Revising Probabilistic Prolog Programs

L. De Raedt, K. Kersting, A. Kimmig, K. Revoredo*, and H. Toivonen**

Institut für Informatik, Albert-Ludwigs-Universität,
Georges-Köhler-Allee, Gebäude 079, D-79110, Freiburg im Breisgau, Germany
{deraedt,kersting,kimmig,revoredo}@informatik.uni-freiburg.de
hannu.toivonen@cs.helsinki.fi

Abstract. In a recently submitted paper [1], the ProbLog (probabilistic prolog) language has been introduced and various algorithms have been developed for solving and approximating ProbLog queries. Here, we define and study the problem of revising ProbLog theories from examples.

1 ProbLog: Probabilistic Prolog

A ProbLog program consists – as Prolog – of a set of definite clauses. However, in ProbLog every clause c_i is labeled with the probability p_i that it is true.

Example 1. Within bibliographic data analysis, the similarity structure among items can improve information retrieval results. Consider a collection of papers $\{\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}\}$ and some pairwise similarities $\texttt{similar}(\texttt{a}, \texttt{b})$, e.g., based on key word analysis. Two items $\texttt{X}$ and $\texttt{Y}$ are $\texttt{related}(\texttt{X}, \texttt{Y})$ if they are similar (such as $\texttt{a}$ and $\texttt{c}$) or if $\texttt{X}$ is similar to some item $\texttt{Z}$ which is related to $\texttt{Y}$. Uncertainty in the data and in the inference can elegantly be represented by the attached probabilities:

$1.0 : \texttt{related}(\texttt{X}, \texttt{Y}) \ : - \texttt{similar}(\texttt{X}, \texttt{Y}).$
$0.8 : \texttt{related}(\texttt{X}, \texttt{Y}) \ : - \texttt{similar}(\texttt{X}, \texttt{Z}), \quad \texttt{related}(\texttt{Z}, \texttt{Y}).$
$0.9 : \texttt{similar}(\texttt{a}, \texttt{c}).\ 0.7 : \texttt{similar}(\texttt{c}, \texttt{b}).\ 0.6 : \texttt{similar}(\texttt{c}, \texttt{d}).\ 0.9 : \texttt{similar}(\texttt{d}, \texttt{b}).$

A ProbLog program $T = \{p_1 : c_1, \cdots, p_n : c_n\}$ now defines a probability distribution over logic programs $L \subseteq L_T = \{c_1, \cdots, c_n\}$ in the following way:

$$P(L|T) = \prod_{c_i \in L} p_i \prod_{c_i \in L_T \setminus L} (1 - p_i).$$

Unlike in Prolog, where one is typically interested in determining whether a query succeeds or fails, in ProbLog one is interested in computing the probability that it succeeds. The *success probability* $P(q|T)$ of a query q in a ProbLog program T is defined by $P(q|T) = \sum_{L \subseteq L_T} P(q, L|T) = \sum_{L \subseteq L_T} P(q|L) \cdot P(L|T)$ with

$$P(q|L) = \begin{cases} 1 & \exists \theta : L \models q\theta \\ 0 & \text{otherwise.} \end{cases}$$

In other words, the success probability of query q corresponds to the probability that the query q has a proof, given the distribution over logic programs.

* Financially supported by Brazilian Research Council, CNPq - Brazil
** Also at University of Helsinki, Finland; supported by the Humboldt foundation

Example 2. There are two proofs of `related(d, b)`, obtained using either the base case and one fact, or the recursive case and two facts. One way of computing the total probability is to disjoin the proofs first and then sum their probabilities. For instance, by excluding `similar(d, b)` in the formula for the second proof the probability of `related(b, d)` is obtained by $1.0 \cdot 0.9 + 0.8 \cdot 0.7 \cdot 0.6 \cdot (1 - 0.9) = 0.9336$. Similar, the probability of `related(a, b)` is 0.62064.

Reference [1] proposes and evaluates various algorithms for computing and approximating the success probability of ProbLog queries, whose evaluation is computationally hard. The authors of [1] tackle this problem by employing a reduction to the computation of the probability of a monotone DNF formula and the use of binary decision diagrams (BDDs). They also apply ProbLog to biological network analysis problems. Other interesting application areas are hypertext and web mining, communication networks, and related domains.

