EM over Binary Decision Diagrams for Probabilistic Logic Programs

Elena Bellodi and Fabrizio Riguzzi

ENDIF – Università di Ferrara – Via Saragat, 1 – 44122 Ferrara, Italy.
{elena.bellodi,fabrizio.riguzzi}@unife.it

Abstract. Recently much work in Machine Learning has concentrated on representation languages able to combine aspects of logic and probability, leading to the birth of a whole field called Statistical Relational Learning. In this paper we present a technique for parameter learning targeted to a family of formalisms where uncertainty is represented using Logic Programming techniques - the so-called Probabilistic Logic Programs such as ICL, PRISM, ProbLog and LPAD. Since their equivalent Bayesian networks contain hidden variables, an EM algorithm is adopted. In order to speed the computation, expectations are computed directly on the Binary Decision Diagrams that are built for inference. The resulting system, called EMBLEM for "EM over Bdds for probabilistic Logic programs Efficient Mining", has been applied to a number of datasets and showed good performances both in terms of speed and memory usage.

Keywords: Statistical Relational Learning, Probabilistic Logic Programming, Distribution Semantics, Logic Programs with Annotated Disjunctions, Expectation Maximization

1 Introduction

Machine Learning has seen the development of the field of Statistical Relational Learning (SRL) where logical-statistical languages are used in order to effectively learn in complex domains involving relations and uncertainty. They have been successfully applied in social networks analysis, entity recognition, collective classification and information extraction, to name a few.

Similarly, a large number of works in Logic Programming have attempted to combine logic and probability, among which the *distribution semantics* [21] is a prominent approach. This semantics underlies for example PRISM [21], the Independent Choice Logic [14], Logic Programs with Annotated Disjunctions (LPADs) [29], ProbLog [4] and CP-logic [27]. The approach is particularly appealing because efficient inference algorithms appeared [4,17], which adopt Binary Decision Diagrams (BDDs).

In this paper we present the EMBLEM system for "EM over Bdds for probabilistic Logic programs Efficient Mining" [1] that learns parameters of probabilistic logic programs under the distribution semantics by using an Expectation

Maximization (EM) algorithm. Such an algorithm is a popular tool in statistical estimation problems involving incomplete data: it is an iterative method to estimate some unknown parameters Θ of a model, given a dataset where some of the data is missing. The aim is to find maximum likelihood or maximum a posteriori (MAP) estimates of Θ [13]. EM alternates between performing an expectation (E) step, where the missing data are estimated given the observed data and current estimate of the model parameters, and a maximization (M) step, which computes the parameters maximizing the likelihood of the data given the sufficient statistics on the data computed in the E step. The translation of the probabilistic programs into graphical models requires the use of hidden variables (see Section 3) and therefore of EM: the main characteristic of our system is the computation of the values of expectations using BDDs.

Since there are transformations with linear complexity that can convert a program in a language under the distribution semantics into the others [28], we will use LPADs for their general syntax. EMBLEM has been tested on the IMDB, Cora and UW-CSE datasets and compared with RIB [20], LeProbLog [4], Alchemy [15] and CEM, an implementation of EM based on [17].

The paper is organized as follows. Section 2 presents LPADs and Section 3 describes EMBLEM. Section 4 discusses related works. Section 5 shows the results of the experiments performed and Section 6 concludes the paper.

2 Logic Programs with Annotated Disjunctions

Formally a *Logic Program with Annotated Disjunctions* [29] consists of a finite set of annotated disjunctive clauses. An annotated disjunctive clause C_i is of the form $h_{i1} : \Pi_{i1}; \ldots; h_{in_i} : \Pi_{in_i} : -b_{i1}, \ldots, b_{im_i}$. In such a clause $h_{i1}, \ldots h_{in_i}$ are logical atoms and $b_{i1}, \ldots, b_{im_i}$ are logical literals, $\Pi_{i1}, \ldots, \Pi_{in_i}$ are real numbers in the interval $[0, 1]$ such that $\sum_{k=1}^{n_i} \Pi_{ik} \leq 1$. $b_{i1}, \ldots, b_{im_i}$ is called the *body* and is indicated with $body(C_i)$. Note that if $n_i = 1$ and $\Pi_{i1} = 1$ the clause corresponds to a non-disjunctive clause. If $\sum_{k=1}^{n_i} \Pi_{ik} < 1$, the head of the annotated disjunctive clause implicitly contains an extra atom *null* that does not appear in the body of any clause and whose annotation is $1 - \sum_{k=1}^{n_i} \Pi_{ik}$. We denote by $ground(T)$ the grounding of an LPAD T.

