Learning the Parameters of Probabilistic Logic Programs from Interpretations

Bernd Gutmann, Ingo Thon, and Luc De Raedt

Department of Computer Science, Katholieke Universiteit Leuven
Celestijnenlaan 200A, POBox 2402, 3001 Heverlee, Belgium
firstname.lastname@cs.kuleuven.be

Abstract. ProbLog is a recently introduced probabilistic extension of the logic programming language Prolog, in which facts can be annotated with the probability that they hold. The advantage of this probabilistic language is that it naturally expresses a generative process over interpretations using a declarative model. Interpretations are relational descriptions or possible worlds. This paper introduces a novel parameter estimation algorithm LFI-ProbLog for learning ProbLog programs from partial interpretations. The algorithm is essentially a Soft-EM algorithm. It constructs a propositional logic formula for each interpretation that is used to estimate the marginals of the probabilistic parameters. The LFI-ProbLog algorithm has been experimentally evaluated on a number of data sets that justifies the approach and shows its effectiveness.

1 Introduction

Statistical relational learning [12] and probabilistic logic learning [5,7] have contributed various representations and learning schemes. Popular approaches include BLPs [15], ICL [18], Markov Logic [19], PRISM [22], PRMs [11], and ProbLog [6,13]. These approaches differ not only in the underlying representations but also in the learning settings they employ.

For learning knowledge-based model construction approaches (KBMC), such as Markov Logic, PRMs, and BLPs, one normally uses relational state descriptions as training examples. This setting is also known as *learning from interpretations*. For training probabilistic programming languages one typically uses *learning from entailment* [7,8]. PRISM and ProbLog, for instance, are probabilistic logic programming languages that are based on Sato's distribution semantics [21]. They use training examples in form of labeled facts where the labels are either the truth values of these facts or target probabilities.

In the learning from entailment setting, one usually starts from observations for a *single target* predicate. In the learning from interpretations setting, however, the observations specify the value for some of the random variables in a state-description. Probabilistic grammars and graphical models are illustrative examples for each setting. Probabilistic grammars are trained on examples in the form of sentences. Each training example states that a particular sentence was derived or not, but it does not explain *how* it was derived. In contrast, Bayesian

D. Gunopulos et al. (Eds.): ECML PKDD 2011, Part I, LNAI 6911, pp. 581–596, 2011.

networks are typically trained on partial or complete state descriptions, which specify the value for some random variables in the network. This also implies that training examples for Bayesian networks can contain much more information. These differences in learning settings also explain why the KBMC and PLP approaches have been applied on different kinds of data sets and applications. Entity resolution and link prediction are examples for domains where KBMC has been successfully applied. This paper aims at bridging the gap between these two types of approaches to learning. We study how the parameters of ProbLog programs can be learned from partial interpretations. The **key contribution** of the paper is a novel algorithm, called LFI-ProbLog, that is used for learning ProbLog programs from partial interpretations. We thoroughly evaluated the algorithm on various standard benchmark problems. LFI-ProbLog is freely available as part of the ProbLog system at `http://dtai.cs.kuleuven.be/problog/` and within YAP Prolog.

The paper is organized as follows: In Section 2, we review logic programming concepts as well as the probabilistic programming language ProbLog. Section 3 formalizes the problem of learning the parameters of ProbLog programs from interpretations. Section 4 introduces LFI-ProbLog. We report on experimental results in Section 5. Before concluding, we discuss related work in Section 6.

2 Probabilistic Logic Programming Concepts

We start by reviewing the main concepts underlying ProbLog.

An atom is an expression of the form $q(t_1, \ldots, t_k)$ where q is a predicate of arity k and the t_i terms. A term is a variable, a constant, or a functor applied to terms. Definite clauses are universally quantified expressions of the form h :- $b_1, \ldots, b_n$ where h and the b_i are atoms. A fact is a clause without a body. A substitution θ is an expression of the form $\{V_1/t_1, \ldots, V_m/t_m\}$ where the V_i are different variables and the t_i are terms. Applying a substitution θ to an expression e yields the instantiated expression $e\theta$ where all variables V_i in e are being simultaneously replaced by their corresponding terms t_i in θ. An expression is called *ground*, if it does not contain variables. The semantics of a set of definite clauses is given by its least Herbrand model, the set of all ground facts entailed by the theory. A set of definite clauses is called *tight* if the dependency graph if acyclic. An h atom depends on an atom b, if b occurs in a clause with head h.

A ProbLog theory (or program) $\mathcal{T}$ consists of a set of labeled facts $\mathcal{F}$ and a set of definite clauses $\mathcal{BK}$ that express the background knowledge. As the semantics of ProbLog is based on the distribution semantics, we require that every atom fulfills the *finite support condition*. This means that the SLD-tree for each ground atom is finite. The facts $p_n :: f_n$ in $\mathcal{F}$ are annotated with the probability p_n that $f_n\theta$ is true for all substitutions θ grounding f_n. The resulting facts $f_n\theta$ are called *atomic choices* [18] and represent the elementary random events; they are assumed to be mutually independent. Each non-ground probabilistic fact represents a kind of template for random variables. Given a

finite[1] number of possible substitutions $\{\theta_{n,1}, \ldots \theta_{n,K_n}\}$ for each probabilistic fact $p_n :: f_n$, a ProbLog program $\mathcal{T} = \{p_1 :: f_1, \cdots, p_N :: f_N\} \cup \mathcal{BK}$ defines a probability distribution over *total choices* L (the random events), where $L \subseteq L_\mathcal{T} = \{f_1\theta_{1,1}, \ldots f_1\theta_{1,K_1}, \ldots, f_N\theta_{N,1}, \ldots, f_N\theta_{N,K_N}\}$.

$$\mathrm{P}(L|\mathcal{T}) = \prod_{f_n\theta_{n,k} \in L} p_n \prod_{f_n\theta_{n,k} \in L_\mathcal{T} \setminus L} (1 - p_n).$$

The following ProbLog theory states that there is a burglary with probability 0.1, an earthquake with probability 0.2 and if either of them occurs the alarm will go off. If the alarm goes off, a person `X` will be notified and will therefore call with the probability of `al(X)`, that is, 0.7.

