Page, 2000), is the parallel or distributed execution of an ILP system. It is also a very promising direction to overcome the efficiency bottleneck. Yu Wang (Wang, 2000) claim to have achieved superlinear speedup in their implementation. Graham *et al.* (Graham et al., 2000) report a linear speedup in their implementation. Work in this line of research has been essentially in distributing parts of the search space among the processors. One alternative would have a master slave architecture where each slave gets just one set of the examples partition. The master then sends each hypothesis to the slaves for evaluation on the subsets of the examples. This approach makes Data Mining applications having millions of examples amenable for ILP systems since each slave processor would process "just" a subset of the examples, requiring therefore less memory resources. A challenging approach would be to have a parallel execution of an ILP engine on top of a parallel Prolog engine. In some datasets it is the large number of examples that requires parallelization of the ILP engine whereas in some others evaluating each hypothesis is hard. In the later case a parallel Prolog engine could be very useful. Combing the workload of the ILP and Prolog engine is a very hard problem.

Most of the techniques we just referred are applicable in a parallel or distributed execution setting and therefore substantial improvements on efficiency may be gained through the combination of the results of all of these lines of research.

The evaluation of our proposal was done using the *IndLog* system (Camacho, 2000). *IndLog* is an example of an empirical ILP system, according to the classification proposed by De Raedt (1992). It is based on the *Mode Directed Inverse Entailment* Muggleton (1995). *IndLog* can handle non-ground background knowledge, can use nondeterminate predicates, uses a strongly typed language and makes use of explicit bias declarations such as mode, type and determination declarations. It has admissible pruning strategies and can handle numerical and imperfect data. To handle numerical data IndLog has the possibility of lazy evaluation of literals (Srinivasan and Camacho, 1999) and performs a user-defined cost search. To handle large amounts of data IndLog may perform lazy evaluation of the positive and negative examples. For a complete

description of *IndLog* refer to (Camacho, 2000).

The structure of the rest of the paper is as follows. In Section 2 we present the Incremental Language Level Search strategy. The experiments that empirically evaluate our proposal are presented in Section 3. The last section draws the conclusions.

2 Incremental Language Level Search

IndLog uses a particular strategy of searching through the hypothesis space that is based on a structural organization imposed on this space. We now present the hypothesis space organization by language levels and then describe the search procedure that takes advantage of such layered organization.

Let $\mathcal{D}$ be the set of definite clauses and $\mathcal{L}_i$ a set of clauses of level i. Consider a partition[1] of $\mathcal{D} = \bigcup_{i=0}^{\infty} \mathcal{L}_i$. Each subset $\mathcal{L}_i$ is called a *language level* and is defined as:

Definition 2.1 Language level

$\mathcal{L}_i = \{clause \mid$ *maximum number of occurrences of a predicate symbol in the body of clause is* $i\}$

A clause belongs to the language level i if it has a predicate symbol P that occurs i times in the body of that clause. No other predicate symbol Q (≠ P) of that clause occurs more than i times.

The maximum number of occurrences of predicate symbols in the body of the clauses determines to which partition the clause belongs. The language $\mathcal{L}_0$ is composed of definite clauses with just the head literal. The language $\mathcal{L}_1$ is composed by definite clauses whose literals in the body have no repeated predicate symbols. The language $\mathcal{L}_2$ will contain clauses whose literals in the body have a maximum number of occurrences of the same predicate symbol of two. Table 1 shows some examples of clauses classified according to language level. The language level definition is applicable to both recursive and non-recursive clauses. Clauses belonging to $\mathcal{L}_i$ will have a minimum length of i+1 (including the head literal) and a maximum of $p * i + 1$[2].

illegal(A, B, C, D, E, D)	$\in$	$\mathcal{L}_0$
multiplication(A,B,C) :- dec(A,D), multiplication(D,B,E), plus(E,B,C)	$\in$	$\mathcal{L}_1$
qSort(X,Y) :- part(X,Z,W), qSort(Z,Z1), qSort(W,W1), app(Z1, W1, Y)	$\in$	$\mathcal{L}_2$
fold(imglobuli, A) :- l(A,B), i(50,B,173), a(A,C), i(0,C,1), b(A,D), i(7,D,10)	$\in$	$\mathcal{L}_3$
p(X) :- b(X), a(X,U), a(X,Y), a(X,Z), a(X,W), b(X)	$\in$	$\mathcal{L}_4$

Table 1: Examples of clauses belonging to different language levels.

One very important property of the partitioning by language level is that all clauses in language $\mathcal{L}_{i+1}$ are subsumed by at least one clause in language $\mathcal{L}_i$ as stated in Theorem 2.1. The subsumption relation among language levels is illustrated in Figure 1.

