# Parameter Learning in ProbLog with Annotated Disjunctions

Wen-Chi Yang[1(✉)], Arcchit Jain[1], Luc De Raedt[1,2], and Wannes Meert[1]

[1] Department of Computer Science, Leuven.AI, KU Leuven, Celestijnenlaan 200a - box 2402 3001, Leuven, Belgium
{wenchi.yang,luc.deraedt,wannes.meert}@kuleuven.be, arcchit.jain.2015@iitkalumni.org
[2] Centre for Applied Autonomous Sensor Systems, Örebro University, Örebro, Sweden

**Abstract.** In parameter learning, a partial interpretation most often contains information about only *a subset of* the parameters in the program. However, standard EM-based algorithms use all interpretations to learn all parameters, which significantly slows down learning. To tackle this issue, we introduce EMPLiFI, an EM-based parameter learning technique for probabilistic logic programs, that improves the efficiency of EM by exploiting the rule-based structure of logic programs. In addition, EMPLiFI enables parameter learning of multi-head annotated disjunctions in ProbLog programs, which was not yet possible in previous methods. Theoretically, we show that EMPLiFI is correct. Empirically, we compare EMPLiFI to LFI-ProbLog and EMBLEM. The results show that EMPLiFI is the most efficient in learning single-head annotated disjunctions. In learning multi-head annotated disjunctions, EMPLiFI is more accurate than EMBLEM, while LFI-ProbLog cannot handle this task.

**Keywords:** Learning from interpretations · Probabilistic logic programming · Expectation maximization

## 1 Introduction

Statistical relational learning [8] and Probabilistic Logic Programming [3,4] have contributed to various representations and learning schemes that reason about objects and uncertain relational structures among them. Popular approaches include PRISM [10], Independent Choice Logic [13], Bayesian Logic Programs [11], Markov Logic Networks [14], Logic Programs with Annotated Disjunctions [17], CP-Logic [16] and ProbLog [7]. Many of these languages are based on variants of the distribution semantics [15]. They vary in the way they define the distribution over logic programs but are equally expressive [2]. In this paper, we use ProbLog's representation. ProbLog has probabilistic facts such as `0.01 :: earthquake`, stating that the probability of having


T. Bouadi et al. (Eds.): IDA 2022, LNCS 13205, pp. 378–391, 2022.
https://doi.org/10.1007/978-3-031-01333-1_30

an earthquake is 0.01, and has clauses such as `alarm :- earthquake`, stating that the alarm goes off if there is an earthquake. In addition, ProbLog supports annotated disjunctions (ADs) such as `0.01 :: alarm(long); 0.19 :: alarm(short); 0.8 :: alarm(none)`, stating that an alarm has exactly one type of the three types.

ProbLog's parameter learning approach, LFI-ProbLog, is designed only for probabilistic facts, and not for ADs. Hence, LFI-ProbLog cannot learn multi-head AD variables. Even though LFI-ProbLog can learn single-head AD parameters, we will show that it is inefficient and in extreme cases, results in incorrect values. Faria et al. tackled a special case of this efficiency issue for single-head ADs [6]. In contrast, we provide a more general solution that, in addition, also covers multi-head ADs. Although our approach is implemented in ProbLog, it can be applied to other EM-based parameter learning algorithms as what we exploit is the rule-based structure that is shared by all probabilistic logic programs.

The contribution is twofold. First, we introduce EMPLiFI, a new parameter learning approach in ProbLog. EMPLiFI correctly learns multi-head ADs and speeds up learning by exploiting the rule-based structure of logic programs. Second, we prove that EMPLiFI is correct and illustrate how it reduces EM iterations. We compare EMPLiFI with two other EM-based learners, LFI-ProbLog and EMBLEM, and show that EMPLiFI is the most accurate in learning multi-head ADs and takes the fewest EM iterations to converge.

## 2 Preliminaries

**Probabilistic Logic Programming.** A ProbLog theory (or program) $\mathcal{T}$ consists of a finite set of probabilistic facts $\mathcal{F}$, a finite set of clauses $\mathcal{BK}$ and a finite set of annotated disjunctions $\mathcal{AD}$. A *probabilistic fact* is an expression $\mathtt{p} :: \mathtt{f}$ that states the ground fact $\mathtt{f}$ is true with probability $\mathtt{p}$. A *clause* is an expression $\mathtt{h} :- \mathtt{b_1}, \cdots, \mathtt{b_n}$ where $\mathtt{h}$ is a literal and $\mathtt{b_1}, \cdots, \mathtt{b_n}$ is a conjunction of literals, stating $\mathtt{h}$ is true if $\mathtt{b_1}, \cdots, \mathtt{b_n}$ is true. ProbLog defines probability distributions over ground facts in a Herbrand base $L_{\mathcal{T}}$. The probabilistic facts define a probability distribution over *possible worlds*. All ground facts in a possible world $W$ are *true* and all that are not in $W$ are *false*. The probability of a possible world $W$ is defined as $P(W|\mathcal{T}) = \prod_{\mathtt{f_i} \in W} \mathtt{p_i} \prod_{\mathtt{f_i} \in L_{\mathcal{T}} \setminus W} (1 - \mathtt{p_i})$. The *success probability* of a query $q$ is the sum of the probabilities of the possible worlds that entail $q$, formally, $P_s(q|\mathcal{T}) = \sum_{I \subseteq L_{\mathcal{T}}, I \models q} P(I|\mathcal{T})$. A *partial interpretation* $I$ is an incomplete possible world that contains truth values of some (but not all) atoms. If an atom $\mathtt{a}$ (resp. $\neg\mathtt{a}$) is in $I$, then $\mathtt{a}$ is true (resp. false). Otherwise, the truth value of $\mathtt{a}$ is unknown. Hence, a partial interpretation $I$ represents a number of possible worlds, and the probability of $I$ is the success probability of the conjunction of the literals in $I$, i.e. $P(I) = P_s(\bigwedge_{l \in I} l)$.

