Learning the Parameters of Probabilistic Logic Programs from Interpretations

Bernd Gutmann
Ingo Thon
Luc De Raedt

Report CW 584, June 15, 2010

Katholieke Universiteit Leuven
Department of Computer Science
Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

Learning the Parameters of Probabilistic Logic Programs from Interpretations

Bernd Gutmann
Ingo Thon
Luc De Raedt [⋆]

Report CW 584, June 15, 2010

Department of Computer Science, K.U.Leuven

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. In this paper, a novel parameter estimation algorithm CoPrEM for learning ProbLog programs from partial interpretations is introduced. The algorithm is essentially a Soft-EM algorithm that computes binary decision diagrams for each interpretation allowing for a dynamic programming approach to be implemented. The CoPrEM algorithm has been experimentally evaluated on a number of data sets, which justify the approach and show its effectiveness.

[⋆] firstname.lastname@cs.kuleuven.be

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

1 Introduction

Statistical relational learning [8] and probabilistic logic learning [2, 4] have contributed various representations and learning schemes. Popular approaches include BLPs [12], Markov Logic [15], PRISM [18], PRMs [7], and ProbLog [3, 9]. These approaches differ not only in the underlying representations but also in the learning setting that is employed. Indeed, for learning knowledge-based model construction approaches (KBMC), such as Markov Logic, PRMs, and BLPs, one has typically used possible worlds (that is, Herbrand interpretations or relational state descriptions) as training examples. For probabilistic programming languages – especially the probabilistic logic programming (PLP) approaches based on Sato's distribution semantics [17] such as PRISM and ProbLog – training examples are typically provided in form of labeled facts. The labels that are either the truth values of these facts or target probabilities. In computational learning theory as well as in logical and relational learning [4], the former setting is known as learning from interpretations and the latter as learning from entailment. The differences between these two settings have been well studied and characterized in the literature, and they have also been used for explaining the differences between various statistical relational learning models [4, 5].

The differences between the two settings are akin to those between learning a probabilistic grammar and learning a graphical model (e.g., a Bayesian network). When learning the parameters of probabilistic grammars, the examples are typically sentences sampled from the grammar, and when learning Bayesian networks, the examples are possible worlds (that is, state descriptions). In the former setting, which corresponds to learning from entailment, one usually starts from observations for a *single target* predicate (or non-terminal), whereas in the later setting, which corresponds to learning from interpretations, the observations may specify the value of all random variables in a state-description.

These differences in learning settings also explain why the KBMC and PLP approaches have been applied on different kinds of data sets and applications. Indeed, for PLP, one has, for instance, learned grammars; for KBMC, one has learned models for entity resolution and link prediction. This paper wants to contribute to bridging the gap between these two types of approaches to learning by studying how the parameters of ProbLog programs can be learned from partial interpretations. The key contribution of the paper is the introduction of a novel algorithm called CoPrEM for learning ProbLog programs from interpretations. The name is due to the main steps of the algorithm which are *Completion*, *Propagation* and an *Expectation Maximization* procedure. It will also be shown that the resulting algorithm applies to the usual type of problems that have made MLNs and PRMs so popular.

The paper is organized as follows: In Section 2, we review logic programming concepts as well as the probabilistic programming language ProbLog. Section 3

[⋆] firstname.lastname@cs.kuleuven.be

formalizes the problem of learning the parameters of ProbLog programs from interpretations. Section 4 introduces CoPrEM for finding the maximum likelihood parameters. We report on some experiments in Section 5, and discuss related work in Section 6 before concluding.

2 Probabilistic Logic Programming Concepts

Before reviewing the main concepts of the probabilistic programming language ProbLog, we give the necessary terminology from the field of logic programming.

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 either a variable, a constant, or a functor applied on 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 where the body is empty. 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 have been 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, that is, the set of all ground facts entailed by the theory (cf. [6] for more details). Prolog uses SLD-resolution as standard proof procedure to answer queries.

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. 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* [14] 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 $L \subseteq L_\mathcal{T} = \{f_1\theta_{1,1}, \ldots f_1\theta_{1,K_1}, \ldots, f_N\theta_{N,1}, \ldots, f_N\theta_{N,K_N}\}$.

