April – An Inductive Logic Programming System

Nuno A. Fonseca[1], Fernando Silva[1], and Rui Camacho[2]

[1] DCC-FC & LIACC, Universidade do Porto
{nf, fds}@ncc.up.pt
[2] Faculdade de Engenharia & LIACC, Universidade do Porto
rcamacho@fe.up.pt

Abstract. Inductive Logic Programming (ILP) is a Machine Learning research field that has been quite successful in knowledge discovery in relational domains. ILP systems use a set of pre-classified examples (positive and negative) and prior knowledge to learn a theory in which positive examples succeed and the negative examples fail. In this paper we present a novel ILP system called April, capable of exploring several parallel strategies in distributed and shared memory machines.

1 Introduction

There is a strong connection between Inductive Logic Programming (ILP) and Logic Programming. ILP inherits from Logic Programming its representation formalism, its semantic orientation, and techniques. It is also common to see ILP systems implemented in Prolog. The major reason for using Prolog is that the inference mechanism implemented by the Prolog engine is fundamental to most ILP learning algorithms. ILP systems can therefore benefit from the extensive performance improvement work that has taken place for Prolog. On the other hand, ILP may be seen as challenging Prolog application since it often uses large sets of ground facts and requires storing a large search tree. Hence, ILP systems implemented in Prolog challenge the limits of Prolog systems due to their heavy usage of resources such as database accesses and memory usage.

The expressiveness of first-order logic gives ILP flexibility and understandability of the induced models. However, ILP systems suffer from significant limitations that reduce their applicability. First, most ILP systems execute in main memory, limiting their ability to process large databases. Second, ILP systems are computationally expensive, e.g., evaluating individual rules may take considerable time. On complex applications, ILP systems can take several hours, if not days, to return a model. Therefore, a major obstacle that ILP systems must overcome is efficiency.

In this paper we succinctly present the April ILP system, a generic purpose ILP system, implemented in Prolog with a modular design, that aims at being efficient, scalable, and flexible. April aims to be an efficient system by having low memory consumption and low response time. To this end it tries to combine and integrate several techniques to maximize efficiency (e.g., query transformations [1], randomized searches [2], coverage caching [3], lazy evaluation of examples [4], tabling [5], and parallelism [6, 7]). April's scalability is achieved by using relational databases to store the examples and ground background knowledge or by exploiting parallelism. April's ability to explore several parallelism approaches is the main difference to other systems. April aims to be flexible by providing a high level of customizations, for instance, allowing the modification of search method, heuristic, etc.

2 System Description

April can address predictive and descriptive ILP tasks. It addresses predictive learning tasks by constructing classification rules (using a MDIE-based algorithm [8]). It can also be applied to find association rules (using an algorithm similar to the one implemented by the Warmr system [9]). The ILP semantics used by April is the learning from entailment semantics. Therefore, when used to learn classification rules April follows the *normal semantic* of ILP [10]. The notion of coverage used in both tasks is intensional coverage.

April can be classified as an empirical (non-incremental), non-interactive, single predicate learning system, that does not perform predicate invention and is capable of handling noise.

April receives as input prior knowledge B (the background knowledge) and examples E, and induces a theory H that describes (explains) the examples. The examples E are represented as Prolog ground facts and the background knowledge as Prolog programs. The predicates in B can therefore be defined either intensionally or extensionally. The hypothesis language is a subset of the language of definite clauses. The hypothesis language is constrained through the use of *meta-language* declarations. April's meta-language includes determination declarations [11], mode and type declarations [8], background predicates' properties, pruning and constraints declarations, and facilities to change system parameters that may affect the hypotheses considered and the way that April operates.

April implements a *covering algorithm* to build a set of classification rules. The rules are found by performing a search through an ordered space of rules. April has two search strategies, namely top-down or stochastic, and different search methods (e.g., breadth-first, beam-search, randomized rapid restarts [2]). Several metrics are also available to score the rules, namely coverage, accuracy, etc.

A main feature of April is its ability to exploit parallelism in distributed or shared memory machines. April has several parallel algorithms built-in. The algorithms follow three main strategies: parallel exploration of the search space; parallel rule evaluation; or data parallelism [7]. One of the algorithms combines several strategies with pipelining and achieves super-linear speedups in a distributed memory computer [6]. A summary of the speedups observed in four applications are presented in the next section on Table 1.

April is implemented in Prolog and runs on top of the YAP Prolog system. Since April is implemented in Prolog the data is stored on Prolog's database (i.e., in memory). However, April has some extensions that allow the system to learn directly from relational databases. For the communication layer April uses LAM MPI, a high-quality open-source implementation of the Message Passing Interface (MPI) specification. LAM can be used by applications running in heterogeneous clusters or in grids, but can also be used in multiple processor computers.

3 Related Work

Since the initial concept proposal of Inductive Logic Programming, in 1990, many ILP systems have been developed[1]. April is specially related to the Aleph [12] system. Like in Aleph, April's core algorithm is based on Mode Direct Inverse Entailment (MDIE), a technique initially used in the Progol [8] system. Besides the core algorithm, April also implements many features found in Aleph. Due to this close relation, April attempts to maintain high level of compatibility with the format of the input files and parameters.