**Annotated Disjunctions in ProbLog.** An *annotated disjunctions* (ADs) is a clause with one or more mutually exclusive heads of the form $\mathtt{p_1} :: \mathtt{h_1}; \cdots ; \mathtt{p_k} :: \mathtt{h_k}$ where $\sum_{i=1}^{k} \mathtt{p_i} = 1$, stating that if the body is true, exactly one head is made true,

where the choice of $\mathtt{h_i}$ is governed by $\mathtt{p_i}$. Although the ProbLog language and semantics allow for ADs, only probabilistic facts (and a transformation) are used for inference. Hence, most transformations encode ADs as probabilistic facts as the first step [5,7]. For example, a three-head AD $\mathtt{0.2 :: a1; 0.2 :: a2; 0.6 :: a3 :\!-b}$ is encoded as

```
0.2::h1.                0.25::h2.                1.0::h3.
a1:-b,h1.               a2:-b,\+h1,h2.           a3:-b,\+h1,\+h2,h3.
```

where `h1`, `h2` and `h3` are hidden facts. The last fact `h3` can be dropped for inference but we keep it in because we will later need it for learning. This encoding is designed to compute the probabilities correctly for the inference task but is insufficient for learning and results in incorrect values (see Sect. 3).

## 3 Learning from Interpretations in ProbLog

In this section, we review LFI-ProbLog and illustrate its two issues. Later, Sect. 4 will introduce a new learning approach that resolves these issues. The parameter learning task in ProbLog is as below.

**Given**

- A ProbLog program $\mathcal{T}(\mathbf{p}) = \mathcal{F} \cup \mathcal{BK} \cup \mathcal{AD}$ where $\mathcal{F}$ is a set of probabilistic facts, $\mathcal{BK}$ is a set of background knowledge and $\mathcal{AD}$ is a set of ADs. $\mathbf{p} = \langle p_1, \cdots, p_N \rangle$ is a set of unknown parameters where each parameter is attached to a probabilistic fact or an AD head.
- A set $\mathcal{I}$ of partial interpretations $\{I_1, ..., I_M\}$.

**Find** maximum likelihood probabilities $\widehat{\mathbf{p}}$ for all interpretations in $\mathcal{I}$. Formally,

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

Given initial parameters $\mathbf{p}^0 = \langle p_1^0, \cdots, p_N^0 \rangle$, an Expectation Maximization (EM) algorithm computes $\mathbf{p}^1$, and in this fashion, enumerates a series of estimations $\mathbf{p}^2, ..., \mathbf{p}^T$. The process terminates after $T$ iterations when the log likelihood does not improve more than an arbitrary small value $\epsilon$. LFI-ProbLog, is summarized by Eq. 1 [7,9], which takes $\mathbf{p}^t$ to compute $\mathbf{p}^{t+1}$. Intuitively, based on $\mathbf{p}^t$, a new estimate $p_n^{t+1}$ is the expected count of $\mathtt{f_n}$ being true divided by the total count of $\mathtt{f_n}$, formally,

$$p_n^{t+1} = \frac{\sum_{m=1}^{M} \sum_{k=1}^{K_n^m} P(\mathtt{f_{n,k}}|I_m, \mathcal{T}(\mathbf{p}^t))}{\sum_{m=1}^{M} K_n^m} \tag{1}$$

where $K_n^m$ is the number of ground instances represented by $\mathtt{p_n :: f_n}$ in $I_m$. We will use the following running examples throughout this paper to illustrate two issues of LFI-ProbLog and our approach to tackle them.

*Running example 1.* Consider the following *Smokers* program with three parameters $\mathbf{p} = \langle \mathtt{p}_1, \mathtt{p}_2, \mathtt{p}_3 \rangle$, stating that a person is a smoker with probability $\mathtt{p}_1$, and any smoker (resp. non-smoker) has cancer with probability $\mathtt{p}_2$ (resp. $\mathtt{p}_3$).

```
person(X).                                  p1::smokes(X):-person(X).
p2::cancer(X):-smokes(X),person(X).         p3::cancer(X):-\+smokes(X),person(X).
```

Consider interpretations $I_1 = \{\mathtt{smokes(a)}, \mathtt{cancer(a)}\}$, $I_2 = \{\mathtt{smokes(b)}, \neg \mathtt{cancer(b)}\}$, $I_3 = \{\neg\mathtt{smokes(c)}, \mathtt{cancer(c)}\}$, $I_4 = \cdots = I_{102} = \{\neg\mathtt{smokes}(\cdot), \neg\mathtt{cancer}(\cdot)\}$. As all interpretations are fully observable, we obtain $\langle \mathtt{p}_1, \mathtt{p}_2, \mathtt{p}_3 \rangle = \langle 2/102, 1/2, 1/100 \rangle$ by simply counting.

*Running example 2.* Consider the following *Colors* program with three parameters $\mathbf{p} = \langle \mathtt{p}_1, \mathtt{p}_2, \mathtt{p}_3 \rangle$ that jointly denote a probability distribution of the color of a ball.

```
p1::green;p2::red;p3::blue:-ball.                ball.
```

Given interpretations $I_1 = \{\mathtt{green}\}$, $I_2 = \{\mathtt{red}\}$ and $I_3 = \{\mathtt{blue}\}$, we obtain $\mathbf{p} = \langle 1/3, 1/3, 1/3 \rangle$ by counting.

The first issue of LFI-ProbLog is efficiency-related. When learning single-head ADs, LFI-ProbLog takes into account *all* interpretations, including the irrelevant ones that do not contain information about the parameter to be learned. These irrelevant interpretations introduce an undesired inertia in EM learning, as illustrated in Example 1.