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

Example 1. 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. Figure 1 shows the Bayesian network that corresponds to this theory.

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

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 can be combined with the background knowledge $\mathcal{BK}$ into a Prolog program. Thus, 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 logically entailed by the program. This Herbrand interpretation represents a *possible world*. For

[1] Throughout the paper, we shall assume that $\mathcal{F}$ is finite, see [17] for the infinite case.

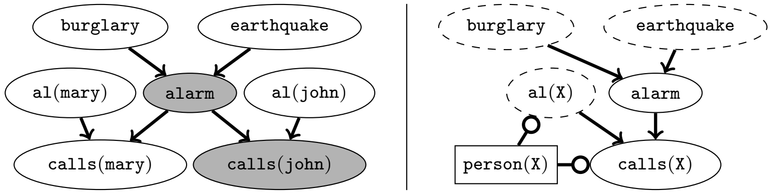

Fig. 1. The Bayesian Network (left) equivalent to Example 1 where the evidence atoms are {`person(mary)`, `person(john)`, `alarm`, ¬`calls(john)`}. An equivalent relational representation (right), where each node corresponds to an atom, box nodes represent logical facts, dashed nodes represent probabilistic facts, and solid elliptic nodes represent derived predicates.

instance, for the total choice $\{\texttt{burglary}\}$ we obtain the interpretation $I = \{\texttt{burglary}, \texttt{alarm}, \texttt{person(john)}, \texttt{person(mary)}\}$. Thus, 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 $0.1 \times (1 - 0.2) \times (1 - 0.3)^2$. We define the *success probability* of a query q as

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

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 succeeds in a randomly selected possible world (according to P_w). The success probability $\mathrm{P}_s(\texttt{calls(john)}|\mathcal{T})$ in Example 1 can be calculated as

$$\begin{aligned} P(\texttt{al(john)} \wedge (\texttt{burglary} \vee \texttt{earthquake})|\mathcal{T}) = \\ &= \mathrm{P}(\texttt{al(john)} \wedge \texttt{burglary}|\mathcal{T}) + \mathrm{P}(\texttt{al(john)} \wedge \texttt{earthquake}|\mathcal{T}) \\ &\quad - \mathrm{P}(\texttt{al(john)} \wedge \texttt{burglary} \wedge \texttt{earthquake}|\mathcal{T}) \\ &= 0.7 \cdot 0.1 + 0.7 \cdot 0.2 - 0.7 \cdot 0.1 \cdot 0.2 = 0.196 \end{aligned}$$

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:

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

So, the predicates do *not* encode a probability distribution over their instances. This differs from probabilistic grammars and its extensions such as stochastic logic programs [1], where each predicate or non-terminal defines a probability distribution over its instances, which enables these approaches to sample instances 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. This setting is known in the literature as learning from entailment [4, 5]. Sato and Kameya's well-known learning algorithm for PRISM [18] 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.

On the other hand, while the ProbLog semantics does not encode a generative process at the level of individual predicates, it does encode a generative process 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. The key contribution of this paper is the introduction of a novel learning algorithm for learning from interpretations in ProbLog. It can learn both from fully and partially observable interpretations.

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 $(\bigwedge_{a_j \in I^+} a_j) \wedge (\bigwedge_{a_j \in I^-} \neg a_j)$. Considering Example 1, the probability of the following partial interpretation

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

is $P_w((I^+, I^-)) = 0.1 \times 0.7 \times (1 - 0.7) \times (0.2 + (1 - 0.2))$ as there are two total choices that result in this partial interpretation.

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 such as ProbLog, ICL and PRISM. The setting 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*

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

Thus, we are given a ProbLog program and a set of partial interpretations and the goal is to find the maximum likelihood parameters.

Finding maximum likelihood parameters $\widehat{\mathbf{p}}$ can be done using a Soft-EM algorithm. There are two cases to consider while developing the algorithm. The first is that the interpretations are complete and everything is observable, the second that partial interpretations are allowed, and that there is partial observability.

