Proof Tree Kernels: a Candidate Ingredient for Intelligent Optimization

Andrea Passerini Paolo Frasconi

Dipartimento di Sistemi e Informatica
Università degli Studi di Firenze

Abstract

Reactive search techniques typically rely on the search history in order to adapt heuristics to the local conformation of the search space. By viewing search history as the trace of the optimization program, we aim to apply strategies for learning from example-traces, as developed in the fields of machine learning and inductive logic programming. We believe that Proof Tree Kernels, which we recently developed as a similarity measure between program traces, should provide a useful ingredient to fully exploit search histories. By retaining much of the structural information contained in traces, they can be employed to model the space conformation in order to appropriately adapt search heuristics or develop evaluation scores for candidate moves.

1 Introduction

The problem of learning from example-traces has been addressed in a number of research fields such as automated program synthesis, inductive logic programming (ILP), intelligent optimization and machine learning. An example-trace is a sequence of steps taken by a program on a particular example input. Example-traces have been used to induce Turing machines [1] and logic programs [10], learn how to solve symbolic integration problems [6], refine logic programs to speed up their execution [12] or learn stochastic logic programs [3]. In the field of intelligent optimization, Tabu Search [4] inspects the search history in order to decide on the admissibility of the next move, thus trying to escape from local minima and better explore the search space. The diversity of these applications as well as the difficulty of the learning tasks considered illustrate the power of learning from example-traces for a wide range of applications.

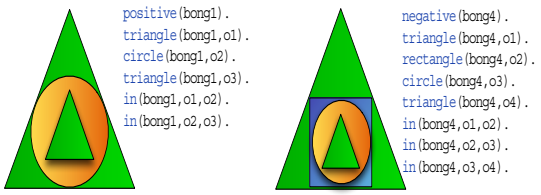
2 Proof Tree Kernels

We recently developed Kernels on Prolog Proof Trees [9] as a mean to exploit example-traces in statistical learning algorithms. The main assumption is that we are given a target program (called the *visitor*), that reflects background knowledge and that takes single examples as its input. The task consists of learning from the training set of traces obtained by executing the visitor program on each example. By developing a kernel on program traces, we are able to plug extensive symbolic domain knowledge into kernel-based statistical learning algorithms [11]. The visitor acts therefore as a knowledge-based mediator between the data and the statistical learning algorithm. The bottom line is that similar instances should produce similar traces when probed with programs that express background knowledge and examine characteristics they have in common. Visitor traces should contain additional information with respect to the final program output alone. Hence, trace kernels can be introduced with the aim of achieving a greater generality and flexibility with respect to various decomposition kernels proposed in the literature (see [11] for a review). Proof Tree Kernels have been successfully applied [9] to a variety of classification and regression tasks in the bioinformatics and chemoinformatics domains. Such extensive experimental evaluation showed the ability of the proposed approach to address the efficiency and robustness issues typical of symbolic learning approaches while retaining the expressiveness of the logic formalism.

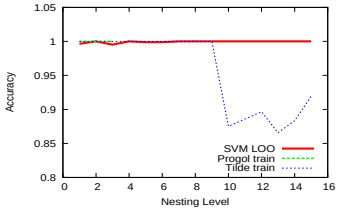
2.1 A Guiding Example

As a guiding example, consider a very simple Bongard problem in which the goal is to classify two-dimensional scenes consisting of sets of nested polygons (triangles, rectangles, and circles). In particular, we focus on the target concept defined by the pattern *triangle-X^n-triangle* for a given n, meaning that a positive example is a scene containing two triangles nested into one another with exactly n objects (possibly triangles) in between. Figure 1(a) shows a pair of examples of such scenes with their representation as Prolog facts and their classification according to the pattern for $n = 1$. A possible example of background knowledge introduces the concepts of *nesting* in containment and *polygon* as a generic object. A visitor exploiting such background knowledge, and having hints on the target concept, could be looking for two polygons contained one into the other. Note that the proof of such visitor will implicitly contain the nesting level between two polygons in the number of times the nesting concept has to be called. A very simple kernel looking at clause identifiers only can effectively learn the concept: Figure 1(b) shows the SVM leave-one-out (LOO) error compared to the empirical error of two ILP algorithms, Tilde [2] and Progol [8], for increasing nesting level of the target concept in a randomly generated dataset of scenes. Results for Progol are available for $n \leq 2$ only, as we stopped training for $n = 3$ after more than one week training time on a 3.20 GHz PENTIUM IV. Note that in order for the ILP algorithms to learn the target concept regardless of the nesting level, it would be necessary to provide