*Example 1.* For LFI-ProbLog, the Smokers program must be transformed into the following program with hidden facts $\mathtt{h}_1, \mathtt{h}_2$ and $\mathtt{h}_3$.

```
p1::h1.          p2::h2.          p3::h3.          person(X).
smokes(X):-person(X) h1.
cancer(X):-smokes(X),person(X),h2.    cancer(X):-\+smokes(X),person(X),h3.
```

Given initial parameters $\mathbf{p}^0 = \langle 0.1, 0.1, 0.1 \rangle$, by applying Eq. 1, we obtain $\mathbf{p}^1 = \langle p_1^1, p_2^1, p_3^1 \rangle$ as follows.

$$p_1^1 = \frac{1+1+0+0\times 99}{102} = \frac{2}{102}, p_2^1 = \frac{1+0+.1+.1\times 99}{102} = 0.108, p_3^1 = \frac{.1+.1+1+0\times 99}{102} = 0.012$$

By repeatedly applying Eq. 1, we further obtain a series of estimates $\mathbf{p}^2 = \langle 2/102, 0.116, 0.01 \rangle$, $\cdots$, $\mathbf{p}^{100} = \langle 2/102, 0.445, 0.01 \rangle$. Notice that $\mathtt{p}_2$ does not converge to 0.5 after 100 iterations even though all interpretations are fully observable. This is resulted from the irrelevant interpretations $I_3, \cdots I_{102}$.

The second issue is that LFI-ProbLog does not correctly learn all possible multi-head ADs. This is because how their probabilities are transformed from $\mathbf{p}$ to $\acute{\mathbf{p}}$ [5] (cf. Sect. 2). This transformation is incorrect in learning as the parameters are not known and must be learned, as illustrated in Example 2.

*Example 2.* For LFI-ProbLog, the Colors program must be transformed into the following program with hidden facts `gh`, `rh` and `bh`, and the given initial parameters must be transformed. Say, given $\mathbf{p}^0 = \langle 0.2, 0.2, 0.6 \rangle$, the transformed probabilities are $\acute{\mathbf{p}}^0 = \langle 0.2, 0.25, 1.0 \rangle$.

```
ṕ1::gh.                ṕ2::rh.                ṕ3::bh.              ball.
green:-ball,gh.        red:-ball,\+gh,rh.     blue:-ball,\+gh,\+rh,bh.
```

By applying Eq. 1, we obtain $\acute{\mathbf{p}}^1$ as follows.

$$\acute{p}_1^1 = \frac{1+0+0}{3} = 0.333 \qquad \acute{p}_2^1 = \frac{0.25+1+1}{3} = 0.750 \qquad \acute{p}_3^1 = \frac{1+1+1}{3} = 1$$

We future obtain $\acute{\mathbf{p}}^2 = \langle 0.333, 0.917, 1 \rangle, \cdots, \acute{\mathbf{p}}^{10} = \langle 0.333, 1, 1 \rangle$, which corresponds to the incorrect AD parameters $\mathbf{p}^{10} = \langle 1/3, 2/3, 0 \rangle$.

## 4 Learning with Annotated Disjunctions

We propose EMPLiFI, a parameter learning approach in Eq. 2, as a solution to the issues discussed in Sect. 3. This section illustrates EMPLiFI, and Sect. 5 will prove EMPLiFI's correctness.

$$p_n^{t+1} = \frac{\sum_{I_m \in I_{\mathtt{p_n}}} \sum_{j=1}^{J_n^m} P(\mathtt{h_{n,j}}, \mathtt{b_{n,j}} | I_m, \mathcal{T}(\mathbf{p}^t))}{\sum_{I_m \in I_{\mathtt{p_n}}} \sum_{j=1}^{J_n^m} P(\mathtt{b_{n,j}} | I_m, \mathcal{T}(\mathbf{p}^t))} \tag{2}$$

where

- $\mathtt{h_{n,j}}$ and $\mathtt{b_{n,j}}$ are the j-th possible ground instance represented by $\mathtt{p_n} :: \mathtt{h_n}$ and the corresponding body in $I_m$
- $J_n^m$ is the total number of ground instances represented by $\mathtt{p_n} :: \mathtt{h_n}$ in $I_m$
- $I_{\mathtt{p_n}}$ is the set of all *relevant interpretations* to $\mathtt{p_n}$

If the denominator is zero, then $p_n^{t+1}$ will not be updated. Intuitively, based on $\mathbf{p}^t$, a new estimate $p_n^{t+1}$ is the expected count of the head divided by the expected count of the body. Unlike LFI-ProbLog that assumes all parameters are attached to a fact, EMPLiFI recognizes and exploits AD rule structures, which enables efficient EM and multi-head AD learning.

### 4.1 Relevant Interpretations

At this point, it is important to stress that some interpretations do not contain information about a rule $\mathtt{p} :: \mathtt{h} :\!\!-\mathtt{b}$. An interpretation $I$ is called irrelevant to $\mathtt{p}$ if the conditional probability components $P(\mathtt{h_n}, \mathtt{b_n} | \cdot)$ and $P(\mathtt{b_n} | \cdot)$ solely depend on the old probability estimate $\mathbf{p}^t$, formally,

$$P(\mathtt{h}, \mathtt{b} | I, \mathcal{T}(\mathbf{p}^t)) = P(\mathtt{h}, \mathtt{b} | \mathcal{T}(\mathbf{p}^t)) \text{ and } P(\mathtt{b} | I, \mathcal{T}(\mathbf{p}^t)) = P(\mathtt{b} | \mathcal{T}(\mathbf{p}^t)) \tag{3}$$

As irrelevant interpretations slow down learning (see Example 1), it is our aim to identify them for each parameter.

The *dependency set* of a ground atom $\mathtt{a}$ in a ProbLog theory $\mathcal{T}$, denoted by $dep_{\mathcal{T}}(\mathtt{a})$, is the set of all atoms that occur in some SLD-proof of $\mathtt{a}$ [7]. The dependency set of multiple atoms is the union of their dependency sets. A ground fact $\mathtt{f} \in \mathcal{T}$ is relevant to an interpretation $I$ if it is in the dependency set of $I$, namely $\mathtt{f} \in dep_{\mathcal{T}}(I)$. Similarly, a ground clause is relevant to $I$ if it is used in the SLD proof of $I$. Then, the *interpretation-restricted theory*, denoted by $\mathcal{T}_r(I)$, is the union of all relevant facts and clauses [9]. A restricted theory is a subset of the original ground program, in fact, it is usually much smaller than the original program. Using the restricted theory, we can define relevant interpretations for learning a parameter.