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 simply counting the number of true ground instances in every interpretation

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

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. Before going 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 essentially assume[2] 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 a unary predicate (that is fully observable). 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 can have `john` and `mary` as neighbors, another one `ann`, `bob` and `eve`. The maximum likelihood estimator accounts for that by allowing the domain sizes K_n^m to be example-dependent. The use of such types is standard practice in statistical relational learning and inductive logic programming.

3.2 Partial Observability

In many applications the training examples are only 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 can replace $\delta_{n,k}^m$ by its conditional expectation given the Interpretation under the current model $\mathbb{E}[\delta_{n,k}^m | I_m]$ in the previous formula yielding:

$$\widehat{p_n} = \frac{1}{M} \sum_{m=1}^{M} \left[\frac{1}{K_n^m} \sum_{k=1}^{K_n^m} \mathbb{E}[\delta_{n,k}^m | I_m] \right]$$

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 a crucial property 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 $\{$`burglary`, `earthquake`, `al(john)`, `al(mary)`, `alarm`$\} \cup I^+$ are relevant. This is due to the fact that the remaining atoms $\{$`calls(john)`, `calls(mary)`$\}$ cannot be used in any proof for the facts observed in the interpretations. Therefore, they do not influence the probability of the partial interpretation[3]. This motivates the following definition.

Definition 2. *Let $\mathcal{T} = \mathcal{F} \cup \mathcal{BK}$ be a ProbLog theory and x a ground atom then the* dependency set of *x is defined as:*

$$\mathrm{dep}_{\mathcal{T}}(x) := \{f \text{ ground fact} | \text{a ground SLD-proof in } \mathcal{T} \text{ for } x \text{ contains } f\}.$$

Thus, $\mathrm{dep}_{\mathcal{T}}(x)$ contains all ground atoms that appear in a possible proof of the atom x. This can be generalized to partial interpretations:

Definition 3. *Let $\mathcal{T} = \mathcal{F} \cup \mathcal{BK}$ be a ProbLog theory and $I = (I^+, I^-)$ a partial interpretation then the* dependency set of the partial interpretation *I is defined as $\mathrm{dep}_{\mathcal{T}}(I) := \bigcup_{x \in (I^+ \cup I^-)} \mathrm{dep}_{\mathcal{T}}(x)$.*

Before showing that it is sufficient to the restrict the calculation of the probability to the dependent atoms, we need the notion of a restricted ProbLog theory.

Definition 4. *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 $\mathrm{dep}_{\mathcal{T}}(I)$.*

[2] This assumption can be relaxed if the types are computable from the Problog program and the current interpretation.

[3] Such atoms play a role similar to that of barren nodes in Bayesian networks [11].

For instance, for the partial interpretation $I = (\{\texttt{burglary}, \texttt{alarm}\}, \varnothing)$, the restricted background theory $\mathcal{BK}^r(I)$ contains the single clause `alarm :- burglary` and $\mathcal{F}^r(I)$ contains only `burglary`.

Theorem 1. *For all ground probabilistic facts f_n and partial interpretations I*

$$P(f_n|I,\mathcal{T}) = \begin{cases} P(f_n|I,\mathcal{T}^r(I)) & \text{if } f_n \in \text{dep}_{\mathcal{T}}(I) \\ p_n & \text{otherwise} \end{cases}$$

where $\mathcal{T}^r(I)$ is the interpretation-restricted ProbLog theory of $\mathcal{T}$.

This theorem shows, that the conditional probability of f_n given I calculated in the theory $\mathcal{T}$ is equivalent to the probability calculated in $\mathcal{T}^r(I)$. The probabilistic facts of $\mathcal{T}^r(I)$ are restricted to the atomic choices occurring in $\text{dep}_{\mathcal{T}}(I)$. Working with $T^r(I)$ is often more efficient than working with $\mathcal{T}$.

Proof. Can be proven using the independence of probabilistic facts in ProbLog.

4 The CoPrEM algorithm

We now develop the Soft-EM algorithm for finding the maximum likelihood parameters $\widehat{\mathbf{p}}$. The algorithm starts by constructing a Binary Decision Diagram (BDD) 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. 4.2).