Theorem 2.1 *Let C_j and C_k be clauses and i an integer ($i \geq 0$).*

$$\forall_{C_j} \exists_{C_k} : (C_j \in \mathcal{L}_{i+1}) \wedge (C_k \in \mathcal{L}_i) \rightarrow C_k \preceq C_j$$

Any clause belonging to language level $\mathcal{L}_{i+1}$ is θ-subsumed by at least one clause in language level $\mathcal{L}_i$.

Proof. We prove that for each clause C_{i+1} in $\mathcal{L}_{i+1}$ there is at least one clause C_i in $\mathcal{L}_i$ whose literals are a subset of C_{i+1}. Then we prove that there is a substitution that completes the subsumption relation ($C_i \preceq C_{i+1}$).

If C_{i+1} is in $\mathcal{L}_{i+1}$ then it has at least one predicate symbol that occurs i+1 times in its body. We will call it p. Since in each language level i all clauses have in their body a maximum of i occurrences of at least one predicate symbol, then necessarily $\mathcal{L}_i$ has a clause (C_i) with just i literals in the body having the predicate symbol p. Assuming that we found a correct

[1] The ∞ sign in the upper limit of the union represents a value limit smaller than infinity since definite clauses, by definition, have a finite number of literals.

[2] Where p is the maximum number of predicate symbols usable in the body of a clause.

Figure 1: Partition of the set of definite clauses by Language Level.

substitution then clause C_i is a subset of C_{i+1}. To prove the subsumption relation we have to find the assumed substitution. The empty substitution is enough to establish the subsumption relation. We just have to choose C_i such that the variables in its body literals are renaming of the literals p of C_{i+1}. With this choice, all body literals of C_i unify with the literals p of C_{i+1} and therefore $C_i\theta \subseteq C_{i+1}$.
■

The suggested partition by language level applies directly to the subsumption lattice that is traversed by the induction algorithm when constructing the hypothesis clause. The subsumption lattice is subdivided into sub-lattices, each corresponding to a particular language level. The search algorithm using the Incremental Language Level Search is summarised in Algorithm 2.1. As can be seen it involves, at each step of the main cycle, a call to a "traditional" hypothesis induction procedure. At each cycle the "traditional" hypothesis induction procedure may only generate clauses within the current language level. After each cycle the language level is increased. The cycle terminates when one of three conditions is met: i) search constraints C are no longer satisfied; ii) there has been no improvement in coverage during current level or; iii) there are no more clauses to refine because they are already refined or pruned away. If the quality of a hypothesis is its coverage[3] then we may use Theorem 2.2 to establish a stopping condition. The search may be terminated after a language level has been search through and no hypothesis was found that would bring improvements when compared to the hypotheses found in previous levels. Search constraints C typically include the maximum number of clauses constructed and the maximum length of the constructed clauses.

Theorem 2.2 *If there is no hypothesis at language level i that subsumes (covers) more positive examples than the best of positive coverage in levels j (< i), then there will be no better clauses at levels k > i.*

Proof. The proof follows from Theorem 2.1 and from the observation that a clause subsumed by another cannot cover more positive examples than the clause which subsumes it.
■

Algorithm 2.1
Incremental Language Level Search (ILLS)

input:
C /* search constraints */
D /* a dataset */
$h = \square$ /* the empty clause */
i = 0 /* language level counter */
SearchStrategy /* a search strategy for the refinement step */
$language = \mathcal{L}_0$ /* initial language is $\mathcal{L}_0$ */
improvement = true /* flag */

output: *h* /* best consistent hypothesis */

while *improvement* **and** *satisfying C* **do**

/* induce hypotheses */
h' =**genHypothesis**(*SearchStrategy,language,C,D*)

/* update */
if $(h = h')$ **then** *improvement = false*
$h = bestOf(h,h')$ /* update best hypothesis */
i = i+1
$language = \mathcal{L}_i$ /* update language level */

endwhile

Language level is not directly related to i-determinacy[4] as can be seen by the following examples: p(X) :- q(X,Y), r(Y,Z) belongs to level 1 and has i-depth of 2; p(X) :- q(X,Y), q(Y,Z) belongs to level 2 and has i-depth of 2.

An advantage of the search by language levels is that the most probable sub-lattices are searched first. In most of the target theories of common applications the repetitions of predicate symbols in the clauses is very small or non - existent. Support for this statement can

[3] As is common in most ILP systems.

[4] First introduced by Muggleton and Feng (Muggleton and Feng, 1990).

be found in the books by Stephen Muggleton (Muggleton, 1992) and the one by Nada Lavrač (Lavrač and Džeroski, 1994). Those books present some theories induced by ILP systems for *real world* applications. With the exception of some biochemical applications where the atoms positions are used, the repetition of predicate symbols in the body of the clauses rarely surpasses 2.