An *atomic choice* is a triple (C_i, θ_j, k) where $C_i \in T$, θ_j is a substitution that grounds C_i and $k \in \{1, \ldots, n_i\}$. (C_i, θ_j, k) means that, for the ground clause $C_i\theta_j$, the head h_{ik} was chosen. In practice $C_i\theta_j$ corresponds to a random variable X_{ij} and an atomic choice (C_i, θ_j, k) to an assignment $X_{ij} = k$. A set of atomic choices κ is *consistent* if $(C, \theta, i) \in \kappa, (C, \theta, j) \in \kappa \Rightarrow i = j$, i.e., only one head is selected for a ground clause. A *composite choice* κ is a consistent set of atomic choices. The *probability* $P(\kappa)$ *of a composite choice* κ is the product of the probabilities of the individual atomic choices, i.e. $P(\kappa) = \prod_{(C_i,\theta_j,k)\in\kappa} \Pi_{ik}$.

A *selection* σ is a composite choice that, for each clause $C_i\theta_j$ in $ground(T)$, contains an atomic choice (C_i, θ_j, k). We denote the set of all selections σ of a program T by $\mathcal{S}_T$. A selection σ identifies a normal logic program w_σ defined as $w_\sigma = \{(h_{ik} \leftarrow body(C_i))\theta_j | (C_i, \theta_j, k) \in \sigma\}$. w_σ is called a *world* of T. Since

selections are composite choices we can assign a probability to possible worlds: $P(w_\sigma) = P(\sigma) = \prod_{(C_i,\theta_j,k)\in\sigma} \Pi_{ik}$.

We consider only *sound* LPADs, in which every possible world has a total well-founded model. In the following we write $w_\sigma \models Q$ to mean that the query Q is true in the well-founded model of the program w_σ.

The probability of a query Q according to an LPAD T is given by

$$P(Q) = \sum_{\sigma\in E(Q)} P(\sigma) \tag{1}$$

where we define $E(Q)$ as $\{\sigma \in \mathcal{S}_T, w_\sigma \models Q\}$, the set of selections corresponding to worlds where the query is true.

To reduce the computational cost of answering queries in our experiments, random variables can be directly associated to clauses rather than to their ground instantiations: atomic choices then take the form (C_i, k), meaning that head h_{ik} is selected from program clause C_i, i.e., that $X_i = k$.

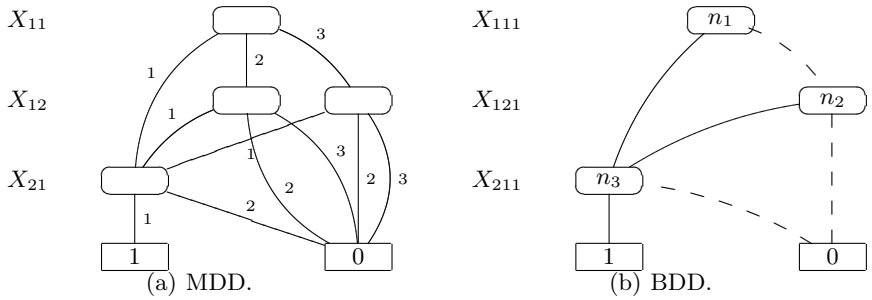
Example 1. The following LPAD T encodes a very simple model of the development of an epidemic or pandemic:

$C_1 = epidemic : 0.6; pandemic : 0.3 : -flu(X), cold.$
$C_2 = cold : 0.7.$
$C_3 = flu(david).$
$C_4 = flu(robert).$

Clause C_1 has two groundings, $C_1\theta_1$ with $\theta_1 = \{X/david\}$ and $C_1\theta_2$ with $\theta_2 = \{X/robert\}$, so there are two random variables X_{11} and X_{12}; C_2 has only one grounding that is associated to the variable X_{21}. X_{11} and X_{12} have three values since C_1 has three head atoms (*epidemic*, *pandemic*, *null*); similarly X_{21} has two values since C_2 has two head atoms (*cold*, *null*).

The worlds in which a query is true can be represented using a Multivalued Decision Diagram (MDD) [25]. An MDD represents a function $f(\mathbf{X})$ taking Boolean values on a set of multivalued variables $\mathbf{X}$ by means of a rooted graph that has one level for each variable. Each node is associated to the variable of its level and has one child for each possible value of the variable. The leaves store either 0 or 1. Given values for all the variables $\mathbf{X}$, we can compute the value of $f(\mathbf{X})$ by traversing the graph starting from the root and returning the value associated to the leaf that is reached. A MDD can be used to represent the set $E(Q)$ by considering the multivalued variables X_{ij}s associated to the $C_i\theta_j$s of $ground(T)$. X_{ij} has values $\{1, \ldots, n_i\}$ and the atomic choice (C_i, θ_j, k) corresponds to the propositional equation $X_{ij} = k$. If we represent with an MDD the function $f(\mathbf{X}) = \bigvee_{\sigma\in E(Q)} \bigwedge_{(C_i,\theta_j,k)\in\sigma} X_{ij} = k$, then the MDD will have a path to a 1-leaf for each world where Q is true. While building MDDs, simplification operations can be applied that delete or merge nodes. Merging is performed when the diagram contains two identical sub-diagrams, while deletion is performed when all arcs from a node point to the same node. In this way a reduced MDD is obtained with respect to a Multivalued Decision Tree (MDT), i.e., a MDD in which every node has a single parent, all the children belong to the