Before giving the details on how to construct and to evaluate BDDs, we review their relevant concepts. A BDD is a compact graph representation of a Boolean formula, an example is shown in Fig. 2. 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 $\text{dep}_{\mathcal{T}}(I_m)$. 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 ordering, 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 some leaf stands for one complete variable assignment. If variable x is assigned 0 (1), the branch to the low (high) child is taken. Each leaf is labeled by the outcome of f given the variable assignment represented by the corresponding path. (where we use $\mathbb{1}$ and $\mathbb{0}$ to denote true and 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. 2, dashed edges indicate 0's and lead to *low children*, solid ones indicate 1's and lead to *high children*.

4.1 Computing the BDD for an interpretation

The different steps we go through to compute the BDD for a partial interpretation I are as follows (and also illustrated in Fig. 2):

1. Compute $\text{dep}_{\mathcal{T}(I)}$, that is, compute 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. Tabling stores each considered atom in order to avoid that the same atom is proven repeatedly.
2. Compute $\mathcal{BK}^r(I)$, the background theory $\mathcal{BK}$ restricted to the interpretation I (cf. Definition 4 and Theorem 1).

1.) Calculate Dependencies:

$dep_{\mathcal{T}}(\texttt{alarm}) = \{\texttt{alarm}, \texttt{earthquake}, \texttt{burglary}\}$

$dep_{\mathcal{T}}(\texttt{calls(john)}) = \{\texttt{burlary}, \texttt{earthquake}$
$\texttt{al(john)}, \texttt{person(john)}, \texttt{calls(john)}, \texttt{alarm}\}$

2.) Restricted theory:

0.1 :: burglary.	person(john).
0.2 :: earthquake.	alarm :- burglary.
0.7 :: al(john).	alarm :- earthquake.

calls(john) :- person(john), alarm, al(john).

3.) Clark's completion:

$\texttt{person(john)} \leftrightarrow \texttt{true}$

$\texttt{alarm} \leftrightarrow (\texttt{burglary} \vee \texttt{earthquake})$

$\texttt{calls(john)} \leftrightarrow \texttt{person(john)} \wedge \texttt{alarm} \wedge \texttt{al(john)}$

4.) Propagated evidence:
$(\texttt{burglary} \vee \texttt{earthquake}) \wedge$
$\neg \texttt{al(john)}$

5.) Build and evaluate BDD

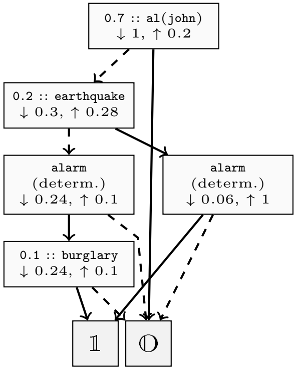

Fig. 2. The different steps of the CoPrEM algorithm for the training example $I^+ = \{\texttt{alarm}\}$, $I^- = \{\texttt{calls(john)}\}$. Normally the alarm node in the BDD is propagated away in Step 4, but kept here for illustrative purposes. The nodes are labeled with their probability and the up- and downward probabilities.

3. Compute $Comp(\mathcal{BK}^r(I))$, where $Comp(\mathcal{BK}^r(I))$ denotes Clark's completion of $\mathcal{BK}^r(I)$; Clark's completion of a set of clauses is computed by replacing all clauses with the same head h :- $body_1$, ..., h :- $body_n$ by the corresponding formula $h \leftrightarrow body_1 \vee \ldots \vee body_n$. Clark's completion allows one to propagate values from the head to the bodies and vice versa. It states that the head is true if *and only if* one of its bodies is true and captures the least Herbrand model semantics of definite clause programs.
4. Simplify $Comp(\mathcal{BK}^r(I))$ by propagating known values for the atoms in I. In this step, we first eliminate the ground atoms with known truth value in I. That is, we simply fill out their value in the theory $Comp(\mathcal{BK}^r(I))$, 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 compactly representing the Boolean formula consisting of the resulting set of clauses; it is this BDD_I that is employed by the algorithm outlined in the next subsection for computing 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 for 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:

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

where v_i is the value of f_i in I and $P(BDD_i)$ is the probability of the BDD as defined in the following subsection.