Varšek and Urbančič (1993) used a language restriction that specifies that only one "attribute = value" test is allowed to appear in the body of the induced rules. This restriction is equivalent to specifying a language level limit of one.

The shift of language bias was already described by De Raedt (De Raedt, 1992). However in ILLS the different languages defined are organised in a subsumption relationship order that as several advantages as the one of Theorem 2.1. In the work by De Raedt there is no such concern with the different languages defined.

3 The experiments

To empirically evaluate our proposals we run IndLog (Camacho, 2000) on several well known datasets. The experiments aim at estimating the efficiency gains when adopting the ILLS. By efficiency gain we mean a reduction in the number of useless hypotheses generated during search. This measure is machine and implementation independent. For each dataset we used in the experiments the final set of induced clause(s) are the same with and without ILLS. Therefore the accuracy of the induced theories does not change by adopting the proposed technique.

Settings

The IndLog V1.0 system was used in the experiments. The datasets used in the experiments are characterised in Table 2 and were downloaded from the Oxford[5] and York [6] Universities Machine Learning repositories.

To evaluate ILLS we run IndLog imposing a limit on the number of clauses constructed during search[7]. The *IndLog* parameter settings used in the experiments are shown in Table A of Appendix A. We then measure the percentage of constructed clauses that are above the Language Level of the target theory of each dataset. This number represents useless clauses that would never been constructed if ILLS were active. In some datasets (like mutagenesis or carcinogenesis) where we do not know the target theory, we tested several values for the maximum language level and for the maximum clause length. Table 3 shows the results of our experiments.

Dataset	positive examples	negative examples	background predicates
carcinogenesis	162	136	38
choline	663	663	32
cyclic	2	2	2
member	20	6	2
mesh	2272	223	29
multiplication	9	15	3
mutagenesis	114	57	18
pyrimidines	1394	1394	244
suramin	7	4	2
train	5	5	10

Table 2: Characterisation of the datasets used in the experiments.

The results indicate that there can be a lot to be gained if the language level of the target theory is 1. The values shown in Table 3 for language level 1 may reach 70 % (multiplication dataset) useless clauses that could be avoided. The ILLS is also recommended if the language level of the target theory is 1 (or small), clause length is large (greater than 4) and the target predicate is determined by a small number of predicates. If there is a small number of predicate symbols for the body literals and the system has to assemble a long clause it is more likely that repetitions will occur and the ILLS may be useful.

It may also be noticed that the gain of using ILLS decreases quite significantly when the target theory is in language level 2 (see the choline and the mesh datasets).

4 Conclusions

In this paper we proposed a general technique that improves the efficiency of an ILP system.

[5] URL:www.comlab.ox.ac.uk/oucl/groups/machlearn/

[6] URL: www.cs.york.ac.uk/mlg/index.html

[7] *IndLog*'s *nodes* parameter.

dataset	max length	max L.L.	useless clauses (%)
carcinogenesis	4	1	29.6
carcinogenesis	6	1	27.6
choline	4	1	9.9
choline	4	2	0.2
choline	6	1	10.2
choline	6	2	0.3
cyclic	3	1	23.8
mutagenesis	4	1	29.4
pyrimidines	4	1	9.6
pyrimidines	6	1	9.9
suramin	4	1	51.1
member	2	1	26.2
mesh	6	2	1.5
multiplication	4	1	70.7
train	4	1	17.4

Table 3: Percentage of useless clauses generated by IndLog when not using ILLS. The parameter settings are shown in Table A of Appendix A.

It is called Incremental Language Level Search (ILLS) and avoids the generation of useless hypotheses. The ILLS is a language bias coupled with a search strategy that imposes a partition (and a meta-order) on the subsumption lattice.

ILLS achieved significant reductions (up to 70.7 %) in the number of useless clauses generated by the induction algorithm.

Acknowledgments

We thank Universidade do Porto and Fundação para a Ciência e Tecnologia that fund our research. We thank the reviewers for their helpful comments.

References

Hendrik Blockeel, Luc Dehaspe, Bart Demoen, , Gerda Janssens, Jan Ramon, and Henk Vandecasteele. 2002. Improving teh efficicency of inductive logic programming through the use of query packs. *Journal of Artificial Intelligence Research*, 16:135–166.

M. Botta, A. Giordana, L. Saitta, and M. Sebag. 1999. relational learning: hard problems and phase transitions. In *Proc. of the 6th Congress AI*IA, LNAI 1792*, pages 178–189. Springer-Verlag.

R. Camacho. 2000. *Inducing Models of Human Control Skills using Machine Learning Algorithms*. Ph.D. thesis, Department of Electrical Engineering and Computation, Universidade do Porto.