level immediately below and all the variables have at least one node. For example, the reduced MDD corresponding to the query *epidemic* from Example 1 is shown in Figure 1(a). The labels on the edges represent the values of the variable associated to the node.



(a) MDD. (b) BDD.

Fig. 1. Decision diagrams for Example 1.

It is often unfeasible to find all the worlds where the query is true so inference algorithms find instead *explanations* for it, i.e. composite choices such that the query is true in all the worlds whose selections are a superset of them. Explanations however, differently from possible worlds, are not necessarily mutually exclusive with respect to each other, but exploiting the fact that MDDs split paths on the basis of the values of a variable and the branches are mutually disjoint so a dynamic programming algorithm can be applied for computing the probability.

Most packages for the manipulation of decision diagrams are however restricted to work on Binary Decision Diagrams, i.e., decision diagrams where all the variables are Boolean. A node n in a BDD has two children: the 1-child, indicated with $child_1(n)$, and the 0-child, indicated with $child_0(n)$. The 0-branch, the one going to the 0-child, is drawn with a dashed line.

To work on MDDs with a BDD package we must represent multivalued variables by means of binary variables. For a multivalued variable X_{ij}, corresponding to ground clause $C_i\theta_j$, having n_i values, we use $n_i - 1$ Boolean variables $X_{ij1}, \ldots, X_{ijn_i-1}$ and we represent the equation $X_{ij} = k$ for $k = 1, \ldots n_i - 1$ by means of the conjunction $\overline{X_{ij1}} \wedge \overline{X_{ij2}} \wedge \ldots \wedge \overline{X_{ijk-1}} \wedge X_{ijk}$, and the equation $X_{ij} = n_i$ by means of the conjunction $\overline{X_{ij1}} \wedge \overline{X_{ij2}} \wedge \ldots \wedge \overline{X_{ijn_i-1}}$. Figure 1(b) shows the reduced BDD corresponding to the MDD in Figure 1(a). BDDs can be used for computing the probability of queries by associating to each Boolean variable X_{ijk} a parameter π_{ik} that represents $P(X_{ijk} = 1)$. If we define $g(i) = \{j|\theta_j$ is a substitution grounding $C_i\}$ then $P(X_{ijk} = 1) = \pi_{ik}$ for all $j \in g(i)$. The parameters are obtained from those of multivalued variables in this way:

$$\pi_{i1} = \Pi_{i1}$$

$$\ldots$$
$$\pi_{ik} = \frac{\Pi_{ik}}{\prod_{j=1}^{k-1}(1-\pi_{ij})}$$
$$\ldots$$

up to $k = n_i - 1$.

3 EMBLEM

EMBLEM applies the algorithm for performing EM over BDDs, proposed in [26,9,10,8], to the problem of learning the parameters of an LPAD. EMBLEM takes as input a number of goals that represent the examples and for each one generates the BDD encoding its explanations. The examples are organized in a set of interpretations (sets of ground facts) each describing a portion of the domain of interest. The queries correspond to ground atoms in an interpretation whose predicate has been indicated as "target" by the user. The predicates can be treated as closed-world or open-world. In the first case the body of clauses is resolved only with facts in the interpretation, in the second case it is resolved both with facts in the interpretation and with clauses in the theory. If the last option is set and the theory is cyclic, we use a depth bound on SLD-derivations to avoid going into infinite loops, as proposed by [6]. Given the program containing only the clauses C_1 and C_2 from Example 1 and the interpretation $\{epidemic, flu(david), flu(robert)\}$, we obtain the BDD in Figure 1(b) that represents the query *epidemic*. A value of 1 for the Boolean variables X_{111} and X_{121} means that, for the ground clauses $C_1\theta_1$ and $C_1\theta_2$, the head $h_{11} =$ *epidemic* is chosen, regardless of the other variables for the clause (X_{112}, X_{122}) that are in fact omitted from the diagram.