$$\mathcal{F} = \{\texttt{0.1} :: \texttt{burglary},\ \texttt{0.2} :: \texttt{earthquake},\ \texttt{0.7} :: \texttt{al(X)}\}$$
$$\mathcal{BK} = \{\texttt{person(mary)}.\ ,\quad \texttt{person(john)}.\ ,\quad \texttt{alarm :- burglary; earthquake}.\ ,$$
$$\texttt{calls(X) :- person(X), alarm, al(X)}.\}$$

The set of atomic choices in this program is {`al(mary)`, `al(john)`, `burglary`, and `earthquake`} and each total choice is a subset of this set. Each total choice L combined with the background knowledge $\mathcal{BK}$ defines a Prolog program. Consequently, the probability distribution at the level of atomic choices also induces a probability distribution over possible definite clause programs of the form $L \cup \mathcal{BK}$. Furthermore, each such program has a unique least Herbrand interpretation, which is the set of all the ground facts that are entailed by the program representing a *possible world*, e.g., for the total choice`burglary` the interpretation {`burglary`, `alarm`, `person(john)`, `person(mary)`} the probability distribution at the level of total choices also induces a probability distribution at the level of possible worlds. The probability $P_w(I)$ of this interpretation is $0.1 \times (1 - 0.2) \times (1 - 0.7)^2$. We define the *success probability* of a query q as

$$\mathrm{P}_s(q|\mathcal{T}) = \sum_{\substack{L \subseteq L_\mathcal{T} \\ L \cup \mathcal{BK} \models q}} \mathrm{P}(L|\mathcal{T}) = \sum_{L \subseteq L_\mathcal{T}} \delta(q, \mathcal{BK} \cup L) \cdot \mathrm{P}(L|\mathcal{T}) \qquad (1)$$

where $\delta(q, \mathcal{BK} \cup L) = 1$ if there exists a θ such that $\mathcal{BK} \cup L \models q\theta$, and 0 otherwise. It can be shown that the success probability corresponds to the probability that the query is true in a randomly selected possible world (according to P_w). The success probability $\mathrm{P}_s($`calls(john)`$|\mathcal{T})$ is 0.196. Observe that ProbLog programs do *not* represent a generative model at the level of the individual facts or predicates. Indeed, it is not the case that the sum of the probabilities of the facts for a given predicate (here `calls/1`) must equal 1:

$$\mathrm{P}_s(\texttt{calls(X)}|\mathcal{T}) \neq \mathrm{P}_s(\texttt{calls(john)}|\mathcal{T}) + \mathrm{P}_s(\texttt{calls(mary)}|\mathcal{T}) \neq 1\ .$$

So, the predicates do *not* encode a probability distribution over their instances. This differs from probabilistic grammars and their extensions such as stochastic logic programs [4], where each predicate or non-terminal defines a probability distribution over its instances, which enables these approaches to sample instances

[1] Throughout the paper, we shall assume that $\mathcal{F}$ is finite, see [21] for the infinite case.

from a specific target predicate. Such approaches realize a generative process at the level of individual *predicates*. Samples taken from a single predicate can then be used as examples for learning the probability distribution governing the predicate. In the literature this setting is known as learning from entailment. Sato and Kameya's well-known learning algorithm for PRISM [22] also assumes that there is a generative process at the level of a single predicate and it is therefore not applicable to learning from interpretations.

While the ProbLog semantics does not encode a generative process at the level of individual predicates, it does encode one at the level of interpretations. This process has been described above; it basically follows from the fact that each total choice generates a unique possible world through its least Herbrand interpretation. Therefore, it is much more natural to learn from interpretations in ProbLog; this is akin to typical KBMC approaches.

A partial interpretation I specifies for some (but not all) atoms the truth value. We represent partial interpretations as $I = (I^+, I^-)$ where I^+ contains all true atoms and I^- all false atoms. The probability of a partial interpretation is the sum of the probabilities of all possible worlds consistent with the known atoms. This is the success probability of the query $(\bigwedge_{a_j \in I^+} a_j) \wedge (\bigwedge_{a_j \in I^-} \neg a_j)$. The probability of the following partial interpretation in the Alarm domain

$$I^+ = \{\texttt{person(mary)}, \texttt{person(john)}, \texttt{burglary}, \texttt{alarm}, \texttt{al(john)}, \texttt{calls(john)}\}$$
$$I^- = \{\texttt{calls(mary)}, \texttt{al(mary)}\}$$

is $P_w((I^+, I^-)) = \big(0.1 \times 0.7\big) \times \big((1 - 0.7)\big) \times \big((0.2 + (1 - 0.2))\big)$.

3 Learning from Interpretations

Learning from (possibly partial) interpretations is a common setting in statistical relational learning that has not yet been studied in its full generality for probabilistic programming languages. In a generative setting, one is typically interested in the maximum likelihood parameters given the training data. This can be formalized as follows.

Definition 1 (Max-Likelihood Parameter Estimation). ***Given*** *a ProbLog program $\mathcal{T}(\mathbf{p})$ containing the probabilistic facts $\mathcal{F}$ with* unknown *parameters $\mathbf{p} = \langle p_1, ..., p_N \rangle$ and background knowledge $\mathcal{BK}$, and a set of (possibly partial) interpretations $\mathcal{D} = \{I_1, \ldots, I_M\}$ (the training examples).* ***Find*** *maximum likelihood probabilities $\widehat{\mathbf{p}} = \langle \widehat{p_1}, \ldots, \widehat{p_N} \rangle$ such that*

$$\widehat{\mathbf{p}} = \arg\max_{\mathbf{p}} P(\mathcal{D}|\mathcal{T}(\mathbf{p})) = \arg\max_{\mathbf{p}} \prod_{m=1}^{M} P_w(I_m|\mathcal{T}(\mathbf{p}))$$

Thus, we are given a ProbLog program and a set of partial interpretations and the goal is to find the maximum likelihood parameters. One has to consider two cases when computing $\widehat{\mathbf{p}}$. For complete interpretations where everything is observable, one can obtain $\widehat{\mathbf{p}}$ by counting (cf. Sect. 3.1). In the more complex case of partial interpretations, one has to use an approach that is capable of handling partial observability (cf. Sect. 3.2).