However, for simplifying the notation in the algorithm below we shall act as if the BDD_I also includes the variables with known truth-value in I. Internally, the probability calculation is implemented using the above theorem. Furthermore, 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. 2 the probabilistic fact burglary. It only occurs on the left path to the 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 no matter whether the missing atom has the value true or false.

4.2 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 similar to the forward/backward algorithm for HMMs or the inside/outside probability of PCFGs. We will talk about the upward/downward algorithm and probability. 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. Using $\pi_N = 1$ we indicate whether a node represents a deterministic node. For every node N in the BDD we compute:

1. the *upward probability* $P(\uparrow N)$ represents the probability that the logical formula encoded by the sub-BDD rooted at N is true. For instance, in Fig. 2, the upward probability of the leftmost node for `alarm` represents the probability of the formula `alarm` $\wedge$ `burglary`.
2. the *downward probability* $P(\downarrow N)$ 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. 2, the downward probability of the right alarm-node would be equal to the probability of $\neg$`earthquake`, $\neg$`al(john)` which is $(1-0.2)\cdot(1-0.7)$.

The following invariants hold for the BDD (excluding $N = \mathbb{O}$, due to the deterministic nodes):

$$\sum\nolimits_{N \text{ node at level } n} P(\uparrow N)\cdot P(\downarrow N) = P(BDD)$$

where the variable N ranges over all nodes occurring at a particular level n in the BDD. That means that $P(\uparrow N)\cdot P(\downarrow N)\cdot (P(BDD))^{-1}$ represents the probability that an assignment satisfying the BDD will pass through the node N. Especially it holds that $P(\downarrow \mathbb{1}) = P(\uparrow \mathit{Root}) = P(BDD)$. The upward and downward probabilities are now computed using the following formulae (cf. also Fig. 3):

$$P(\uparrow \mathbb{O}) = 0 \qquad P(\uparrow \mathbb{1}) = 1 \qquad P(\downarrow \mathit{Root}) = 1$$
$$P(\uparrow N) = P(\uparrow h(N))\cdot p_N^{\pi_N} + P(\uparrow l(N))\cdot (1-p_N)^{\pi_N}$$
$$P(\downarrow N) = \sum\nolimits_{N=h(M)} P(\downarrow M)\cdot p_M^{\pi_M} + \sum\nolimits_{N=l(M)} P(\downarrow M)\cdot (1-p_M)^{\pi_M}$$

where π_N is 0 for nodes representing deterministic nodes and 1 otherwise.

The algorithm proceeds by first computing the downward probabilities from the root to leafs and then the upward probability from leafs to the root. Intermediate results are stored and reused when nodes are revisited. The algorithm is sketched in Algorithm 1. The $\mathbb{E}[\delta^m_{n,k}|I_m]$ are then computed as

$$\mathbb{E}[\delta^m_{n,k}|I_m] = \sum\nolimits_{N \text{ represents } f_m} \frac{P(\downarrow N)\cdot p_N \cdot P(\uparrow h(N))}{P(BDD)} \quad .$$

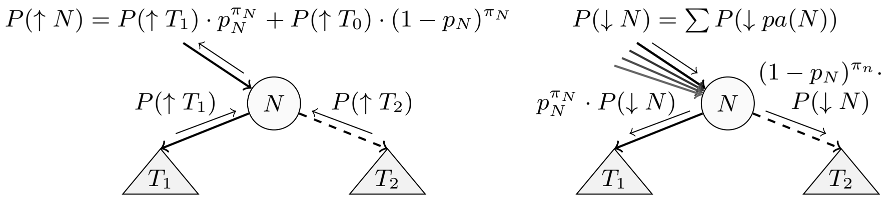
$P(\uparrow N) = P(\uparrow T_1) \cdot p_N^{\pi_N} + P(\uparrow T_0) \cdot (1 - p_N)^{\pi_N}$
$P(\uparrow T_1)$
N
$P(\uparrow T_2)$
T_1
T_2
$P(\downarrow N) = \sum P(\downarrow pa(N))$
$(1 - p_N)^{\pi_n} \cdot$
$P(\downarrow N)$
$p_N^{\pi_N} \cdot P(\downarrow N)$
N
T_1
T_2

Fig. 3. 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 zero otherwise.