Then EMBLEM enters the EM cycle, in which the steps of expectation and maximization are repeated until the log-likelihood of the examples reaches a local maximum. The necessity of exploiting EM depends on the fact that, to determine the parameters Π_{ik}, the number of times that a head h_{ik} has been chosen is required. The information about which selection was used in the derivation of a goal is unknown, so the random variables are hidden and we compute expected counts. For a single example Q:

- Expectation: computes $\mathbf{E}[c_{ik0}|Q]$ and $\mathbf{E}[c_{ik1}|Q]$ for all rules C_i and $k = 1, \ldots, n_i - 1$, where c_{ikx} is the number of times a variable X_{ijk} takes value x for $x \in \{0, 1\}$, with j in $g(i)$. $\mathbf{E}[c_{ikx}|Q]$ is given by $\sum_{j \in g(i)} P(X_{ijk} = x|Q)$.
- Maximization: computes π_{ik} for all rules C_i and $k = 1, \ldots, n_i - 1$.

$$\pi_{ik} = \frac{\mathbf{E}[c_{ik1}|Q]}{\mathbf{E}[c_{ik0}|Q] + \mathbf{E}[c_{ik1}|Q]} \tag{2}$$

If we have more than one example the contributions of each example simply sum up when computing $\mathbf{E}[c_{ijx}]$.

$P(X_{ijk} = x|Q)$ is given by $P(X_{ijk} = x|Q) = \frac{P(X_{ijk}=x,Q)}{P(Q)}$ with

$$\begin{aligned} P(X_{ijk} = x, Q) &= \sum_{\sigma \in \mathcal{S}_T} P(Q, X_{ijk} = x, \sigma) \\ &= \sum_{\sigma \in \mathcal{S}_T} P(Q|\sigma) P(X_{ijk} = x|\sigma) P(\sigma) \\ &= \sum_{\sigma \in E(Q)} P(X_{ijk} = x|\sigma) P(\sigma) \end{aligned}$$

where $P(X_{ijk} = 1|\sigma) = 1$ if $(C_i, \theta_j, k) \in \sigma$ for $k = 1, \ldots, n_i - 1$ and 0 otherwise.

Since there is a one to one correspondence between the possible worlds where Q is true and the paths to a 1 leaf in a Binary Decision Tree (a MDT with binary variables),

$$P(X_{ijk} = x, Q) = \sum_{\rho \in R(Q)} P(X_{ijk} = x|\rho) \prod_{d \in \rho} \pi(d)$$

where ρ is a path, and if σ corresponds to ρ, then $P(X_{ijk} = x|\sigma)$=$P(X_{ijk} = x|\rho)$. $R(Q)$ is the set of paths in the BDD for query Q that lead to a 1 leaf, d is an edge of ρ and $\pi(d)$ is the probability associated to the edge: if d is the 1-branch from a node associated to a variable X_{ijk}, then $\pi(d) = \pi_{ik}$, if d is the 0-branch from a node associated to a variable X_{ijk}, then $\pi(d) = 1 - \pi_{ik}$.

Now consider a BDT in which only the merge rule is applied, fusing together identical sub-diagrams. The resulting diagram, that we call Complete Binary Decision Diagram (CBDD), is such that every path contains a node for every level. For a CBDD, $P(X_{ijk} = x, Q)$ can be further expanded as

$$P(X_{ijk} = x, Q) = \sum_{\rho \in R(Q), (X_{ijk}=x) \in \rho} \prod_{d \in \rho} \pi(d)$$

where $(X_{ijk} = x) \in \rho$ means that ρ contains an x-edge from a node associated to X_{ijk}. We can then write

$$P(X_{ijk} = x, Q) = \sum_{n \in N(Q), v(n)=X_{ijk}, \rho_n \in R_n(Q), \rho^n \in R^n(Q,x)} \prod_{d \in \rho_n} \pi(d) \prod_{d \in \rho^n} \pi(d)$$

where $N(Q)$ is the set of nodes of the CBDD, $v(n)$ is the variable associated to node n, $R_n(Q)$ is the set containing the paths from the root to n and $R^n(Q, x)$ is the set of paths from n to the 1 leaf through its x-child.

$$\begin{aligned} P(X_{ijk} = x, Q) &= \sum_{n \in N(Q), v(n)=X_{ijk}} \sum_{\rho_n \in R_n(Q)} \sum_{\rho^n \in R^n(Q,x)} \prod_{d \in \rho_n} \pi(d) \prod_{d \in \rho^n} \pi(d) \\ &= \sum_{n \in N(Q), v(n)=X_{ijk}} \sum_{\rho_n \in R_n(Q)} \prod_{d \in \rho_n} \pi(d) \sum_{\rho^n \in R^n(Q,x)} \prod_{d \in \rho^n} \pi(d) \\ &= \sum_{n \in N(Q), v(n)=X_{ijk}} F(n) B(child_x(n)) \pi_{ikx} \end{aligned}$$

where π_{ikx} is π_{ik} if x=1 and $(1-\pi_{ik})$ if x=0. $F(n)$ is the *forward probability* [10], the probability mass of the paths from the root to n, while $B(n)$ is the *backward probability* [10], the probability mass of paths from n to the 1 leaf. If *root* is the root of a tree for a query Q then $B(root) = P(Q)$.