**Definition 1** ***(Relevant Interpretation)*** *For a ProbLog theory $\mathcal{T}$, an interpretation $I$ is relevant to an atom $\mathtt{a_n} \in \mathcal{T}$ if and only if $\mathtt{a_n}$ is in the interpretation-restricted theory $\mathcal{T}_r(I)$, namely $\mathtt{a_n} \in \mathcal{T}_r(I)$.*

Since a parameter $\mathtt{p_n}$ always corresponds a unique atom $\mathtt{a_n}$ in $\mathcal{T}$, we define $\mathtt{p_n}$-relevant interpretations using $\mathtt{a_n}$, formally, $\mathcal{I}_{\mathtt{p_n}} = \{I \in \mathcal{I} | \mathtt{a_n} \in \mathcal{T}_r(I)\}$. We have defined relevant interpretations for single-head ADs and probabilistic facts.

*Example 3.* Consider the Smokers program and $I_2 = \{\mathtt{smokers(b)}, \neg\mathtt{cancer(b)}\}$, the dependency set of $I_2$ is $dep_{\mathcal{T}}(I_2) = \{\mathtt{h1}, \mathtt{h2}, \mathtt{smokes(b)}, \mathtt{cancer(b)}\}$ and the corresponding restricted theory $\mathcal{T}_r(I_2)$ is

```
p1::h1.                          p2::h2.                          person(b).
smokes(b):-person(b),h1.         cancer(b):-smokes(b),person(b),h2.
```

Therefore, $I_2$ is relevant to $\mathtt{p_1}$ and $\mathtt{p_2}$ according to Definition 1. We obtain the relevant interpretation sets for all three parameters as $\mathcal{I}_{\mathtt{p_1}} = \{I_1, \cdots, I_{102}\}, \mathcal{I}_{\mathtt{p_2}} = \{I_1, I_2\}$, and $\mathcal{I}_{\mathtt{p_3}} = \{I_3, \cdots, I_{102}\}$. Given initial parameters $\mathbf{p}^0 = \langle 0.1, 0.1, 0.1 \rangle$, we obtain $\mathbf{p}^1 = \langle 2/102, 1/2, 1/100 \rangle$ after one EM iteration by applying Eq. 2, as opposed to Example 1.

### 4.2 Directly Learning Multi-head ADs

Recall that ProbLog's transformations result in incorrect learning of multi-head ADs (see Sect. 3). To gain correctness, it is required to maintain mutual exclusivity in the interpretation-restricted theory. To do so, we define the *AD dependency set*, $dep^{AD}_{\mathcal{T}}(I) \supseteq dep_{\mathcal{T}}(I)$ to include also mutually exclusive atoms. Intuitively, if $dep^{AD}_{\mathcal{T}}(I)$ contains an AD head, it must also contain all mutually exclusive heads and their dependency sets. Then, an AD dependency set defines an *AD interpretation-restricted theory* as in Sect. 4.1.

**Definition 2** ***(Relevant Interpretation with AD).*** *For a ProbLog theory $\mathcal{T}$, an interpretation $I$ is relevant to an atom $\mathtt{a_n} \in \mathcal{T}$ if and only if $\mathtt{a_n}$ is in the AD interpretation-restricted theory $\mathcal{T}^{AD}_r(I)$, namely $\mathtt{a_n} \in \mathcal{T}^{AD}_r(I)$.*

After defining relevant interpretations for ADs, we can now learn multi-head ADs using Eq. 2.

*Example 4.* Consider the Colors program and $I_1 = \{\texttt{green}\}$, the AD dependency set $dep_{\mathcal{T}}^{AD}(I_1)$ is $\{\texttt{ball}, \texttt{green}, \texttt{red}, \texttt{blue}, \texttt{gh}, \texttt{rh}, \texttt{bh}\}$ and the AD restricted theory $\mathcal{T}_r^{AD}(I_1)$ is

```
p1::gh.                 p2::rh.                 p3::bh.          ball.
green:-ball,gh.         red:-ball,\+gh,rh.      blue:-ball,\+gh,\+rh,bh.
```

$I_1$ is relevant to $\texttt{p}_1$, $\texttt{p}_2$ and $\texttt{p}_3$ according to Definition 2. Similarly, $I_2$ and $I_3$ are also relevant to all three parameters. Given initial parameters $\mathbf{p}^0 = \langle 0.2, 0.2, 0.6 \rangle$, we obtain $\mathbf{p}^1 = \langle 1/3, 1/3, 1/3 \rangle$ after one EM iteration by applying Eq. 2.

## 5 Proofs

Section 5.1 will prove EMPLiFI's correctness and Sect. 5.2 will provide insight into how EMPLiFI improves efficiency of EM parameter learning.

### 5.1 Correctness

We start from the EM algorithm for Bayesian Networks [12], i.e. Eq. 4, that differs from EMPLiFI by learning from all interpretations. Since Eq. 4 is correct [12], we can prove the correctness of EMPLiFI, i.e. Eq. 2, by showing they converge to the same values, i.e. Proposition 1.

$$p_n^{t+1} = \frac{\sum_{m=1}^{M} \sum_{j=1}^{J_n^m} P(\mathtt{h}_{\mathtt{n,j}}, \mathtt{b}_{\mathtt{n,j}} | \mathtt{I}_{\mathtt{m}}, \mathcal{T}(\mathbf{p}^{\mathtt{t}}))}{\sum_{m=1}^{M} \sum_{j=1}^{J_n^m} P(\mathtt{b}_{\mathtt{n,j}} | \mathtt{I}_{\mathtt{m}}, \mathcal{T}(\mathbf{p}^{\mathtt{t}}))} \tag{4}$$