Algorithm 1 Calculating the probability of a BDD

```
function Up(BDD node $n$)
    if $n$ is the 1-terminal then return 1
    if $n$ is the 0-terminal then return 0
    let $h$ and $l$ be the high and low children of $n$
    if $n$ represents probabilistic fact then
        return $p_n \cdot \text{Up}(h) + (1 - p_n) \cdot \text{Up}(l)$
    else
        return $\text{Up}(h) + \text{Up}(l)$

function Down(BDD node $n$)
    $q$ priority queues sorting according to BDD-order
    $enqueue(q, n)$
    initialize Down as array of 0's with length $height(n)$
    while $q$ not empty do
        $n = dequeue(q)$
        $pi = 1$ if $n$ probabilistic fact 0 otherwise
        $\text{Down}[h(n)] += \text{Down}[n] \cdot p_n^{pi}$
        $\text{Down}[l(n)] += \text{Down}[n] \cdot (1 - p_n)^{pi}$
        $enqueue(q, h(n))$ if not yet in $q$
        $enqueue(q, l(n))$ if not yet in $q$
```

5 Experiments

We implemented the algorithm in YAP Prolog[4] utilizing SimpleCUDD[5] for the BDD operations. All experiments were run on a 2.8Ghz computer with 8Gb memory. We performed three types of experiments. In the first we evaluated whether the CoPrEM algorithm can be used to learn which clauses are correct in a program,in the second one we used CoPrEM to learn a sequential ProbLog model and compared it to CPT-L [19].

5.1 7-segment display

To evaluate our approach we use the *LED Display Domain Data Set*[6] from the UCI repository and answer question Q_1 as to *whether* CoPrEM *can be used to determine which clauses are correct in a ProbLog program.* The set up is akin to a Bayesian structure learning approach and the artificial dataset allows us to control all properties of the problem.

The problem is to learn recognizing digits based on features stemming from a 7-segment display. Such a display contains seven light-emitting diodes (LED) arranged like an eight. By switching all LEDs to a specific configuration one can

[4] http://www.dcc.fc.up.pt/∼vsc/Yap/

[5] http://www.cs.kuleuven.be/∼theo/tools/simplecudd.html

[6] http://archive.ics.uci.edu/ml/datasets/LED+Display+Domain

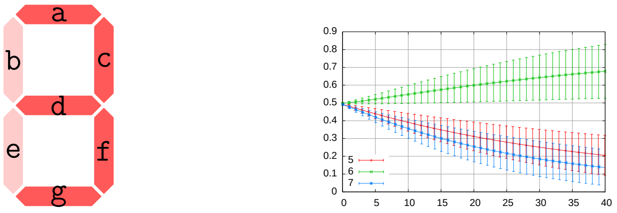

Fig. 4. The learned fact probabilities after n iterations of Soft-EM in the LED domain. For the input corresponding to a 6 the graph shows, the probability of assigning the class 5, 6 or 7. The correct rule, which predicts class 6, gets the highest probability assigned. The remaining classes have probabilities assigned. The probabilities for 5 and 7 represent the upper respectively lower bound for the left out curves.

represent a number. So this problem domain consists of seven Boolean features and 0 concepts. Solving this problem is non-trivial due to noise which flips each of the Boolean features with a certain probability. To further complicate the problem, we removed attributes including the class attribute at random. As model we considered the combination of all possible concepts (both correct and incorrect). The concepts are expressed in terms of the observed segments as given in Figure 4. For instance, the correct representation of the number 3 is:

$$\texttt{0.9 :: p99.} \qquad\qquad \texttt{predict(3) :- a}, \neg\texttt{b}, \texttt{c}, \texttt{d}, \neg\texttt{e}, \texttt{f}, \texttt{g}, \texttt{p99}.$$

Each rule represents one concept – a configuration of the segments in the display. It uses a probabilistic fact (p99 in this case) to capture the probability that this rule or concept holds. The final model contains $2^7 \times 10 = 1280$ probabilistic facts, and equally many clauses.