The expression $F(n)B(child_x(n))\pi_{ikx}$ represents the sum of the probabilities of all the paths passing through the x-edge of node n and is indicated with $e^x(n)$. Thus

$$P(X_{ijk} = x, Q) = \sum_{n \in N(Q), v(n)=X_{ijk}} e^x(n) \tag{3}$$

For the case of a BDD, i.e., a diagram obtained by applying also the deletion rule, Formula 3 is no longer valid since also paths where there is no node associated to X_{ijk} can contribute to $P(X_{ijk} = x, Q)$. These paths might have been obtained from a BDD having a node m associated to variable X_{ijk} that is a descendant of node n along the 0-branch and whose outgoing edges both point to $child_0(n)$. The correction of formula (3) to take into account this aspect is applied in the Expectation step.

EMBLEM's main procedure consists of a cycle in which the procedures Expectation and Maximization are repeatedly called. Procedure Expectation returns the log likelihood of the data that is used in the stopping criterion: EMBLEM stops when the difference between the log likelihood of the current iteration and the one of the previous iteration drops below a threshold ϵ or when this difference is below a fraction δ of the current log likelihood.

Procedure Expectation takes as input a list of BDDs, one for each example, and computes the expectations for each one, i.e. $P(X_{ijk} = x, Q)$ for all variables X_{ijk} in the BDD and values $x \in \{0, 1\}$. In the procedure we use $\eta^x(i, k)$ to indicate $\sum_{j \in g(i)} P(X_{ijk} = x, Q)$. Expectation first calls GetForward and GetBackward that compute the forward, the backward probability of nodes and $\eta^x(i, k)$ for non-deleted paths only. Then it updates $\eta^x(i, k)$ to take into account deleted paths. The expectations are updated in this way: for all rules i and for $k = 1$ to $n_i - 1$, $\mathbf{E}[c_{ikx}] = \mathbf{E}[c_{ikx}] + \eta^x(i, k)/P(Q)$.

Procedure Maximization computes the parameters values for the next EM iteration, as specified in (2).

Procedure GetForward traverses the diagram one level at a time starting from the root level, where F(root)=1, and for each node n it computes its contribution to the forward probabilities of its children. Then the forward probabilities of both children are updated in this way: $F(child_x(node)) = F(child_x(node)) + F(node) \cdot \pi_{ikx}$.

Function GetBackward computes the backward probability of nodes by traversing recursively the tree from the leaves to the root. When the calls of GetBackward for both children of a node n return, we have all the information that is needed to compute the e^x values and the value of $\eta^x(i, k)$ for non-deleted paths. An array ς is used here to store the contributions of the deleted paths by starting from the root level and accumulating $\varsigma(l)$ for the various levels l.

A fully detailed description of EMBLEM together with an example of its execution can be found in [1].

4 Related Works

Our work has close connection with various other works. [9,10] proposed an EM algorithm for learning the parameters of Boolean random variables given observations of the values of a Boolean function over them, represented by a BDD. EMBLEM is an application of that algorithm to probabilistic logic programs. Independently, also [26] proposed an EM algorithm over BDD to learn parameters for the CPT-L language.[7] presented the CoPrEM algorithm that performs EM over BDDs for the ProbLog language.

Approaches for learning probabilistic logic programs can be classified into three categories: those that employ constraint techniques (such as [16,18]), those that use EM and those that adopt gradient descent.

Among the approaches that use EM, [12] first proposed to use it to induce parameters and the Structural EM algorithm to induce ground LPADs structures. Their EM algorithm however works on the underlying Bayesian network. RIB [20] performs parameter learning using the information bottleneck approach, which is an extension of EM targeted especially towards hidden variables. The PRISM system [21,22] is one of the first learning algorithms based on EM.

Among the works that use a gradient descent technique, LeProbLog [5,6] finds the parameters of a ProbLog program that minimize the Mean Squared Error of the probability of queries and uses BDD to compute the gradient.

Alchemy [15] is a state of the art SRL system that offers various tools for inference, weight learning and structure learning of Markov Logic Networks (MLNs). MLNs differ significantly from the languages under the distribution semantics since they extend first-order logic by attaching weights to logical formulas, reflecting "how strong" they are, but do not allow to exploit logic programming techniques.