**Proposition 1.** *Given a program $\mathcal{T}$, a set of partial interpretations $\mathcal{I}$, and initial parameters $\mathbf{p}^0$. Let $\mathbf{p}^{t,1}$ and $\mathbf{p}^{t,2}$ be the parameter estimates generated by Eqs. 2 and 4, respectively. It is true then* $\lim_{t\to\infty} p_n^{t,1} = p_n^{t,2}$

*Proof.* We prove by induction. When $t = 0$, $\mathbf{p}^{0,1} = \mathbf{p}^{0,2} = \mathbf{p}^0$ holds. Assume that when $t = k$, $\mathbf{p}^{k,1} = \mathbf{p}^{k,2}$ holds. Then, for $t = k+1$, by applying Eq. 2, we obtain $p_n^{k+1,1}$, which we rewrite using $A$ and $B$ to save space.

$$p_n^{k+1,1} = \frac{\sum_{I \in \mathcal{I}_{\mathtt{p_n}}} \sum_{j=1}^{J_n^m} P(\mathtt{h}_{\mathtt{n,j}}, \mathtt{b}_{\mathtt{n,j}} | I, \mathcal{T}(\mathbf{p}^{k,1}))}{\sum_{I \in \mathcal{I}_{\mathtt{P_n}}} \sum_{j=1}^{J_n^m} P(\mathtt{b}_{\mathtt{n,j}} | I, \mathcal{T}(\mathbf{p}^{k,1}))} = \frac{A}{B} \tag{5}$$

We finish the proof by showing that $p_n^{k+1,2}$ also converges to $\frac{A}{B}$.

$$
\begin{aligned}
p_n^{k+1,2} &= \frac{\sum_{m=1}^{M}\sum_{j=1}^{J_n^m} P(\mathtt{h_{n,j}}, \mathtt{b_{n,j}} | I_m, \mathcal{T}(\mathbf{p}^{k,2}))}{\sum_{m=1}^{M}\sum_{j=1}^{J_n^m} P(\mathtt{b_{n,j}} | I_m, \mathcal{T}(\mathbf{p}^{k,2}))} \ (\because \text{ Equation 4}) \\
&= \frac{\sum_{m=1}^{M}\sum_{j=1}^{J_n^m} P(\mathtt{h_{n,j}}, \mathtt{b_{n,j}} | I_m, \mathcal{T}(\mathbf{p}^{k,1}))}{\sum_{m=1}^{M}\sum_{j=1}^{J_n^m} P(\mathtt{b_{n,j}} | I_m, \mathcal{T}(\mathbf{p}^{k,1}))} \ (\because \mathbf{p}^{k,1} = \mathbf{p}^{k,2}) \\
&= \frac{A + \sum_{I \notin \mathcal{I}_{\mathtt{p_n}}}\sum_{j=1}^{J_n^m} P(\mathtt{h_{n,j}}, \mathtt{b_{n,j}} | I, \mathcal{T}(\mathbf{p}^{k,1}))}{B + \sum_{I \notin \mathcal{I}_{\mathtt{p_n}}}\sum_{j=1}^{J_n^m} P(\mathtt{b_{n,j}} | I, \mathcal{T}(\mathbf{p}^{k,1}))} (\because \text{Equation 5}) \\
&= \frac{A + \sum_{I \notin \mathcal{I}_{\mathtt{p_n}}}\sum_{j=1}^{J_n^m} P(\mathtt{h_{n,j}}, \mathtt{b_{n,j}} | \mathcal{T}(\mathbf{p}^{k,1}))}{B + \sum_{I \notin \mathcal{I}_{\mathtt{p_n}}}\sum_{j=1}^{J_n^m} P(\mathtt{b_{n,j}} | \mathcal{T}(\mathbf{p}^{k,1}))} (\because \text{Equation 3}) \\
&= \frac{A + M_2 \times P(\mathtt{h_n}, \mathtt{b_n} | \mathcal{T}(\mathbf{p}^{k,2}))}{B + M_2 \times P(\mathtt{b_n} | \mathcal{T}(\mathbf{p}^{k,2}))} \ (\text{Let} M_2 = \sum_{I \notin \mathcal{I}_{\mathtt{p_n}}} J_n^m \text{ and } \because \mathbf{p}^{k,1} = \mathbf{p}^{k,2}) \\
&= \frac{A + M_2 \times P(\mathtt{h_n}, \mathtt{b_n} | \mathcal{T}(\mathbf{p}^{k+1,2}))}{B + M_2 \times P(\mathtt{b_n} | \mathcal{T}(\mathbf{p}^{k+1,2}))} \ (\because \mathbf{p}^{k,2} = \mathbf{p}^{k+1,2}) \qquad (6)
\end{aligned}
$$

By definition,

$$
p_n^{k+1,2} = \frac{P(\mathtt{h_n}, \mathtt{b_n} | \mathcal{T}(\mathbf{p}^{k+1,2}))}{P(\mathtt{b_n} | \mathcal{T}(\mathbf{p}^{k+1,2}))} \qquad (7)
$$

By combining Eqs. 6 and 7, we obtain $p_n^{k+1,2} = \frac{A}{B}$.

### 5.2 Convergence Rate

We will prove that EMPLiFI always updates the parameters by a larger margin by considering only relevant interpretations, namely Proposition 2.