3.1 Full Observability

It is clear that in the fully-observable case the maximum likelihood estimators $\widehat{p_n}$ for the probabilistic facts $p_n :: f_n$ can be obtained by counting the number of true ground instances in every interpretation, that is,

$$\widehat{p_n} = \frac{1}{Z_n} \sum_{m=1}^{M} \sum_{k=1}^{K_n^m} \delta_{n,k}^m \qquad \text{where} \qquad \delta_{n,k}^m := \begin{cases} 1 \text{ if } f_n\theta_{n,k}^m \in I_m \\ 0 \qquad \text{else} \end{cases} \tag{2}$$

and $\theta_{n,k}^m$ is the k-th possible ground substitution for the fact f_n in the interpretation I_m and K_n^m is the number of such substitutions. The sum is normalized by $Z_n = \sum_{m=1}^{M} K_n^m$, the total number of ground instances of the fact f_n in all training examples. If Z_n is zero, i.e., no ground instance of f_n is used, $\widehat{p_n}$ is undefined and one must not update p_n.

Before moving on to the partial observable case, let us consider the issue of determining the possible substitutions $\theta_{n,k}^m$ for a fact $p_n :: f_n$ and an interpretation I_m. To resolve this, we assume that the facts f_n are typed and that each interpretation I_m contains an explicit definition of the different types in the form of fully-observable unary predicates. In the alarm example, the predicate `person/1` can be regarded as the type of the (first) argument of `al(X)` and `calls(X)`. This predicate can differ between interpretations. One person, i.e., can have `john` and `mary` as neighbors, another one `ann`, `bob` and `eve`.

3.2 Partial Observability

In many applications the training examples are partially observed. In the alarm example, we may receive a phone call but we may not know whether an earthquake has in fact occurred. In the partial observable case – similar to Bayesian networks – a closed-form solution of the maximum likelihood parameters is infeasible. Instead, one has to replace in (2) the term $\delta_{n,k}^m$ by $\mathbb{E}_{\mathcal{T}}[\delta_{n,k}^m | I_m]$, i.e., the conditional expectation given the interpretation under the current model $\mathcal{T}$,

$$\widehat{p_n} = \frac{1}{Z_n} \sum_{m=1}^{M} \sum_{k=1}^{K_n^m} \mathbb{E}_{\mathcal{T}}[\delta_{n,k}^m | I_m] \ . \tag{3}$$

As in the fully observable case, the domains are assumed to be given. Before describing the Soft-EM algorithm for finding $\widehat{p_n}$, we illustrate one of its crucial properties using the alarm example. Assume that our partial interpretation is $I^+ = \{\texttt{person(mary)}, \texttt{person(john)}, \texttt{alarm}\}$ and $I^- = \varnothing$. It is clear that for calculating the marginal probability of all probabilistic facts – these are the expected counts – only the atoms in $\{\texttt{burglary}, \texttt{earthquake}, \texttt{al(john)}, \texttt{al(mary)}\} \cup I$ are relevant. This is due to the fact that the remaining atoms $\{\texttt{calls(john)}, \texttt{calls(mary)}\}$ cannot be used in any proof for the facts observed in the interpretations. We call atoms, which are relevant for the distribution of the ground atom x, the *dependency set* of x. It is defined as $\mathrm{dep}_{\mathcal{T}}(x) := \{f \text{ ground fact} \mid \text{a ground SLD-proof in } \mathcal{T} \text{ for } x \text{ contains } f\}$. Our goal is to restrict the probability calculation to the dependent atoms only. Hence we generalize this set to partial interpretations I as follows $\mathrm{dep}_{\mathcal{T}}(I) := \bigcup_{x \in (I^+ \cup I^-)} \mathrm{dep}_{\mathcal{T}}(x)$ and introduce the notion of a restricted ProbLog theory.

Definition 2. *Let $\mathcal{T} = \mathcal{F} \cup \mathcal{BK}$ be a ProbLog theory and $I = (I^+, I^-)$ a partial interpretation. Then we define $\mathcal{T}^r(I) = \mathcal{F}^r(I) \cup \mathcal{BK}^r(I)$, the* interpretation-restricted *ProbLog theory, as follows. $\mathcal{F}^r(I) = L_\mathcal{T} \cap \mathrm{dep}_\mathcal{T}(I)$ and $\mathcal{BK}^r(I)$ is obtained by computing all ground instances of clauses in $\mathcal{BK}$ in which all atoms appear in $\mathrm{dep}_\mathcal{T}(I)$.*

For the partial interpretation $I = (\{\texttt{burglary}, \texttt{alarm}\}, \varnothing)$, for instance, $\mathcal{BK}^r(I)$ is $\{\texttt{alarm :- burglary}, \texttt{alarm :- earthquake}\}$ and the restricted set of facts $\mathcal{F}^r(I)$ is $\{\texttt{0.1 :: burglary}, \texttt{0.2 :: earthquake}\}$.

The restricted theory $T^r(I)$ cannot be larger than $\mathcal{T}$. More important, it is always finite since we assume the finite support property and the evidence being a finite conjunction of ground atoms. In many cases it will be much smaller, which allows for learning in domains where the original theory does not fit in memory. It can be shown using the independence of probabilistic facts in ProbLog, that the conditional probability of a ground instance of f_n given I calculated in the theory $\mathcal{T}$ is equivalent to the probability calculated in $\mathcal{T}^r(I)$, that is,

$$\mathbb{E}_\mathcal{T}[\delta^m_{n,k}|I_m] = \begin{cases} \mathbb{E}_{\mathcal{T}^r(I_m)}[\delta^m_{n,k}|I_m] & \text{if } f_n \in \mathrm{dep}_\mathcal{T}(I_m) \\ p_n & \text{otherwise} \end{cases} \tag{4}$$

We exploit this property in the following section when developing the Soft-EM algorithm for finding the maximum likelihood parameters $\widehat{\mathbf{p}}$ defined in (3).