5 Experiments

EMBLEM has been tested over three real world datasets: IMDB[1], UW-CSE[2] [23] and Cora[3] [23]. We implemented EMBLEM in Yap Prolog[4] and we compared it with RIB [20]; CEM, an implementation of EM based on the `cplint` inference library [17,19]; LeProblog [5,6] and Alchemy [15]. All experiments were performed on Linux machines with an Intel Core 2 Duo E6550 (2333 MHz) processor and 4 GB of RAM.

To compare our results with LeProbLog we exploited the translation of LPADs into ProbLog proposed in [3], for Alchemy we exploited the translation between LPADs and MLNs used in [20].

For the probabilistic logic programming systems (EMBLEM, RIB, CEM and LeProbLog) we consider various options. The first consists in choosing between

[1] `http://alchemy.cs.washington.edu/data/imdb`
[2] `http://alchemy.cs.washington.edu/data/uw-cse`
[3] `http://alchemy.cs.washington.edu/data/cora`
[4] `http://www.dcc.fc.up.pt/~vsc/Yap`

associating a distinct random variable to each grounding of a probabilistic clause or a single random variable to a non-ground probabilistic clause expressing whether the clause is used or not. The latter case makes the problem easier. The second option is concerned with imposing a limit on the depth of derivations as done in [6], thus eliminating explanations associated to derivations exceeding the depth limit. This is necessary for problems that contain cyclic clauses, such as transitive closure clauses. The third option involves setting the number of restarts for EM based algorithms. All experiments for probabilistic logic programming systems have been performed using open-world predicates.

IMDB regards movies, actors, directors and movie genres and it is divided into five mega-examples. We performed training on four mega-examples and testing on the remaining one. Then we drew a Precision-Recall curve and computed the Area Under the Curve (AUCPR) using the method reported in [2]. We defined 4 different LPADs, two for predicting the target predicate `sameperson/2`, and two for predicting `samemovie/2`. We had one positive example for each fact that is true in the data, while we sampled from the complete set of false facts three times the number of true instances in order to generate negative examples.

For predicting `sameperson/2` we used the same LPAD of [20]:

```
sameperson(X,Y):p:- movie(M,X),movie(M,Y).
sameperson(X,Y):p:- actor(X),actor(Y),workedunder(X,Z),
                    workedunder(Y,Z).
sameperson(X,Y):p:- gender(X,Z),gender(Y,Z).
sameperson(X,Y):p:- director(X),director(Y),genre(X,Z),genre(Y,Z).
```

where `p` is a placeholder meaning the parameter must be learned. We ran EMBLEM on it with the following settings: no depth bound, random variables associated to instantiations of clauses and a number of restarts chosen to match the execution time of EMBLEM with that of the fastest other algorithm.

The queries that LeProbLog takes as input are obtained by annotating with 1.0 each positive example for `sameperson/2` and with 0.0 each negative example for `sameperson/2`. We ran LeProbLog for a maximum of 100 iterations or until the difference in Mean Squared Error (MSE) between two iterations got smaller than 10^{-5}; this setting was used in all the following experiments as well. For Alchemy we always used the preconditioned rescaled conjugate gradient discriminative algorithm [11]. For this experiments we specified `sameperson/2` as the only non-evidence predicate.

A second LPAD has been created to evaluate the performance of the algorithms when some atoms are unseen:

```
sameperson_pos(X,Y):p:- movie(M,X),movie(M,Y).
sameperson_pos(X,Y):p:- actor(X),actor(Y),
                        workedunder(X,Z),workedunder(Y,Z).
sameperson_pos(X,Y):p:- director(X),director(Y),genre(X,Z),
                        genre(Y,Z).
sameperson_neg(X,Y):p:- movie(M,X),movie(M,Y).
sameperson_neg(X,Y):p:- actor(X),actor(Y),
```

```
                        workedunder(X,Z),workedunder(Y,Z).
sameperson_neg(X,Y):p:- director(X),director(Y),genre(X,Z),
                        genre(Y,Z).
sameperson(X,Y):p:- \+ sameperson_pos(X,Y), sameperson_neg(X,Y).
sameperson(X,Y):p:- \+ sameperson_pos(X,Y),\+ sameperson_neg(X,Y).
sameperson(X,Y):p:- sameperson_pos(X,Y), sameperson_neg(X,Y).
sameperson(X,Y):p:- sameperson_pos(X,Y), \+ sameperson_neg(X,Y).
```

The `sameperson_pos/2` and `sameperson_neg/2` predicates are unseen in the data. Settings are the same as the ones for the previous LPAD. In this experiment Alchemy was run with the `−withEM` option that turns on EM learning.