**Proposition 2.** *Given a program $\mathcal{T}$, a set of interpretations $\mathcal{I}$, and parameter estimates $\mathbf{p}^t$. Let $\mathbf{p}^{t+1,1}$ and $\mathbf{p}^{t+1,2}$ be the next parameter estimates generated by Eqs. 2 and 4, respectively. It is true that either $p_n^{t+1,1} \leq p_n^{t+1,2} \leq p_n^t$ or $p_n^{t+1,1} \geq p_n^{t+1,2} \geq p_n^t$ holds.*

*Proof.* Following the same reasoning as in Proposition 1, we have

$$
p_n^{t+1,1} = \frac{A}{B} \text{ and } p_n^{t+1,2} = \frac{A + M_2 \times P(\mathtt{h_n}, \mathtt{b_n} | \mathcal{T}(\mathbf{p}^t))}{B + M_2 \times P(\mathtt{b_n} | \mathcal{T}(\mathbf{p}^t))}
$$

We also have $P(\mathtt{h_n}, \mathtt{b_n} | \mathcal{T}(\mathbf{p}^t)) = p_n^t \times P(\mathtt{b_n} | \mathcal{T}(\mathbf{p}^t))$ by definition. Hence,

$$
p_n^{t+1,2} = \frac{p_n^{t+1,1} \times B + M_2 \times p_n^t \times P(\mathtt{b_n} | \mathcal{T}(\mathbf{p}^t))}{B + M_2 \times P(\mathtt{b_n} | \mathcal{T}(\mathbf{p}^t))} = \frac{p_n^{t+1,1} \times B + p_n^t \times C}{B + C} \qquad (8)
$$

where $C = M_2 \times P(\mathtt{b_n} | \mathcal{T}(\mathbf{p}^t))$. As $B, C \geq 0$, we have proven Proposition 2.

Equation 8 illustrates that irrelevant interpretations create an inertia, namely $p_n^t \times C$, where $p_n^t$ is the old estimate and $C$ is proportional to the number of irrelevant instances. This inertia not only slows down learning, but also causes numerical instability and results in incorrect values when $C >> B$ as standard EM terminates before reaching the true probabilities (cf. Example 1).

## 6 Experiments

There are two well-known parameter learning algorithms and implementations for probabilistic logic programming: EMBLEM [1] and LFI-ProbLog [7,9]. We compare EMPLiFI to these two learners to answer the following questions.

**Q1** How much does EMPLiFI speed up EM learning?
**Q2** How well does EMPLiFI handle multi-head ADs?
**Q3** How well does EMPLiFI handle missing data?
**Q4** Does EMPLiFI require more computational resources?

### Programs

*Emergency Power Supply (EPS)* [18] is propositional, acyclic and contains 24 probabilities[1]. It can be handled by all learner as it has no multi-head ADs.

```
0.95::lowSupply:-a1.
1.0::lowSupply:-highSupply.
0.95::failure:-highLoad,\+highSupply.
0.95::failure:-lowLoad,\+lowSupply.
0.98::emergency:-\+a3,\+a4.
0.7::ll1:-emergency.
0.7::pl1:-emergency.
0.6::highLoad:-ll2,ll3,pl2,pl3.
0.95::lowLoad:-highLoad.
0.8::lowLoad:-ll1, pl1.
0.8::lowLoad:-ll2, pl2.
0.8::lowLoad:-ll3, pl3.
0.95::highSupply:-a2,a3.
0.95::highSupply:-a2,a4.
0.95::highSupply:-a3,a4.
0.75::a2:-a3.
0.75::a2:-a4.
0.85::a1.
0.95::a3.
0.95::a4.
0.8::ll2.
0.8::pl2.
0.8::ll3.
0.8::pl3.
```

*Smokers* [7] contains 4 probabilities. It has no multi-head ADs but is relational and cyclic. We omit ground facts `person/1` and `friend/2` to save space.

```
0.2::smokes(X):-person(X).
0.3::smokes(X):-friend(X,Y),smokes(Y),person(X),person(Y),X\=Y.
0.3::cancer(X):-smokes(X),person(X).
0.1::cancer(X):-\+smokes(X),person(X).
```

*Dice* is an AD with 6 heads. The dice has a higher change of throwing a six.

```
0.15::one;0.15::two;0.15::three;0.15::four;0.15::five;0.25::six.
```

[1] http://www.machineryspaces.com/emergency-power-supply.html.

*Colors* consists of a single-head AD and two multi-head ADs. To learn this program, one must perform EM, even in the fully observable case because the AD bodies are not mutually exclusive.

```
ball.          0.8::green:-ball.          0.8::large;0.1::medium;0.1::small.
0.3::large; 0.6::medium; 0.1::small:- green.
```

### Experimental Setup and Results

Experiments were run on a 2.4 GHz Intel i5 processor. Learning terminates with $\epsilon = 1e{-}6$. All interpretations are sampled from the above programs. Partial interpretations are generated by randomly discarding literals in the interpretation, given a missing rate $m \in [0, 1]$. When $m = 0$, interpretations are fully observable. We obtain average measurements by executing all tasks using 5 random seeds. EMPLiFI and LFI-ProbLog programs are compiled as SDDs. Tables 1, 2 and 3 list parameter errors and EM iteration counts. Table 4 lists compilation, evaluation, total times and circuit sizes, which refer to node counts.

**Q1 How Much Does EMPLiFI Speed up EM Learning?** We run all three learners on Smokers and 100 fully observable interpretations. Table 1a shows that EMPLiFI and LFI-ProbLog converge to the same values, but EMPLiFI takes fewer EM iterations. This is consistent with Propositions 1 and 2. EMBLEM is not accurate in Table 1a. Since EMBLEM is not designed to learn all parameters at the same time [1], we split this task into four sub-tasks where each task learns one parameter and all other parameters are set to ground truth values. Results are in Table 1b, where EMBLEM is still the least accurate.

