Improving the efficiency of ILP systems *

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. Inductive Logic Programming (ILP) is a promising technology for knowledge extraction applications. ILP has produced intelligible solutions for a wide variety of domains where it has been applied. The ILP lack of efficiency is, however, a major impediment for its scalability to applications requiring large amounts of data. In this paper we propose a set of techniques that improve ILP systems efficiency and make then more likely to scale up to applications of knowledge extraction from large datasets. We propose and evaluate the *lazy evaluation* of examples, to improve the efficiency of ILP systems. *Lazy evaluation* is essentially a way to avoid or postpone the evaluation of the generated hypotheses (coverage tests).
The techniques were evaluated using the IndLog system on ILP datasets referenced in the literature. The proposals lead to substantial efficiency improvements and are generally applicable to any ILP system.

key words: Inductive Logic Programming, Efficiency

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 speedup the evaluation of hypotheses.

A typical ILP system carries out a search through an ordered hypothesis space. During the search hypothesis are generated and their quality estimated against the given examples. Improving the efficiency of such search procedure may be done by avoiding to generate useless hypothesis or/and improving the evaluation procedures.

* The work presented in this paper has been partially supported by Universidade do Porto, project APRIL (Project POSI/SRI/40749/2001), funds granted to *LIACC* through the *Programa de Financiamento Plurianual, Fundação para a Ciência e Tecnologia* and *Programa POSI*.

Avoiding to generate useless hypotheses may be achieved with the specification of language bias limiting therefore the size of the search space ([**?**]). An alternative approach considers the study of refinement operators that allow to efficiently navigate through a hypothesis space ([**?**]).

The problem of efficient testing of candidate hypotheses has been tackled by the following techniques. Work of a stochastic nature (see [**?**,**?**]). These reduce the evaluation effort at the cost of being correct only with high probability. A study on exact transformations of queries when evaluating hypotheses may be found in [**?**] and [**?**]. In [**?**], the authors illustrated that query execution was a very high percentage of total running time.

In this paper we address the problem of ILP system efficiency by avoiding or reducing the computational cost in the evaluation of the hypotheses using the examples. The proposed techniques are globally called *lazy evaluation* of examples and reduce considerably the amount of examples necessary to evaluate each hypothesis. The proposed techniques may be adopted in any ILP system.

The structure of the rest of the paper is as follows. In Section **??** we present the *lazy evaluation* of examples techniques. The experiments that empirically evaluate our proposals are described in Section **??**. The last section draws the conclusions.

2 Lazy evaluation of examples

Language bias may be used to avoid the generation, and therefore, the evaluation of a significant number of hypotheses. However, once an hypothesis has been generated the problem then is how to evaluate it efficiently using the available data (examples and background knowledge). This problem is specially critical if either the number of examples is large, like in Data Mining applications, or the evaluation of individual hypothesis (theorem proving effort) is hard [**?**]. The second problem has been addressed recently by means of techniques like query packs [**?**] or query transformations [**?**] and [**?**]. To handle the first situation, a probabilistic evaluation has been proposed [**?**] with the advantage of avoiding the use of all of the examples. This approach however prevents the system from having a correct measure of the hypothesis value.

We propose *lazy evaluation of examples* as a way to avoid unnecessary use of examples and therefore speed up the evaluation of each hypothesis. As the probabilistic approach we do not use all the examples to evaluate each hypothesis. Contrary to the probabilistic approach we get an exact count but only when it is absolutely necessary to do so. We can still profit from improvements due to query transformations and therefore combine the two technique for increasing speedup. We distinguish between lazy evaluation of positive examples, lazy evaluation of negative examples and positive evaluation avoidance. The techniques are now described.

Some systems like Progol [?], Aleph[1] or IndLog [?] rely on heuristics to guide the search during refinement. If the search is complete then the role of the heuristic is to improve speed. The final hypothesis should be the same. For some heurstics it is important to determine the exact number of examples covered. However, an hypothesis will only be accepted if it is consistent with the negative examples. A partial clause[2] must be specialised otherwise the search for further refinements of the partial clause terminates there. In this circumstances "lazy" evaluation of the negative examples is useful. When using the *lazy evaluation of negatives* we are only interesting in knowing if the hypothesis covers more than the allowed number of negative examples. We are not really interesting in knowing how much above the noise level the hypothesis is. Testing stops as soon as there are no more negative examples to be tested or the number of negative examples covered exceeds the allowed number for consistency (noise level). The noise level is quite often very close to zero and therefore the number of negative examples used in the test of each clause is very small. If the heuristic does not use the negative counting then this produces exactly the same results (clauses and accuracy) of the non-lazy approach but with a very significant speedup. It is also very common that the negative examples outnumber the positives. We may still use heuristics based on length and on positive cover.

IndLog also allows the positive cover to be evaluated lazily. A partial clause must be either specialised (if it covers more positives that the best consistent clause found so far) or is justifiably pruned away otherwise. When using lazy evaluation of positives it is relevant only to determine if an hypothesis covers more positives or not than the current best consistent hypothesis. We might then evaluate the positive examples just until the best cover so far is exceeded. If the best cover is exceeded we retain the hypothesis (either accept it as final is consistent or refine it otherwise) if not we may justifiably discard it. Only when accepting a consistent hypothesis we need to evaluate its exact positive cover.

We may go a bit further and simply do not evaluate the positive cover at all of partial clauses. The advantage of the first version of lazy evaluation of positives is that we may discard hypotheses that are worse than the best consistent so far, while in the second version we may keep around some partial clauses with very poor positive cover. The advantage of the second version however is that for each hypothesis we only test it on the negatives until it covers at least the noise level. Generating hypotheses is very efficient and although we may generate more hypotheses we may still gain by the increase in speed of their evaluation process. This technique may be very useful in domains where the evaluation of each hypothesis is very time-consuming.