We generated 100 training examples. Each feature got perturbed with noise by 10% chance, and each feature had a 30% probability to be unobserved. All results are averaged over ten runs. After training we inspected manually whether the right target concept was learned (cf. Fig. 4). The results indicate that the algorithm learns the right concept even from noisy incomplete data. The runtimes are of the algorithm are as follows: the generation of the BDDs took $\approx 10sec$ in total where grounding took $\approx 4sec$ and the completion took $\approx 5sec$. As the BDDs do not change, this has to be done only once. Evaluating the BDD afterward took $\approx 2.5sec$ on average. These results affirmatively answer Q_1.

5.2 Comparisons to CPT-L

CPT-L is a formalism based on CP-Logic. It is tailored towards sequential domains. A CPT-L model consists out of a set of rules which – when applied to the current state – jointly generate the next state. A single CPT-L rule is of the form $h_1 :: p_1 \vee \ldots \vee h_n :: p_n \leftarrow b_1, \ldots, b_n$ where the p_i are probability labels (summing to 1), the h_i are atoms h_i and the b_i are literals. The rule states that for all ground instances of the rule, whenever the body is true in the current state exactly one of the heads is true. A CTP-L model essentially represents a probability distribution over sequences of relational interpretations (a kind of relational Markov model). It is possible to transform a CPT-L model into ProbLog. Furthermore, the learning setting developed for CPT-L is similar to that introduced here, except that it is tailored towards sequential data. Thus the present ProbLog setting is more general and can be applied to the CPT-L problem. This also raises the question Q_3 as to *whether the more general setting* CoPrEM *setting for ProbLog can be used on realistic sequential problems for CPT-L.*

To answer this question, we used the *Massively Multiplayer Online Game* dataset used in the CPT-L paper [19] together with the original model and translated it to ProbLog. We report only runtime results, as all other results are identical. Using the CPT-L implementation learning takes on average 0.2 minutes per iteration whereas in LFI an average iteration takes 0.1 minutes. This affirmatively answers Q_3, and shows that CoPrEM is competitive to CPT-L even so the latter is tailored towards sequential problems.

6 Related Work

There is a rich body of related work in the literature on statistical relational learning and probabilistic logic programming. First, as already stated in the introduction, current approaches to learning probabilistic logic programs for ProbLog [3], PRISM [18], and SLPs [13], almost all learn from entailment. For ProbLog, there exists a least square approach where the examples are ground facts together with their target probability [9]. This approach implements a different approach to learning that is *not* based on an underlying generative process, neither at the level of predicates nor at the level of interpretations and this explains why it requires the target probabilities to be specified. Hence, it essentially corresponds to a regression approach and contrasts with the generative learning at the level of interpretations that is pursued in the present paper.

For ProbLog, there is also the restricted study of [16] that learns acyclic ground (that is, propositional) ProbLog programs from interpretations using a transformation to Bayesian networks. Such a transformation is also employed in the learning approaches for CP-logic [20]. Sato and Kameya have contributed various interesting and advanced learning algorithms that have been incorporated in PRISM, but all of them learn from entailment. Ishihata *et al.* [10] mention the possibility of using a BDD-like approach in PRISM but have – to the best of the authors' knowledge – not yet implemented such an approach. Furthermore, implementing this would require a significant extension of PRISM to relax the exclusive explanation assumption. This assumption made by PRISM – but not by ProbLog – states that different proofs for a particular fact should be disjoint, that is, not overlap. It allows PRISM to avoid the use of BDDs and optimizes both learning and inference algorithms. They also do not specify how to obtain the BDDs from PRISM. The upward/downward procedure used in our algorithm to estimate the parameters from a set of BDDs is essentially an extension of that of the approaches of Ishihata *et al.* [10] and of Thon *et al.* [19] that have been independently developed. The algorithm of Ishihiata et al. learns the probability of literals for arbitrary Boolean formulae from examples using a BDD while that of Thon [19] has been tailored towards learning with BDDs for the sequential logic CPT-L. One important extension in the present algorithm, as compared to [10], is that it can deal with deterministic nodes and dependencies that may occur in BDDs, such as those generated by our algorithm.