**Table 1.** Smokers. EMPLiFI is the most accurate and takes the fewest EM cycles.

| param name | empl err | lfip err | embl err |
|---|---|---|---|
| smo[.2] | **-.015** | **-.015** | -.200 |
| smo[.3] | **.046** | **.046** | .092 |
| can[.3] | **-.025** | **-.025** | -.142 |
| can[.1] | -.055 | -.055 | **-.049** |
| #iters | **19.6** | 51.2 | 108.0 |

(1a) Learning all parameters

| param name | empl | | lfip | | embl | |
|---|---|---|---|---|---|---|
| | err | iters | err | iters | err | iters |
| smo[.2] | **-.007** | **12.4** | **-.007** | **12.4** | -.200 | 171.0 |
| smo[.3] | **.039** | **20.0** | **.039** | 56.4 | -.083 | 38.0 |
| can[.3] | **-.025** | **3.0** | **-.025** | 41.6 | -.178 | 21.0 |
| can[.1] | **-.055** | **3.0** | **-.055** | 14.6 | -.100 | 61.0 |

(1b) Learning one parameter

**Q2 How Well Does EMPLiFI Handle Multi-head ADs?** This experiment shows that EMPLiFI is more accurate than EMBLEM and LFI-ProbLog in learning multi-head ADs. We run all three learners on Dice (Table 2) and Colors (Table 3) with 1k sampled interpretations under two settings. The first setting

**Table 2.** Dice. LFI-ProbLog fails when only positive literals are given.

| Param name | Fully observable | | | Only positive | | |
|---|---|---|---|---|---|---|
| | empl | lfip | embl | empl | lfip | embl |
| on[.15] | −.012 | −.012 | −.012 | −.012 | .85 | −.012 |
| tw[.15] | −.017 | −.017 | −.017 | −.017 | .85 | −.017 |
| th[.15] | .010 | .010 | .010 | .010 | .85 | .010 |
| fo[.15] | .000 | .000 | -.000 | .000 | .85 | .000 |
| fi[.15] | −.003 | −.003 | −.003 | −.003 | .85 | −.003 |
| si[.25] | .022 | .022 | .021 | .022 | .75 | .022 |
| #iters | 3.0 | 3.0 | 6.99k | 2.0 | 2.0 | 58.0 |

**Table 3.** Colors. EMPLiFI is the most accurate.

| Param name | Fully observable | | | Only positive | | |
|---|---|---|---|---|---|---|
| | empl | lfip | embl | empl | lfip | embl |
| gr[.8] | **−.013** | −.013 | −.048 | **.072** | **.072** | .200 |
| la[.3] | **−.012** | −.078 | −.064 | **−.127** | −.148 | −.300 |
| me[.6] | **.015** | .017 | −.026 | **.100** | .170 | .146 |
| sm[.1] | **−.003** | −.005 | .088 | **.028** | .210 | .154 |
| la[.8] | −.008 | **.002** | −.009 | **.046** | .077 | .200 |
| me[.1] | **−.008** | −.015 | −.100 | **−.045** | −.047 | −.100 |
| sm[.1] | .015 | **.012** | −.099 | **−.001** | .146 | −.100 |
| #iters | 25.4 | **14.4** | 25.6k | 25.0 | **14.4** | 171.0 |

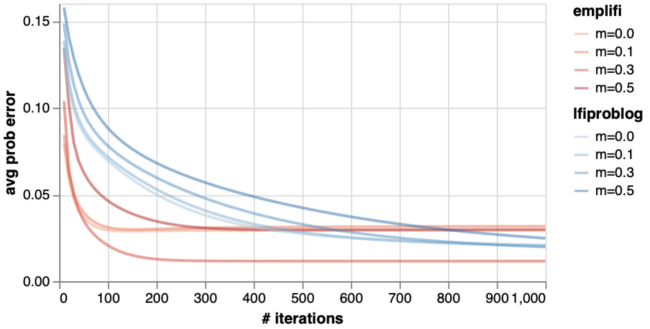


**Fig. 1.** The average error of EPS parameters decreases over EM iterations under all settings. EMPLiFI generally takes fewer iterations to converge compared to LFI-ProbLog.

learns from full interpretations (e.g. {one,¬two, ⋯, ¬six}), and the second setting learns from only positive literals (e.g. {one}). The second setting is fully observable as the truth values of all missing literals can be derived. Table 2 shows that when given all literals, all learners can learn multi-head ADs. However, when given only positive literals, LFI-ProbLog cannot learn. Table 3 shows an example that EMBLEM also fails at learning multi-head ADs.

**Q3 How Well Does EMPLiFI Handle Missing Data?** This experiment shows that EMPLiFI reduces EM iterations in the partially observable setting. We run EMPLiFI and LFI-ProbLog on EPS and 10k interpretations for a series of missingness values in $[0, 1]$. EMBLEM cannot handle this task with the limit of 3.5 GB memory. Figure 1 shows that EMPLiFI converges much sooner than LFI-ProbLog given either full or partial interpretations.

**Q4 Does EMPLiFI Require More Computational Resources?** We run EMPLiFI and LFI-ProbLog on EPS with 10k interpretations for a series of missingness values. Table 4 shows that EMPLiFI has larger circuits thus longer

compilation and evaluation time. This is because EMPLiFI includes information of mutually exclusive AD heads. However, EMPLiFI achieves shorter total execution time compared to LFI-ProbLog as it reduces EM iterations.

**Table 4.** EPS. The total runtime is the sum of compilation and evaluation time.

| missing rate | emplifi | | | | | | lfiproblog | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Size | Comp | Eval | Iters | Eval/Iter | Total | Size | Comp | Eval | Iters | Eval/Iter | Total |
| 0.0 | 75.0 | 16.6 | **91.1** | **199.4** | 5.0 | **107.6** | 58.0 | **6.1** | 106.3 | 406.6 | **0.3** | 112.4 |
| 0.1 | 75.9 | 233.6 | **986.2** | **114.0** | 8.7 | **1219.8** | 57.1 | **92.6** | 6865.40 | 1447.6 | **4.7** | 6957.9 |
| 0.3 | 74.9 | 455.1 | **5338.4** | **245.6** | 21.7 | **5793.4** | 53.7 | **165.3** | 15320.8 | 1765.2 | **8.7** | 15486.2 |
| 0.5 | 67.6 | 438.3 | **9133.7** | **418.6** | 21.8 | **9571.9** | 47.0 | **140.8** | 15651.8 | 1920.8 | **8.2** | 15792.6 |
| 0.7 | 50.5 | 265.54 | **6399.1** | **551.8** | 11.6 | **6664.6** | 35.3 | **94.3** | 12029.1 | 2229.8 | **5.4** | 12123.4 |
| 0.9 | 24.5 | 45.4 | **473.4** | **243.6** | 1.9 | **518.8** | 18.2 | **20.1** | 502.7 | 579.4 | **0.9** | 522.8 |