Figure 1: Experimental evaluation on the Bongard problem



(a) Graphical and Prolog facts representation of two Bongard scenes. The left and right examples are positive and negative, respectively, according to the pattern *triangle-X-triangle*



(b) Comparison between SVM leave-one-out error, Progol and Tilde empirical error in learning the *triangle-X^n-triangle* for different values of n

a more informed `inside` predicate, which explicitly contains such nesting level as one of its arguments. The ability of the kernel to extract information from the predicate proof, on the other hand, allows our method to be employed when only partial background knowledge is available, which is typically the case in real world applications.

3 Trace Kernels for Reactive Search

So far, the application of trace kernels was limited to offline supervised learning and specifically tuned for proof trees obtained from Prolog programs. Trace kernels, however, allow to introduce a far more general framework that is applicable to a variety of research areas. In the field of intelligent optimization, reactive strategies analyse past search history in order to refine the current search heuristic. It seems reasonable to view search history as the trace of a certain program (the optimization algorithm) run on a given example (the function to be optimized). Being able to identify a certain pattern in the recent trace would provide useful insights for scoring candidate next moves. For instance, escaping local minima implies being able to recognize if a certain configuration is still within its attraction basin. Traces of already visited configurations should provide useful information on the basin structure, allowing to adapt parameters of local search strategies such as Variable Neighbourhood Search [7] and Tabu Search [4] or develop a scoring function for candidate moves in a reinforcement learning fashion. Furthermore, suboptimal local minima within the search space could be early detected whenever traces are found similar to those of previously visited basins. In applying kernel over program traces to such a task, we have to cope with a number of novel problems, the first being how to collect supervision in the form of portions of past history together to their degree of success. Nonetheless, the availability of a kernel function providing a similarity measure between such traces could be a valuable ingredient in the overall mechanism.

References

[1] A.W. Biermann and R. Krishnaswamy. Constructing programs from example computations. *IEEE Transactions on Software Engineering*, 2(3):141–153, 1976.

[2] H. Blockeel and L. De Raedt. Top-down induction of first-order logical decision trees. *Artificial Intelligence*, 101(1-2):285–297, 1998.

[3] L. De Raedt, K. Kersting, and S. Torge. Towards learning stochastic logic programs from proof-banks. In *Proc. of AAAI'05*, pages 752–757, 2005.

[4] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers: Boston, 1998.

[5] C. Goller. *A Connectionist Approach for Learning Search-Control Heuristics for Automated Deduction Systems*. PhD thesis, Technical University Munich, Computer Science, 1997.

[6] T. M. Mitchell, P. E. Utgoff, and R. Banerji. Learning by experimentation: Acquiring and refining problem-solving heuristics. In *Machine learning: An artificial intelligence approach*, volume 1, pages 163–190. Morgan Kaufmann, 1983.

[7] N. Mladenović and P. Hansen. Variable neighborhood search. *Comput. Oper. Res.*, 24(11):1097–1100, 1997.

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

[9] A. Passerini, P. Frasconi, and L. De Raedt. Kernels on prolog proof trees: Statistical learning in the ILP setting. *Journal of Machine Learning Research*, 7:307–342, 2006.

[10] E.Y. Shapiro. *Algorithmic program debugging*. MIT Press, 1983.

[11] J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[12] J. M. Zelle and R. J. Mooney. Combining FOIL and EBG to speed-up logic programs. In *Proc. of IJCAI'93*, pages 1106–1111, 1993.