4 The LFI-ProbLog Algorithm

The algorithm starts by constructing a Binary Decision Diagram (BDD) [2] for every training example I_m (cf. Sect. 4.1), which is then used to compute the expected counts $\mathbb{E}[\delta^m_{n,k}|I_m]$ (cf. Sect. 4.3). A BDD is a compact graphical representation of a Boolean formula. In our case, the Boolean formula (or, equivalently, the BDD) represents the conditions under which the partial interpretation will be generated by the ProbLog program and the variables in the formula are the ground atoms in $\mathrm{dep}_\mathcal{T}(I_m)$. Basically, any truth assignment to these facts that satisfies the Boolean formula (or the BDD) will result in the partial interpretation. Given a fixed variable order, a Boolean function f can be represented as a full Boolean decision tree where each node N on the ith level is labeled with the ith variable and has two children called low $l(N)$ and high $h(N)$. Each path from the root to a leaf represents a complete variable assignment. If variable x is assigned 0 (1), the branch to the low (high) child is taken. Each leaf is labeled with the value of f given the variable assignment represented by the corresponding path from the root. We use $\mathbb{1}$ to denote true and $\mathbb{0}$ to denote false. Starting from such a tree, one obtains a BDD by merging isomorphic subgraphs and deleting redundant nodes until no further reduction is possible. A node is redundant if and only if the subgraphs rooted at its children are isomorphic. In Fig. 1, dashed edges indicate 0's and lead to *low children*, solid ones indicate 1's and lead to *high children*.

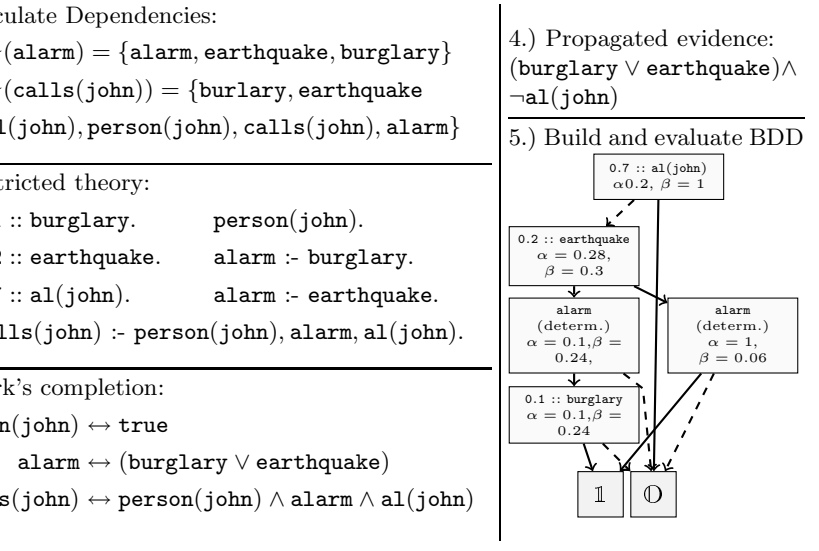

Fig. 1. The different steps of the LFI-ProbLog algorithm for the training example $I^+ = \{\texttt{alarm}\}$, $I^- = \{\texttt{calls(john)}\}$. Normally the alarm node in the BDD is propagated away in Step 4, but it is kept here for illustrative purposes. The nodes are labeled with their probability and the up- and downward probabilities.

4.1 Computing the BDD for an Interpretation

The LFI-ProbLog algorithm generates the BDD that encodes a partial interpretation I. Due to the usage of Clark's completion in Step 3 the algorithm requires a *tight* ProbLog program as input. Clark's completion allows one to propagate values from the head to the bodies of clauses and vice versa. It states that the head is true if *and only if* at least one of its bodies is true, which captures the least Herbrand model semantics of tight definite clause programs. The algorithm works as follows (c.f. Fig 1):

1. Compute $\text{dep}_{\mathcal{T}(I)}$. This is the set of ground atoms that may have an influence on the truth value of the atoms with known truth value in the partial interpretation I. This is realized by applying the definition of $\text{dep}_{\mathcal{T}(I)}$ directly using a tabled meta-interpreter in Prolog. We use tabling to store subgoals and avoid recomputation.
2. Use $\text{dep}_{\mathcal{T}(I)}$ to compute $\mathcal{BK}^r(I)$, the background theory $\mathcal{BK}$ restricted to the interpretation I (cf. Definition 2 and (4)).
3. Compute $clark(\mathcal{BK}^r(I))$, which denotes Clark's completion of $\mathcal{BK}^r(I)$; it is computed by replacing all clauses with the same head h :- $body_1$, ..., h :- $body_n$ by the corresponding formula $h \leftrightarrow body_1 \vee \ldots \vee body_n$.
4. Simplify $clark(\mathcal{BK}^r(I))$ by propagating known values for the atoms in I. This step eliminates ground atoms with known truth value in I. That is, we simply fill out their value in the theory $clark(\mathcal{BK}^r(I))$, and then we

propagate these values until no further propagation is possible. This is akin to the first steps of the Davis-Putnam algorithm.
5. Construct the BDD_I, which compactly represents the Boolean formula consisting of the resulting set of clauses. This BDD_I is used by the Algorithm 1 outlined in Section 4.3 to compute the expected counts.

In Step 4 of the algorithm, atoms f_n with known truth values v_n are removed from the formula and in turn from the BDD. This has to be taken into account both when calculating the probability of the interpretation and the expected counts of these variables. The probability of the partial interpretation I given the ProbLog program $\mathcal{T}(\mathbf{p})$ can be calculated as:

$$P_w(I|\mathcal{T}(\mathbf{p})) = P(BDD_I) \cdot \prod_{f_n \text{ known in } I} P(f_n = v_n) \tag{5}$$

where v_n is the value of f_n in I and $P(BDD_i)$ is the probability of the BDD as defined in the following subsection. The probability calculation is implemented using (5). For ease of notation, however, we shall act as if BDD_I included the variables corresponding to random facts with known truth value in I.

In addition, for computing the expected counts, we also need to consider the nodes and atoms that have been removed from the Boolean formula when the BDD has been computed in a compressed form. See for example in Fig. 1 (5) the probabilistic fact `burglary`. It only occurs on the left path to the $\mathbb{1}$-terminal, but it is with probability 0.1 also true on the right path. Therefore, we treat missing atoms at a particular level as if they were there and simply go to the next node independent of whether the missing atom has the value true or false.