Our approach can also be related to the work on knowledge-based model construction approaches in statistical relational learning such as BLPs, PRMs and MLNs [15]. While the setting in the present paper is the standard setting for these kind of representations, there are significant differences at the algorithmic level, beside the representational. First, for BLPs, PRMs, and also CP-logic, typically for each training example a ground Bayesian network is constructed where a standard learning algorithm can be used. Second, for MLNs, many different learning algorithms have been developed. Even so the representation generated by Clark's completion - a weighted ALL-SAT problem - is quite close to the representation of Markov Logic, there are subtle differences. We use probabilities on single facts, while Markov Logic uses weights on clauses. This generates a clear probabilistic semantics. It is unclear whether and how BDDs could be incorporated in Markov Logic, but seems to be a promising future research direction.

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 such as MLNs, PRMs, BLPs, and probabilistic logic programming approaches using Sato's distribution semantics such as ProbLog and PRISM. The CoPrEM algorithm tightly couples logical inference with a probabilistic EM algorithm at the level of BDDs. The approach was experimentally evaluated and it was shown that the approach essentially can be applied to the typical kind of problems that have made the knowledge-based model construction approach in statistical relational learning so popular. The authors hope that this contributes towards bridging the gap between the knowledge-based model construction approach and the probabilistic logic programming approach based on the distribution semantics.

Acknowledgements

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's Seventh Framework Programme under grant agreement First-MM-248258.

References

1. Cussens, J.: Parameter estimation in stochastic logic programs. Machine Learning 44(3), 245–271 (2001)
2. De Raedt, L., Frasconi, P., Kersting, K., Muggleton, S. (eds.): Probabilistic Inductive Logic Programming — Theory and Applications, Lecture Notes in Artificial Intelligence, vol. 4911. Springer (2008)
3. De Raedt, L., Kimmig, A., Toivonen, H.: Problog: A probabilistic Prolog and its application in link discovery. In: Veloso, M. (ed.) Proceedings of the 20th International Joint Conference on Artificial Intelligence. pp. 2462–2467 (2007)
4. De Raedt, L.: Logical and Relational Learning. Springer (2008)
5. De Raedt, L., Kersting, K.: Probabilistic inductive logic programming. In: Algorithmic Learning Theory. pp. 19–36. No. 3244 in LNCS, Springer (2004)
6. Flach, P.: Simply logical : intelligent reasoning by example. Wiley (1994)
7. 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 (2001)
8. Getoor, L., Taskar, B. (eds.): An Introduction to Statistical Relational Learning. MIT Press (2007)
9. 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 2008. LNCS, vol. 5211, pp. 473–488. Springer (2008)
10. Ishihata, M., Kameya, Y., Sato, T., Minato, S.: Propositionalizing the EM algorithm by BDDs. In: ILP (2008)
11. Jensen, F.: Bayesian Networks and Decision Graphs. Springer (2001)
12. Kersting, K., De Raedt, L.: Bayesian logic programming: theory and tool. In: Getoor, L., Taskar, B. (eds.) An Introduction to Statistical Relational Learning. MIT Press (2007)

13. 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 (1996)
14. Poole, D.: The independent choice logic and beyond. In: Raedt, L.D., Frasconi, P., Kersting, K., Muggleton, S. (eds.) Probabilistic Inductive Logic Programming. Lecture Notes in Computer Science, vol. 4911, pp. 222–243. Springer (2008)
15. Richardson, M., Domingos, P.: Markov logic networks. Machine Learning 62, 107–136 (2006)
16. Riguzzi, F.: Learning ground problog programs from interpretations. In: Proceedings of the 6th Workshop on Multi-Relational Data Mining (MRDM07) (2007)
17. 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 (1995)
18. Sato, T., Kameya, Y.: Parameter learning of logic programs for symbolic-statistical modeling. Journal of Artificial Intelligence Research 15, 391–454 (2001)
19. 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. LNCS, vol. 5211, pp. 506–521. Springer (2008)
20. 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.) Proceedings of the 10th European Conference on Logics in Artificial Intelligence. LNCS, vol. 4160, pp. 452–464. Springer (2006)