## 7 Related Work

We review EM-based parameter learners. PRISM [10] is one of the first EM learning algorithms, however, it imposes strong restrictions on the allowed programs. LFI-ProbLog [7,9] performs parameter learning of probabilistic logic programs. Before learning AD parameters, it must transform the program, which introduces latent variables that slow down learning. In extreme cases, it can converge to incorrect values. Asteroidea [6,18] tackles this issue by avoiding EM iterations for probabilistic rules, which is a specialization of EMPLiFI that supports single-head ADs, but not multi-head ADs. EMBLEM [1] is another EM-based parameter learner. It can naturally express and learn AD parameters as it is based on the language of Logic Programs with Annotated Disjunctions [17]. Similar to the aforementioned work, EMBLEM uses knowledge compilation techniques. However, it differs in the construction of BDDs as it focuses on learning a single target predicate. When multiple target predicates are present, EMBLEM can converge to incorrect values.

## 8 Conclusion

We have introduced EMPLiFI, an EM-based algorithm for probabilistic logic programs. EMPLiFI supports multi-head ADs and improves efficiency by learning from only relevant interpretations. Theoretically, we have proven that EMPLiFI is correct. Empirically, we have shown that EMPLiFI, compared to LFI-ProbLog and EMBLEM, is more accurate in learning multi-head ADs, and takes fewer iterations to converge. EMPLiFI is available in the ProbLog[2] repository.

[2] https://github.com/ML-KULeuven/problog.

**Acknowledgments.** This work was supported by the FNRS-FWO joint programme under EOS No. 30992574. It has also received funding from the Flemish Government under the "Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen" programme, the EU H2020 ICT48 project "TAILOR" under contract #952215, the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation, and the KU Leuven Research fund.

## References

1. Bellodi, E., Riguzzi, F.: Expectation maximization over binary decision diagrams for probabilistic logic programs. Intell. Data Anal. **17**(2), 343–363 (2013). https://doi.org/10.3233/IDA-130582
2. De Raedt, L., et al.: Towards digesting the alphabet-soup of statistical relational learning. In: Roy, D., Winn, J., McAllester, D., Mansinghka, V., Tenenbaum, J. (eds.) Proceedings of the 1st Workshop on Probabilistic Programming: Universal Languages, Systems and Applications, Whistler, Canada, December 2008
3. De Raedt, L.: Logical and relational learning. In: Zaverucha, G., da Costa, A.L. (eds.) Advances in Artificial Intelligence - SBIA 2008, p. 1. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-68856-3
4. De Raedt, L., Kersting, K.: Probabilistic inductive logic programming. In: De Raedt, L., Frasconi, P., Kersting, K., Muggleton, S. (eds.) Probabilistic Inductive Logic Programming. LNCS (LNAI), vol. 4911, pp. 1–27. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78652-8_1
5. De Raedt, L., Kimmig, A.: Probabilistic (logic) programming concepts. Mach. Learn. **100**(1), 5–47 (2015). https://doi.org/10.1007/s10994-015-5494-z
6. Faria, F.H.O.V.D., Gusmão, A., Cozman, F.G., Mauá, D.: Speeding-up problog's parameter learning. arXiv:1707.08151 (2017)
7. Fierens, D., et al.: Inference and learning in probabilistic logic programs using weighted Boolean formulas. Theory Pract. Logic Program. **15**(3), 358–401 (2015). https://doi.org/10.1017/S1471068414000076
8. Getoor, L., Taskar, B.: Introduction to Statistical Relational Learning (Adaptive Computation and Machine Learning). MIT Press, Cambridge (2007)
9. Gutmann, B., Thon, I., De Raedt, L.: Learning the parameters of probabilistic logic programs from interpretations. In: Gunopulos, D., Hofmann, T., Malerba, D., Vazirgiannis, M. (eds.) ECML PKDD 2011. LNCS (LNAI), vol. 6911, pp. 581–596. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23780-5_47
10. Kameya, Y., Sato, T.: Parameter learning of logic programs for symbolic-statistical modeling. arXiv:1106.1797 (2011)
11. Kersting, K., Raedt, L.D.: Bayesian Logic Programming: Theory and Tool. MIT Press, Cambridge (2007)
12. Neapolitan, R.E.: Learning Bayesian Networks. Prentice-Hall Inc., Hoboken (2003)
13. Poole, D.: The independent choice logic for modelling multiple agents under uncertainty. Artif. Intell. **94**(1), 7–56 (1997). https://doi.org/10.1016/S0004-3702(97)00027-1, Economic Principles of Multi-Agent Systems
14. Richardson, M., Domingos, P.: Markov logic networks. Mach. Learn. **62**(1–2), 107–136 (2006). https://doi.org/10.1007/s10994-006-5833-1
15. Sato, T.: A statistical learning method for logic programs with distribution semantics. In: Proceedings of the 12th International Conference on Logic Programming, ICLP 1995, pp. 715–729. MIT Press (1995)

16. Vennekens, J., Denecker, M., Bruynooghe, M.: CP-logic: a language of causal probabilistic events and its relation to logic programming. Theory Pract. Logic Program. **9**(3), 245–308 (2009). https://doi.org/10.1017/S1471068409003767
17. Vennekens, J., Verbaeten, S., Bruynooghe, M.: Logic programs with annotated disjunctions. In: Demoen, B., Lifschitz, V. (eds.) ICLP 2004. LNCS, vol. 3132, pp. 431–445. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-27775-0_30
18. Vieira de Faria, F.H.O., Gusmão, A.C., De Bona, G., Mauá, D.D., Cozman, F.G.: Speeding up parameter and rule learning for acyclic probabilistic logic programs. Int. J. Approx. Reason. **106**, 32–50 (2019). https://doi.org/10.1016/j.ijar.2018.12.012. ISSN 0888613X. Elsevier Inc.