4.2 Automated Theory Splitting

For large ground theories the naively constructed BDDs are too big to fit in memory. BDD tools use heuristics to find a variable order that minimizes size of the BDD. The runtime of this step is exponential in the size of the input, which is prohibitive for parameter learning. We propose an algorithm that identifies independent parts of the grounded theory $clark(\mathcal{BK}^r(I))$ (the output of Step 4). The key observation is, that the BDD for the Boolean formula $A \wedge B$ can be decomposed into two BDDs, one for BDD for A and one for B respectively, if A and B do not share a common variable. Since each variable is contained in at most one BDD, the expected counts of variables can be computed as the union of the expected count calculation on both BDDs. In order to use the automatic theory splitting, one has to replace step 5 of the BDD construction (cf. Section 4.1) with the following algorithm. The idea is to identify sets of independent formulae in a theory by mapping the theory onto a graph as follows.

1. Add one node per clause in $clark(\mathcal{BK}^r(I))$.
2. Add an edge between two nodes if the corresponding clauses share an atom.
3. Identify the connected components in the resulting graph.
4. Build for each of the connected components one BDD representing the conjunction of the clauses in the component.

The resulting set of BDDs are used by the algorithm outlined in the next section to compute the expected counts.

4.3 Calculate Expected Counts

One can calculate the expected counts $\mathbb{E}[\delta^m_{n,k}|I_m]$ by a dynamic programming approach on the BDD. The algorithm is akin to the forward/backward algorithm for HMMs or the inside/outside probability of PCFGs. We use p_N as the probability that the node N will be left using the branch to the high-child and $1-p_N$ otherwise. For a node N corresponding to a probabilistic fact f_i this probability is $p_N = p_i$ and $p_N = 1$ otherwise. We use the indicator function $\pi_N = 1$ to test whether a node N is deterministic. For every node N in the BDD we compute:

1. The *upward probability* $\alpha(N)$ represents the probability that the logical formula encoded by the sub-BDD rooted at N is true. For instance, in Fig. 1 (5), the upward probability of the leftmost node for `alarm` represents the probability that the formula `alarm` $\wedge$ `burglary` is true.
2. The *downward probability* $\beta(N)$ represents the probability of reaching the current node N on a random walk starting at the root, where at deterministic nodes both paths are followed in parallel. If all random walkers take the same decisions at the remaining nodes it is guaranteed that only one reaches the $\mathbb{1}$-terminal. This is due to the fact that the values of all deterministic nodes are fixed given the values for all probabilistic facts. For instance, in Fig. 1 (5), the downward probability of the left alarm node is equal to the probability of $\neg$`earthquake` $\wedge$ $\neg$`al(john)`, which is $(1-0.2)\cdot(1-0.7)$.

Due to their definition, summing values of α and β at any level n in the BDD yields the BDD probability, that is, $P(BDD) = \sum_{N \text{ node at level } n} \alpha(N)\beta(N)$. Each path from the root to the $\mathbb{1}$-terminal corresponds to an assignment of values to the variables that satisfies the Boolean formula underlying the BDD. The probability that such a path passes through the node N can be computed as $\alpha(N)\cdot\beta(N)\cdot(P(BDD))^{-1}$. The upward and downward probabilities are computed using the following formulae (cf. Fig. 2):

$$\alpha(\mathbb{0}) = 0 \qquad \alpha(\mathbb{1}) = 1 \qquad \beta(Root) = 1$$
$$\alpha(N) = \alpha(h(N))\cdot p_N^{\pi_N} + \alpha(l(N))\cdot(1-p_N)^{\pi_N}$$
$$\beta(N) = \sum\nolimits_{N=h(M)} \beta(M)\cdot p_M^{\pi_M} + \sum\nolimits_{N=l(M)} \beta(M)\cdot(1-p_M)^{\pi_M}$$

where π_N is 0 for nodes representing deterministic nodes and 1 otherwise. Due to the definition of α and β, the probability of the BDD is returned both at the root and at the $\mathbb{1}$-terminal, that is, $P(BDD) = \alpha(Root) = \beta(\mathbb{1})$. Given these values, one can compute the expected counts $\mathbb{E}[\delta^m_{n,k}|I_m]$ as

$$\mathbb{E}[\delta^m_{n,k}|I_m] = \sum\nolimits_{N \text{ represents } f_m} \beta(N)\cdot p_N \cdot \alpha(h(N))\cdot(P(BDD))^{-1} \ .$$

One computes the downward probability α from the root to the leaves and the upward probability β from the leaves to the root. Intermediate results are stored and reused when nodes are revisited. Both parts are sketched in Algorithm 1.

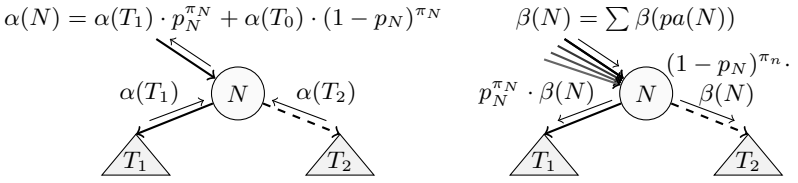

Fig. 2. Propagation step of the upward probability (left) and for the downward probability (right). The indicator function π_N is 1 if N is a probabilistic node and 0 otherwise.

Algorithm 1. Calculating α and β. The nodes $l(h)$ and $h(n)$ are the low and high child of the node n respectively.

```
function ALPHA(BDD node n)
    If n is the 𝟙 then return 1
    If n is the 𝟘 then return 0
    if n probabilistic fact then
        return p_n · ALPHA(h(n))
               +(1−p_n)·ALPHA(l(n))
    return ALPHA(h(n))+
           quadALPHA(l(n))
```

```
function BETA(BDD node n)
    q := priority queue using the BDD's order
    enqueue(q, n)
    BETA := array of 0's of length size(BDD)
    BETA[root(n)]:= 1
    while q not empty do
        n := dequeue(q)
        BETA[h(n)]+ = BETA[n] · p_n^{π_n}
        BETA[l(n)]+ = BETA[n] · (1 − p_n)^{π_n}
        enqueue(q, h(n)) if not yet in q
        enqueue(q, l(n)) if not yet in q
```

5 Experiments

We implemented LFI-ProbLog in YAP Prolog and use SimpleCUDD for the BDD operations. We used two datasets to evaluate LFI-ProbLog. The *WebKB* benchmark serves as test case to compare with state-of-the-art systems. The *Smokers* dataset is used to test the algorithm in terms of the learned model, that is, how close are the parameters to the original ones. The experiments were run on an Intel Core 2 Quad machine (2.83 GHz) with 8GB RAM.