When performing lazy evaluation of positive and negative examples we may use a breadth-first search strategy for example. This is not a too bad choice if, like in most applications, one is looking for short clauses that are very close to

[1] Available from http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/
[2] A clause that covers more than the allowed number of negative examples.

the top of the subsumption lattice.

Lazy evaluation of either negative or positive examples cannot be used when the technique called *lazy evaluation of literals* [**?**] is used. To compute the constant values all of the positive and negative coverage has to be computed exactly. It is also not applicable in data sets with positives only examples and the use of compression measure ([**?**]) or a user defined cost function like Aleph and IndLog might use ([**?**]).

3 The experiments

To empirically evaluate our proposals we run IndLog [**?**] encoded in Yap Prolog [**?**] (version 4.21) and run on a PC. For each dataset the induced clause(s) are the same with and without lazy evaluation. Therefore the accuracy of the induced theories does not change by adopting any of the proposed techniques. The datasets used in the experiments were downloaded from the Oxford [3] and York [4] Universities Machine Learning repositories.

In the experiments the IndLog settings were such that all lazy evaluations for the same dataset produce the same final theory and construct the same set of hypotheses. For the *lazy evaluation of negatives* we measured the number of theorem proving calls used to evaluate the negative examples and also the CPU time spent. For the *lazy evaluation of positives* we measured the number of theorem proving calls used to evaluate the positive examples and also the CPU time needed. For the *no positive evaluation* form of lazy evaluation we measured the number of theorem proving calls used to evaluate both the negative and positive examples, the spent CPU time and the number of nodes constructed during the search. The results are shown in Table **??**.

Dataset	lazy negs		lazy pos		no pos			
	negs calls (%)	cpu time (%)	pos calls (%)	cpu time (%)	nodes (%)	negs calls (%)	pos calls (%)	cpu time (%)
amine uptake	25	75	69	99	146	56	14	97
carcinogenesis	27	79	80	115	105	76	23	68
choline	27	74	54	92	100	44	13	46
mesh	86	99	93	99	105	125	77	81
multiplication	31	99	91	104	152	95	44	127
mutagenesis	40	99	66	101	100	48	10	98
pyrimidines	38	76	63	99	120	62	13	58
triazines	8	56	39	77	148	55	26	48

Table 1. Percentage of theorem proving calls and CPU time savings when using lazy valuation of examples (lazy / no lazy × 100%).

[3] URL: http://www.comlab.ox.ac.uk/oucl/groups/machlearn/
[4] URL:http://www.cs.york.ac.uk/mlg/index.html

Table **??** shows the percentage of the measured values when compared with the run when not using any kind of lazy evaluation. Except for the small (number of examples) datasets there is significant gain in using the lazy evaluation technique.

4 Conclusions

We proposed and evaluated a set of techniques globaly called *lazy evaluation* of examples that improve the efficiency of ILP systems. As shown by the empirical results the *lazy evaluation* may produce substantial improvements to an ILP system.

References

1. Nédellec, C. and Rouveirol, C. and Adé, H. and Bergadano, F. and Tausend, B. *Declarative Bias in ILP* ed. De Raedt, L., in Advances in Inductive Logic Programming, 82-103, 1996, IOS
2. Patrick van der Laag and Shan-Hwei Nienhuys-Cheng *Completeness and Properness of Refinement Operators* Journal of Logic Programming, 34, 3, 201–226, 1998
3. A. Srinivasan *A study of two sampling methods for analysing large datasets with ILP* Data Mining and Knowledge Discovery,vol. 3, N. 1, 95–123, 1999
4. M. Sebag and C. Rouveirol *Tractable induction and classification in first-order logic via stochastic matching* (IJCAI97) Proceedings of the 15th International Joint Conference on Artificial Intelligence, 888–893, 1997
5. Santos Costa, Vítor and Srinivasan, Ashwin and Camacho, Rui, *A note on two simple transformations for improving the efficiency of an ILP system* Proceedings of the 10th International Conference on Inductive Logic Programming, LNAI, vol 1866, ed. J. Cussens and A. Frisch, 225–242, 2000
6. Vítor Costa and Ashwin Srinivasan and Rui Camacho and Hendrik Blockeel and Bart Demoen and and Gerda Janssens and Jan Struyf and Henk Vandecasteele and Wim Van Laer *Query Transformations for Improving the Efficiency of ILP Systems* Journal of Machine Learning Research, 2002
7. M. Botta and A. Giordana and L. Saitta and M. Sebag *Relational learning: hard problems and phase transitions* Proc. of the 6th Congress AI*IA, LNAI 1792, 178–189, 1999
8. Blockeel, Hendrik and Dehaspe, Luc and Demoen, Bart and Janssens, Gerda and Ramon, Jan and Vandecasteele, Henk *Executing Query Packs in ILP* Proceedings of the 10th International Conference on Inductive Logic Programming,LNAI 1866, ed. J. Cussens and A. Frisch, 60–77, 2000
9. Muggleton, S. *Inverse Entailment and Progol* New Generation Computing, Special issue on Inductive Logic Programming, vol. 13, N. 3-4, 245–286, 1995
10. R. Camacho, *Inducing Models of Human Control Skills using Machine Learning Algorithms* PhD thesis,Department of Electrical Engineering and Computation, Universidade do Porto, 2000
11. A. Srinivasan and R.C. Camacho *Numerical reasoning with an ILP program capable of lazy evaluation and customised search* Journal of Logic Programming, vol. 40, N. 2-3, 185–214, 1999
12. Costa, V. and Damas, L. and Reis, R. and Azevedo, R., *YAP Prolog User's Manual* Universidade do Porto, 1989