2 Revising ProbLog Theories

Large ProbLog theories can be obtained automatically in many of the domains mentioned above, e.g., by statistical similarity, relevance or link analysis. Unfortunately, large theories are hard to utilize, both computationally and by end users, and there is need to revise them to smaller ones. Furthermore, new information can often improve the quality of an initial ProbLog theory, and also guide in reducing its size. For instance in our bibliographic example, user feedback might have revealed that items `a, b` and `c, d` are related but `d, b` are actually not. Thus, the initial theory in Example 1 should be revised.

The present paper introduces the revision problem for ProbLog theories:

Definition 1. Given *a ProbLog theory S (a set of ProbLog clauses), a set of positive and negative examples P and N in the form of ground goals, a constant $k \in \mathbb{N}$,* **find** *a theory $T \subseteq S$ of size at most k (i.e. $|T| \leq k$) that maximizes the likelihood $L(E|T) = \prod_{e \in E} L(e|T)$ of examples $E = P \cup N$ where*

$$L(e|T) = \begin{cases} P(e|T) & \text{if } e \in P \\ 1 - P(e|T) & \text{if } e \in N \end{cases} \tag{1}$$

In the ProbLog theory revision problem, we are interested in finding a small number of clauses from T that maximizes the likelihood of the data. Here a ProbLog theory T is used to determine a relative class distribution: it gives the probability $P(e|T)$ that any given example e is positive. (This is subtly different from specifying the distribution of (positive) examples.) The examples are assumed to be mutually independent, so the total likelihood is obtained as a simple product. For an optimal ProbLog theory T, the probability of the positives is as close to 1 as possible, and for the negatives as close to 0 as possible. However, because we want to allow misclassifications, but with a high cost, in order to avoid overfitting, to effectively handle noisy data, and to obtain smaller theories, one has to slightly redefine $P(e|T)$ in Equation (1), for instance as

$$\hat{P}(e|T) = \max\big(\min[1 - \epsilon, P(e|T)], \epsilon\big) \text{ for some } \epsilon > 0.$$

This avoids the possibility that the likelihood function becomes 0, e.g., when a positive example is not covered by the theory at all.

It is now instructive to look at specific instances of this problem:

- $k > |S|$: find the maximum likelihood theory. Closely corresponds to traditional theory revision in ILP; however, in the current ProbLog setting, only deletions of clauses are allowed as operations on the theory.
- $k < |S|$: theory compression.
- $k < |S|$ and $|N| = 0$: find the k clauses from S that contribute the most to the success probabilities of the positives.

3 The ProbLog Theory Revision Algorithm

The ProbLog theory revision algorithm performs a *greedy* search in the space of subsets of S. In each step, the algorithm finds the clause whose deletion results in the best likelihood score, and then deletes it. This process is continued until both $|T| \leq k$ *and* deleting further clauses does not improve the likelihood.

Example 3. Reconsider our `related` theory in Example 1 and assume that the user feedback revealed two positive examples `related(a,b)` and `related(c,d)`, and one negative example `related(d,b)`. With $\epsilon = 0.05$, their initial likelihood is 0.033. The greedy approach first deletes 0.9 : `similar(d,b)` and thereby increases the likelihood to 0.2008. The probability of the positive example `related(a,b)` is now 0.504 (was 0.62064), and that of the negative example `related(d,b)` is 0.336 (was 0.9336).

An important computational optimization is that the BDDs are reused. Their costly construction has to be done only once in the very first iteration; later on only the truth values of variables in the existing BDD are manipulated.

Experimental results on revising a large, real-world ProbLog theory for link mining are promising.

4 Conclusion

We have introduced a new type of theory revision problem involving probabilistic theories and sketched an algorithm for solving it. The problem setting is related to probabilistic ILP approaches such as Sato's PRISM and Muggleton's SLPs, which – if at all – have focused on learning theories from scratch. Only Revoredo *et al.* considered revision of BLPs. The revision problem as introduced here is closely related to the traditional ILP one but employs probabilistic principles to guide the search. Furthermore, by using the constant k it is possible to influence the degree of compression that is desired. An important question for further research is concerned with allowing for other operations than deletions of clauses.

References

1. Anonym. ProbLog: A probabilistic Prolog and its application in link discovery, submitted, 2006.