5.1 WebKB

The goal of this experiment is to answer the following questions:

Q1. *Is LFI-ProbLog competitive with existing state-of-the-art frameworks?*
Q2. *Is LFI-ProbLog insensitive to the initial probabilities?*
Q3. *Is the theory splitting algorithm capable of handling large data sets?*

In this experiment, we used the WebKB [3] dataset. It contains four folds, each describing the link structure of pages from one of the following universities: Cornell, Texas, Washington, and Wisconsin. WebKB is a collective classification task, that is, one wants to predict the class of a page depending on the classes of the pages that link to it and depending on the words being used in the

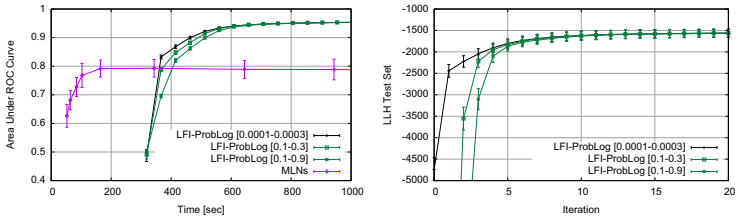

Fig. 3. Area under the ROC curve against the learning time (left) and test set log likelihood for each iteration of the EM algorithm (right) for WebKB

text. To allow for an objective comparison with Markov Logic networks and the results of Domingos and Lowd [9], we used their slightly altered version of WebKB. In their setting each page is assigned exactly one of the classes "course", "faculty", "other", "researchproject", "staff", or "student". Furthermore, the class "person", present in the original version, has been removed. We use the following model that contains one non-ground probabilistic fact for each pair of CLASS and WORD. To account for the link structure, it contains one non-ground probabilistic fact for each pair of CLASS1 and CLASS2.

$$\mathrm{P} :: \texttt{pfWoCla}(\texttt{Page}, \textsc{Class}, \textsc{Word}).$$
$$\mathrm{P} :: \texttt{pfLiCla}(\texttt{Page1}, \texttt{Page2}, \textsc{Class1}, \textsc{Class2}).$$

The probabilities P are unknown and have to be learned by LFI-ProbLog. As there are 6 classes and 771 words, our model has $6 \times 771 + 6 \times 6 = 4662$ parameters. In order to combine the probabilistic facts and predict the class of a page we add the following background knowledge.

$$\texttt{cl}(\texttt{Pa}, \texttt{C}) \text{ :- } \texttt{hasWord}(\texttt{Pa}, \texttt{Word}),\ \texttt{pfWoCla}(\texttt{Pa}, \texttt{Word}, \texttt{C}).$$
$$\texttt{cl}(\texttt{Pa}, \texttt{C}) \text{ :- } \texttt{linksTo}(\texttt{Pa2}, \texttt{Pa}),\ \texttt{pfLiCla}(\texttt{Pa2}, \texttt{Pa}, \texttt{C2}, \texttt{C}),\ \texttt{cl}(\texttt{Pa2}, \texttt{C2}).$$

We performed a 4-fold cross validation, that is, we trained the model on three universities and then tested it on the fourth one. We repeated this for all four universities and averaged the results. We measured the area under the precision-recall curve (AUC-PR), the area under the ROC curve (AUC-ROC), the log likelihood (LLH), and the accuracy after each iteration of the EM algorithm. Our model does not express that each page has exactly one class. To account for this, we normalize the probabilities per page. Figure 3 (left) shows the AUC-ROC plotted against the average training time. The initialization phase, that is running steps 1-4 of LFI-ProbLog, takes ≈ 330 seconds, and each iteration of the EM algorithm takes ≈ 62 seconds. We initialized the probabilities of the model randomly with values sampled from the uniform distribution between 0.1 and 0.9, which is shown as the graph for LFI-ProbLog [0.1-0.9]. After 10 iterations (≈ 800 s) the AUC-ROC is 0.950 ± 0.002, the AUC-PR is 0.828 ± 0.006, and the accuracy is 0.769 ± 0.010.

We compared LFI-ProbLog with Alchemy [9] and LeProbLog [13]. Alchemy is an implementation of Markov Logic networks. We use the model suggested by

Domingos and Lowd [9] that uses the same features as our model, and we train it according to their setup.[2]. The learning curve for AUC-ROC is shown in Figure 3 (left). After 943 seconds Alchemy achieves an AUC-ROC of 0.923 ± 0.016, an AUC-PR of 0.788 ± 0.036, and an accuracy of 0.746 ± 0.032. LeProbLog is a regression-based parameter learning algorithm for ProbLog. The training data has to be provided in the form of queries annotated with the target probability. It is not possible to learn from interpretations. For WebKB, however, one can map one interpretation to several training examples $P(\texttt{class}(\text{URL},\textsc{Class}) = \text{P}$ per page where P is 1 if the class of URL is Class and else 0. This is possible, due to the existence of a *target predicate*. We used the standard settings of LeProblog and limit the runtime to 24 hours. Within this limit, the algorithm performed 35 iteration of gradient descent. The final model obtained an AUC-PR of 0.419 ± 0.014, an AUC-ROC of 0.738 ± 0.014, and an accuracy of 0.396 ± 0.020. These results affirmatively answer **Q1**.

We tested how sensitive LFI-ProbLog is for the initial fact probabilities by repeating the experiment with values sampled uniformly between 0.1 and 0.3 and sampled uniformly between 0.0001 and 0.0003 respectively. As the graphs in Figure 3 indicate, the convergence is initially slower and the initial LLH values differ. This is due to the fact that the ground truth probabilities are small, and if the initial fact probabilities are small too, one obtains a better initial LLH. All settings converge to the same results, in terms of AUC and LLH. This suggests that LFI-ProbLog is insensitive to the start values (cf. **Q2**).