Table 1 shows the AUCPR averaged over the five folds for EMBLEM, RIB, LeProbLog, CEM and Alchemy. Results for the two LPADs are shown respectively in the IMDB-SP and IMDBu-SP rows. Table 2 shows the learning times in hours.

For predicting `samemovie/2` we used the LPAD:

```
samemovie(X,Y):p:- movie(X,M),movie(Y,M),actor(M).
samemovie(X,Y):p:- movie(X,M),movie(Y,M),director(M).
samemovie(X,Y):p:- movie(X,A),movie(Y,B),actor(A),director(B),
                   workedunder(A,B).
samemovie(X,Y):p:- movie(X,A),movie(Y,B),director(A),director(B),
                   genre(A,G),genre(B,G).
```

To test the behaviour when unseen predicates are present, we transformed the program for `samemovie/2` as we did for `sameperson/2`, thus introducing the unseen predicates `samemovie_pos/2` and `samemovie_neg/2`. We ran EMBLEM on them with no depth bound, one variable for each instantiation of the rules and one random restart. With regard to LeProbLog and Alchemy, we ran them with the same settings as IMDB-SP and IMDBu-SP, by replacing `sameperson` with `samemovie`.

Table 1 shows, in the IMDB-SM and IMDBu-SM rows, the average AUCPR for EMBLEM, LeProblog and Alchemy. For RIB and CEM we obtained a lack of memory error (indicated with "me"); Table 2 shows the learning times in hours.

The Cora database contains citations to computer science research papers. For each citation we know the title, authors, venue and the words that appear in them. The task is to determine which citations are referring to the same paper, by predicting the predicate `samebib(cit1,cit2)`.

From the MLN proposed in [24][5] we obtained two LPADs. The first contains 559 rules and differs from the direct translation of the MLN because rules involving words are instantiated with the different constants, only positive literals for the `hasword` predicates are used and transitive rules are not included:

```
samebib(B,C):p:- author(B,D),author(C,E),sameauthor(D,E).
samebib(B,C):p:- title(B,D),title(C,E),sametitle(D,E).
samebib(B,C):p:- venue(B,D),venue(C,E),samevenue(D,E).
```

[5] `http://alchemy.cs.washington.edu/mlns/er`

```
samevenue(B,C):p:-haswordvenue(B,word_06),haswordvenue(C,word_06).
...
sametitle(B,C):p:-haswordtitle(B,word_10),haswordtitle(C,word_10).
....
sameauthor(B,C):p:- haswordauthor(B,word_a),
                    haswordauthor(C,word_a).
.....
```

The dots stand for the rules for all the possible words. The Cora dataset comprises five mega-examples each containing facts for the four predicates `samebib/2`, `samevenue/2`, `sametitle/2` and `sameauthor/2`, which have been set as target predicates. We used as negative examples those contained in the Alchemy dataset. We ran EMBLEM on this LPAD with no depth bound, a single variable for each instantiation of the rules and a number of restarts chosen to match the execution time of EMBLEM with that of the fastest other algorithm.

The second LPAD adds to the previous one four transitive rules of the form

```
samebib(A,B):p :- samebib(A,C),samebib(C,B).
```

for every target predicate, for a total of 563 rules. In this case we had to run EMBLEM with a depth bound equal to two and a single variable for each non--ground rule; the number of restarts was one. As for LeProbLog, we separately learned the four predicates because learning the whole theory at once would give a lack of memory error. We annotated with 1.0 each positive example for `samebib/2`, `sameauthor/2`, `sametitle/2`, `samevenue/2` and with 0.0 the negative examples for the same predicates. As for Alchemy we learned weights with the four predicates as the non-evidence predicates. Table 1 shows in the Cora and CoraT (Cora transitive) rows the average AUCPR obtained by training on four mega-examples and testing on the remaining one. CEM and Alchemy on CoraT gave a lack of memory error while RIB was not applicable because it was not possible to split the input examples into smaller independent interpretations as required by RIB.

The UW-CSE dataset contains information about the Computer Science department of the University of Washington through 22 different predicates, such as `yearsInProgram/2`, `advisedBy/2`, `taughtBy/3` and is split into five mega--examples. The goal here is to predict the `advisedBy/2` predicate, namely the fact that a person is advised by another person: this was our target predicate. The negative examples have been generated by considering all couple of persons `(a,b)` where `a` and `b` appear in an `advisedby/2` fact in the data and by adding a negative example `advisedby(a,b)` if it is not in the data.

The theory used was obtained from the MLN of [23][6]. It contains 86 rules, such as for instance:

```
advisedby(S, P) :p :- courselevel(C,level_500),taughtby(C,P,Q),
                      ta(C, S, Q).
```

[6] `http://alchemy.cs.washington.edu/mlns/uw-cse`

We ran EMBLEM on it with a single variable for each instantiation of a rule, a depth bound of two and one random restart.