Vítor Costa, Ashwin Srinivasan, Rui Camacho, Hendrik Blockeel, Bart Demoen, , Gerda Janssens, Jan Struyf, Henk Vandecasteele, and Wim Van Laer. 2002 (to appear). Query transformations for improving the efficiency of ilp systems. *Journal of Machine Learning Research*.

Luc De Raedt. 1992. *Interactive Theory Revision: An Inductive Logic Programming Approach*. Academic Press, London, UK.

L. Dehaspe and L. De Raedt. 1995. Parallel inductive logic programming. In *Proceedings of the MLnet Familiarization Workshop on Statistics, Machine Learning and Knowledge Discovery in Databases*.

J. Graham, David Page, and A. Will. 2000. Parallel inductive logic programming. In *Proceedings of teh Systems, Man and Cybernetics Conference (SMC-2000)*.

N. Lavrač and S. Džeroski. 1994. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood.

S. Muggleton and C. Feng. 1990. Efficient induction of logic programs. In *Proceedings of the 1st Conference on Algorithmic Learning Theory*, pages 368–381. Ohmsma, Tokyo, Japan.

S. Muggleton, editor. 1992. *Inductive Logic Programming*. Academic Press.

S. Muggleton. 1995. Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286.

C. Nédellec, C. Rouveirol, H. Adé, F. Bergadano, and B. Tausend. 1996. Declarative bias in ILP. In L. De Raedt, editor, *Advances in Inductive Logic Programming*, pages 82–103. IOS Press.

Hayato Ohwada, Hiroyuki Nishiyama, and Fumio Mizoguchi. 2000. Concurrent execution of optimal hypothesis search for inverse entailment. In J. Cussens and A. Frisch, editors, *Proceedings of the 10th International Conference on Inductive Logic Programming*, volume 1866 of *Lecture Notes in Artificial Intelligence*, pages 165–173. Springer-Verlag.

David Page. 2000. ILP: Just do it. In

J. Cussens and A. Frisch, editors, *Proceedings of the 10th International Conference on Inductive Logic Programming*, volume 1866 of *Lecture Notes in Artificial Intelligence*, pages 3–18. Springer-Verlag.

Vítor Santos Costa, Ashwin Srinivasan, and Rui Camacho. 2000. A note on two simple transformations for improving the efficiency of an ILP system. In J. Cussens and A. Frisch, editors, *Proceedings of the 10th International Conference on Inductive Logic Programming*, volume 1866 of *Lecture Notes in Artificial Intelligence*, pages 225–242. Springer-Verlag.

M. Sebag and C. Rouveirol. 1997. Tractable induction and classification in first-order logic via stochastic matching. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 888–893. Morgan Kaufmann.

A. Srinivasan and R.C. Camacho. 1999. Numerical reasoning with an ILP program capable of lazy evaluation and customised search. *Journal of Logic Programming*, 40(2,3):185–214.

A. Srinivasan. 1999. A study of two sampling methods for analysing large datasets with ILP. *Data Mining and Knowledge Discovery*, 3(1):95–123.

Patrick van der Laag and Shan-Hwei Nienhuys-Cheng. 1998. Completeness and properness of refinement operators. *Journal of Logic Programming*, 34(3):201–226, March.

Alen Varšek, Tanja Urbančič, and Bogdan Filipič. 1993. Genetic algorithms in controller design and tuning. *IEEE Transactions on Systems, Man and Cybernetics*, 23(5):1330–1339.

Yu Wang. 2000. Parallel inductive logic in data mining. Technical report, Department of Computing and Information Science, Queen's University, Kingston, Canada.

A IndLog Settings

Table A shows the settings of the most relevant *IndLog* parameters. Parameter *nodes* specifies the maximum number of the subsumption lattice nodes visited during the inductive search. *Noise* specifies the maximum number of negative examples a clause may cover in order to be accepted. *Min cover* specifies the minimum number of positives examples a clause has to cover in order to be accepted (avoids the generation of clauses with a very poor coverage, possibly overfitting the data). *i-depth* specifies the maximum depth of a literal with respect to the head literal of that clause. A literal that has an input argument that is connected to the head is said to be at i-depth 1. If an input argument of a literal is connected to an output argument of a literal of i-depth j then that literal is at i-depth of j+1.

dataset	max nodes	noise	i-depth	min cover
carcinogenesis	1000	15	4	5
choline	200	10	4	20
cyclic	100	0	2	1
mutagenesis	1000	5	2	10
pyrimidines	200	20	4	50
suramin	200	0	4	1
member	100	0	1	1
mesh	1000	5	5	10
multiplication	1000	0	3	1
train	1000	0	2	1

Table 4: IndLog parameter settings used in the experiments. The heuristic was set to length (breadth-first search).