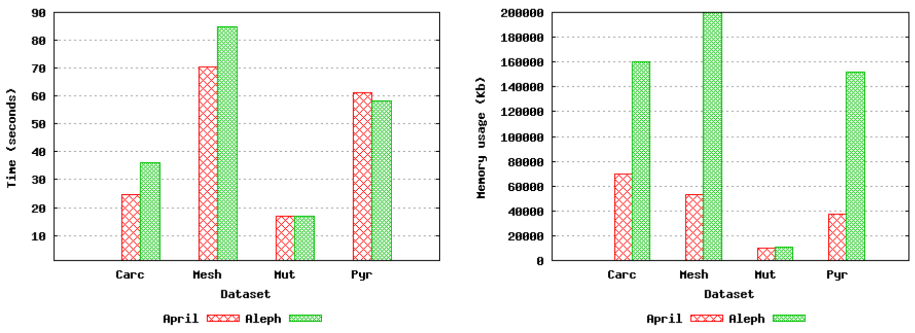

Fig. 1. Average execution time and memory consumption of April and Aleph systems in four ILP applications

A summary of an empirical comparison with Aleph is presented in Figure 1. It plots the average execution time and memory usage on four ILP applications (**Carc**inogenesis, **Mesh**, **Mut**agenesis and **Pyr**imidines) using a 10-fold cross-validation methodology. The values presented are the average of ten sequential runs to find a single rule. Figure 1 shows that sequential April is competitive against Aleph as the sequential execution time is concerned (the quality of the rules produced is also comparable). Although both systems are implemented in YAP Prolog, April's memory usage is considerably lower than Aleph. Therefore, for larger applications (number of examples or greater search spaces) April should behave better.

The main difference between April and other systems, including Aleph, resides in the ability to run in parallel using different parallel algorithms (see [6, 7]). A

[1] Srinivasan pointed out in a presentation at the ILP 2005 conference that around 100 ILP systems have been developed to date.

detailed survey of parallel ILP systems is available in [7]. Table 1 shows the speedups observed on a Beowulf cluster with one of April's parallel algorithms, the $p^2 - mdie$ parallel algorithm [6], in four ILP applications. One can observe that the speedups are good. It is important to point out that the improvements in performance obtained using the $p^2 - mdie$ parallel algorithm did not affect significantly the quality of the theories found.

Table 1. Average speedup observed for 2, 4 , 6 and 8 processors

Dataset	**2**	**4**	**6**	**8**
Carc	1.20	3.04	8.00	11.86
Mesh	1.66	4.58	6.48	7.09
Mut	3.42	6.95	6.75	8.99
Pyr	2.03	4.15	6.49	8.28

Acknowledgments. This work has been partially supported by MYDDAS (POSC/EIA/59154/2004) and funds granted to *LIACC* through the *Programa de Financiamento Plurianual, Fundação para a Ciência e Tecnologia* and *Programa POSI.*

References

1. Vítor Santos Costa, Ashwin Srinivasan, Rui Camacho, Hendrik Blockeel, Bart Demoen, Gerda Janssens, Jan Struyf, Henk Vandecasteele, and Wim Van Laer. Query transformations for improving the efficiency of ilp systems. *Journal of Machine Learning Research*, 4:465–491, 2003.
2. F. Železný, A. Srinivasan, and D. Page. Lattice-search runtime distributions may be heavy-tailed. In *Proceedings of the 12th International Conference on Inductive Logic Programming*, volume 2583 of *LNAI*, pages 333–345. Springer-Verlag, 2002.
3. James Cussens. Part-of-speech disambiguation using ilp. Technical Report PRG-TR-25-96, Oxford University Computing Laboratory, 1996.
4. Rui Camacho. As lazy as it can be. In *The Eighth Scandinavian Conference on Artificial Intelligence (SCAI'03)*, pages 47–58. Bergen, Norway, November 2003.
5. Ricardo Rocha, Nuno A. Fonseca, and Vitor Santos Costa. On Applying Tabling to Inductive Logic Programming. In *Proceedings of the 16th European Conference on Machine Learning, ECML-05*, volume 3720 of *LNAI*, pages 707–714, 2005. Springer-Verlag.
6. Nuno A. Fonseca, Fernando Silva, Vitor Santos Costa, and Rui Camacho. A pipelined data-parallel algorithm for ILP. In *Proceedings of 2005 IEEE International Conference on Cluster Computing*, 2005. IEEE.
7. Nuno A. Fonseca, Fernando Silva, and Rui Camacho. Strategies to Parallelize ILP Systems. In *Proceedings of the 15th International Conference on Inductive Logic Programming*, volume 3625 of *LNAI*, pages 136–153, 2005. Springer-Verlag.
8. S. Muggleton. Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286, 1995.
9. Luc Dehaspe and Hannu Toironen. *Relational Data Mining*, chapter Discovery of relational association rules, pages 189–208. Springer-Verlag, 2000.
10. S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19/20:629–679, 1994.
11. T. Davies and Stuart Russell. A logical approach to reasoning by analogy. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence*, pages 264–270, Los Altos, California, 1987.
12. Ashwin Srinivasan. The Aleph Manual, 2003.