The BDDs for the WebKB dataset are too large to fit in memory and the automatic variable reordering is unable to construct the BDD in a reasonable amount of time. We used two different approaches to resolve this. In the first approach, we manually split each training example, that is, the grounded theory together with the known class for each page, into several training examples. The results shown in Figure 3 are based on this manual split. In the second approach, we used the automatic splitting algorithm presented in Section 4.2. The resulting BDDs are identical to the manual split setting, and the subsequent runs of the EM algorithm converge to the same results. Hence when plotting against the iteration, the graphs are identical. The resulting ground theory is much larger and the initialization phase therefore takes 247 minutes. However, this is mainly due to the overhead for indexing, database access and garbage collection in the underlying Prolog system. Grounding and Clark's completion take only 6 seconds each, the term simplification step takes roughly 246 minutes, and the final splitting algorithm runs in 40 seconds. As we did not optimize the implementation of the term simplification, we see a big potential for improvement, for instance by tabling intermediate simplification steps. This affirmatively answers **Q3**.

5.2 Smokers

We set up an experiment on an instance of the *Smokers* dataset (cf. [9]) to answer the question

[2] Daniel Lowd provided us with the original scripts for the experiment setup. We report on the evaluation based on the rerun of the experiment.

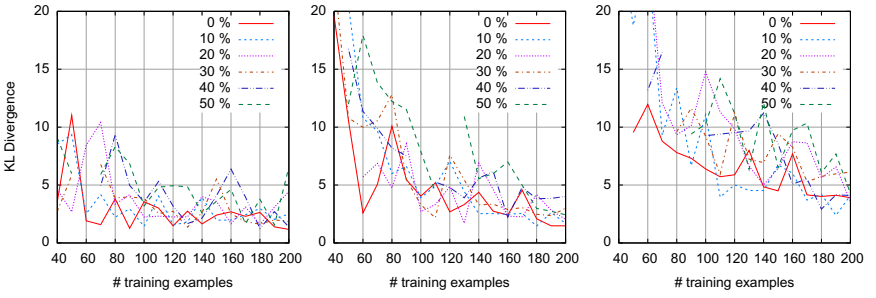

Fig. 4. Result for KL-divergence in the smokers domain. The plots are for left to right 3, 4, 5 smokers. Different graphs correspond to different amounts of missing data.

Q4. *Is LFI-Problog able to recover the parameters of the original model with a reasonable amount of data?*

Missing or incorrect values are two different types of noise that can occur in real-world data. While incorrect values can be compensated by additional data, missing values cause local maxima in the likelihood function. In turn, they cause the learning algorithm to yield parameters different from the ones used to generate the data. LFI-ProbLog computes the maximum likelihood parameters given some evidence. Hence the algorithm should be capable of recovering the parameters used to generate a set of interpretations. We analyze how the amount of required training data increases as the size of the model increases. Furthermore, we test for the influence of missing values on the results. We assess the quality of the learned model, that is, the difference to the original model parameters by computing the Kullback Leibler (KL) divergence. ProbLog allows for an efficient computation of this measure due to the independence of the probabilistic facts. In this experiment, we use a variant of the "Smokers" model which can be represented in ProbLog as follows:

$$
\begin{aligned}
p\_si &:: \texttt{smokes\_i(X,Y)} \quad \text{// person influenced by a smoking friend}\\
p\_sp &:: \texttt{smokes\_p(X)} \quad \text{// person starts smoking without external reason}\\
p\_cs &:: \texttt{cancer\_s(X).} \quad \text{// cancer is caused by smoking}\\
p\_cp &:: \texttt{cancer(X).} \quad \text{// cancer without external reason}\\
smokes(X) &\texttt{ :- friend(X,Y),smokes(Y),smokes\_i(X,Y));smokes\_p(X).}\\
cancer(X) &\texttt{ :- smokes(X),cancer\_s(X));cancer\_p(X).}
\end{aligned}
$$

Due to space restrictions, we omit the details on how to represent this such that the program is tight. We set the number of persons to 3,4 and 5 respectively and sampled from the resulting models up to 200 interpretations each. From these datasets we derived new instances by randomly removing $10-50\%$ of the atoms. The size of an interpretation grows quadratically with the number of persons. The model, as described above, has an implicit parameter tying between ground

instances of non-ground facts. Hence the number of model parameters does not change with the number of persons. To measure the influence of the model size, we therefore trained grounded versions of the model, where the grounding depends on the number of persons. For each dataset we ran LFI-ProbLog for 50 iterations of EM. Manual inspection showed that the probabilities stabilized after a few, typically 10, iterations. Figure 4 shows the KL divergence for 3, 4 and 5 persons respectively. The closer the KL divergence is to 0, the closer the learned model is to the original parameters. As the graphs show, the learned parameters approach the parameters of the original model as the number of training examples grows. Furthermore, the amount of missing values has little influence on the distance between the true and the learned parameters. Hence LFI-Problog is capable of recovering the original parameters and it is robust against missing values. This affirmatively answers **Q4**.

6 Related Work

Most of the existing parameter learning approaches for ProbLog [6], PRISM [22], and SLPs [17] are based on learning from entailment. For ProbLog, there exists a learning algorithm based on regression where each training example is a ground fact together with the target probability [13]. In contrast to LFI-ProbLog, this approach does *not* assume an underlying generative process; neither at the level of predicates nor at the level of interpretations. Sato and Kameya have contributed various interesting and advanced learning algorithms that have been incorporated in PRISM. Ishihata *et al.* [14] consider a parameter learning setting based on Binary Decision Diagrams (BDDs) [2]. In contrast to our work, they assume the BDDs to be given, whereas LFI-ProbLog, constructs them in an intelligent way from evidence and a ProbLog theory. Ishihata *et al.* suggest that their approach can be used to perform learning from entailment for PRISM programs. This approach has been recently adopted for learning CP-Logic programs (cf. [1]). The BDDs constructed by LFI-ProbLog are a compact representation of all possible worlds that are consistent with the evidence. LFI-ProbLog estimates the marginals of the probabilistic facts in a dynamic programming manner on the BDDs. While this step is inspired by [14], we tailored it towards the specifics of LFI-ProbLog, that is, we allow deterministic nodes to be present in the BDDs. This extension is crucial, as the removal of deterministic nodes can results in an exponential growth of the Boolean formulae underlying the BDD construction. Riguzzi [20] uses a transformation of ground ProbLog programs to Bayesian networks in order to learn ProbLog programs from interpretations. Such a transformation is also employed in the learning approaches for CP-logic [24,16]. Thon et al. [23] studied how CPT-L, a sequential variant of CP-Logic, can be learned from sequences of interpretations. CPT-L is closely related to LFI-ProbLog. However, CPT-L is targeted towards the sequential aspect of the theory, whereas we consider a more general settings with arbitrary theories. Thon et al. assume full observability, which allows them to split the sequence into separate transitions. They build one BDD per transition, which

is much easier to construct than one large BDD per sequence. Our splitting algorithm is capable of exploiting arbitrary independence. LFI-ProbLog can also be related to knowledge-based model construction approaches in statistical relational learning such as BLPs, PRMs and MLNs [19]. While the setting explored in this paper is standard for the aforementioned formalisms, our approach has significant representational and algorithmic differences from the algorithms used in those formalisms. In BLPS, PRMs and CP-logic, each training example is typically used to construct a ground Bayesian network on which a standard learning algorithm is applied. Although the representation generated by Clark's completion is quite close to the representation of Markov Logic, there are subtle differences. While Markov Logic uses weights on clauses, we use probabilities attached to single facts.

7 Conclusions

We have introduced a novel parameter learning algorithm from interpretations for the probabilistic logic programming language ProbLog. This has been motivated by the differences in the learning settings and applications of typical knowledge-based model construction approaches and probabilistic logic programming approaches. The LFI-ProbLog algorithm tightly couples logical inference with a probabilistic EM algorithm at the level of BDDs. Possible directions of future work include using d-DNNF representations instead of BDDs [10] and a transformation to Boolean formulae that does not require tight programs.

Acknowledgments. Bernd Gutmann is supported by the Research Foundation-Flanders (FWO-Vlaanderen). This work is supported by the GOA project 2008/08 Probabilistic Logic Learning and by the European Community under contract number FP7-248258-First-MM. We thank Vìtor Santos Costa and Paulo Moura for their help with YAP Prolog.

References

1. Bellodi, E., Riguzzi, F.: EM over binary decision diagrams for probabilistic logic programs. Tech. Rep. CS-2011-01, Università di Ferrara, Italy (2011)
2. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Trans. Computers 35(8), 677–691 (1986)
3. Craven, M., Slattery, S.: Relational learning with statistical predicate invention: Better models for hypertext. Machine Learning 43(1/2), 97–119 (2001)
4. Cussens, J.: Parameter estimation in stochastic logic programs. Machine Learning 44(3), 245–271 (2001)
5. De Raedt, L., Frasconi, P., Kersting, K., Muggleton, S. (eds.): Probabilistic Inductive Logic Programming — Theory and Applications. LNCS (LNAI), vol. 4911. Springer, Heidelberg (2008)
6. De Raedt, L., Kimmig, A., Toivonen, H.: Problog: A probabilistic Prolog and its application in link discovery. In: Veloso, M. (ed.) IJCAI, pp. 2462–2467 (2007)
7. De Raedt, L.: Logical and Relational Learning. Springer, Heidelberg (2008)

8. De Raedt, L., Kersting, K.: Probabilistic inductive logic programming. In: Ben-David, S., Case, J., Maruoka, A. (eds.) ALT 2004. LNCS (LNAI), vol. 3244, pp. 19–36. Springer, Heidelberg (2004)
9. Domingos, P., Lowd, D.: Markov Logic: An Interface Layer for Artificial Intelligence. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, San Francisco (2009)
10. Fierens, D., Van den Broeck, G., Thon, I., Gutmann, B., De Raedt, L.: Inference in probabilistic logic programs using weighted cnf's. In: The 27th Conference on Uncertainty in Artificial Intelligence, UAI 2011 (to appear, 2011)
11. Getoor, L., Friedman, N., Koller, D., Pfeffer, A.: Learning probabilistic relational models. In: Džeroski, S., Lavrač, N. (eds.) Relational Data Mining, pp. 307–335. Springer, Heidelberg (2001)
12. Getoor, L., Taskar, B. (eds.): An Introduction to Statistical Relational Learning. MIT Press, Cambridge (2007)
13. Gutmann, B., Kimmig, A., De Raedt, L., Kersting, K.: Parameter learning in probabilistic databases: A least squares approach. In: Daelemans, W., Goethals, B., Morik, K. (eds.) ECML PKDD 2008, Part I. LNCS (LNAI), vol. 5211, pp. 473–488. Springer, Heidelberg (2008)
14. Ishihata, M., Kameya, Y., Sato, T., Minato, S.: Propositionalizing the EM algorithm by BDDs. In: ILP (2008)
15. Kersting, K., Raedt, L.D.: Bayesian logic programming: theory and tool. In: Getoor, L., Taskar, B. (eds) [12]
16. Meert, W., Struyf, J., Blockeel, H.: Learning ground cp-logic theories by leveraging bayesian network learning techniques. Fundam. Inform. 89(1), 131–160 (2008)
17. Muggleton, S.: Stochastic logic programs. In: De Raedt, L. (ed.) Advances in Inductive Logic Programming. Frontiers in Artificial Intelligence and Applications, vol. 32. IOS Press, Amsterdam (1996)
18. Poole, D.: The independent choice logic and beyond. In: De Raedt, L. et al [5],
19. Richardson, M., Domingos, P.: Markov logic networks. Machine Learning 62, 107–136 (2006)
20. Riguzzi, F.: Learning ground problog programs from interpretations. In: Proceedings of the 6th Workshop on Multi-Relational Data Mining, MRDM 2007 (2007)
21. Sato, T.: A statistical learning method for logic programs with distribution semantics. In: Sterling, L. (ed.) Proceedings of the 12th International Conference on Logic Programming, pp. 715–729. MIT Press, Cambridge (1995)
22. Sato, T., Kameya, Y.: Parameter learning of logic programs for symbolic-statistical modeling. Journal of Artificial Intelligence Research 15, 391–454 (2001)
23. Thon, I., Landwehr, N., De Raedt, L.: A simple model for sequences of relational state descriptions. In: Daelemans, W., Goethals, B., Morik, K. (eds.) ECML PKDD 2008, Part I. LNCS (LNAI), vol. 5211, pp. 506–521. Springer, Heidelberg (2008)
24. Vennekens, J., Denecker, M., Bruynooghe, M.: Representing causal information about a probabilistic process. In: Fisher, M., van der Hoek, W., Konev, B., Lisitsa, A. (eds.) JELIA 2006. LNCS (LNAI), vol. 4160, pp. 452–464. Springer, Heidelberg (2006)