The annotated queries that LeProbLog takes as input have been created by annotating with 1.0 each positive example for `advisedby/2` and with 0.0 the negative examples. As for Alchemy, we learned weights with `advisedby/2` as the only non-evidence predicate. Table 1 shows the AUCPR averaged over the five mega-examples for all the algorithms.

Table 3 shows the p-value of a paired two-tailed t-test at the 5% significance level of the difference in AUCPR between EMBLEM and RIB/LeProbLog/CEM/ Alchemy (significant differences in bold).

Table 1. Results of the experiments on all datasets. IMDBu refers to the IMDB dataset with the theory containing unseen predicates. CoraT refers to the theory containing transitive rules. Numbers in parenthesis followed by r mean the number of random restarts (when different from one) to reach the area specified. "me" means memory error during learning. AUCPR is the area under the precision-recall curve averaged over the five folds. R is RIB, L is LeProbLog, C is CEM, A is Alchemy.

Dataset	AUCPR				
	EMBLEM	R	L	C	A
IMDB-SP	0.202(500r)	0.199	0.096	0.202	0.107
IMDBu-SP	0.175(40r)	0.166	0.134	0.120	0.020
IMDB-SM	1.000	me	0.933	0.537	0.820
IMDBu-SM	1.000	me	0.933	0.515	0.338
Cora	0.995(120r)	0.939	0.905	0.995	0.469
CoraT	0.991	no	0.970	me	me
UW-CSE	0.883	0.588	0.270	0.644	0.294

Table 2. Execution time in hours of the experiments on all datasets. R is RIB, L is LeProbLog, C is CEM and A is Alchemy.

Dataset	Time(h)				
	EMBLEM	R	L	C	A
IMDB-SP	0.01	0.016	0.35	0.01	1.54
IMDBu-SP	0.01	0.0098	0.23	0.012	1.54
IMDB-SM	0.00036	me	0.005	0.0051	0.0026
IMDBu-SM	3.22	me	0.0121	0.0467	0.0108
Cora	2.48	2.49	13.25	11.95	1.30
CoraT	0.38	no	4.61	me	me
UW-CSE	2.81	0.56	1.49	0.53	1.95

From the results we can observe that over IMDB EMBLEM has comparable performances with CEM for IMDB-SP, with similar execution time. On IMDBu-SP it has better performances than all other systems, with a learning time equal

Table 3. Results of t-test on all datasets. p is the p-value of a paired two-tailed t-test (significant differences at the 5% level in bold) between EMBLEM and all the others. R is RIB, L is LeProbLog, C is CEM, A is Alchemy.

Dataset	p			
	EMBLEM-R	EMBLEM-L	EMBLEM-C	EMBLEM-A
IMDB-SP	0.2167	**0.0126**	0.3739	**0.0134**
IMDBu-SP	0.1276	0.1995	**0.001**	**4.5234e-005**
IMDB-SM	me	0.3739	**0.0241**	0.1790
IMDBu-SM	me	0.3739	0.2780	**2.2270e-004**
Cora	**0.011**	0.0729	1	**0.0068**
CoraT	no	**0.0464**	me	me
UW-CSE	**0.0054**	**1.5017e-004**	**0.0088**	**4.9921e-004**

to the fastest other algorithm. On IMDB-SM it reaches the highest area value in less time (only one restart is needed). On IMDBu-SM it still reaches the highest area with one restart but with a longer execution time. Over Cora it has comparable performances with the best other system CEM but in a significantly lower time and over CoraT is one of the few systems to be able to complete learning, with better performances in terms of area and time. Over UW-CSE it has significant better performances with respect to all the algorithms.

Memory errors, that we encountered with some systems over certain datasets, have to be ascribed to the memory needs of the systems; for instance, some of them are not able to manage the LPAD for CoraT because its transitive rules generate large BDDs.

6 Conclusions

We have proposed a technique which applies an EM algorithm for learning the parameters of Logic Programs with Annotated Disjunctions. It can be applied to all languages that are based on the distribution semantics and exploits the BDDs that are built during inference to efficiently compute the expectations for hidden variables.

We executed the algorithm over the real datasets IMDB, UW-CSE and Cora, and evaluated its performances - together with those of four other probabilistic systems - through the AUCPR and AUCROC. These results show that EMBLEM uses less memory than RIB, CEM and Alchemy, allowing it to solve larger problems, as one can see from Table **??** where, for some datasets, not all the mentioned algorithms are able to terminate. Moreover its speed allows to perform a high number of restarts making it escape local maxima and achieve higher AUCPR.

EMBLEM is available in the `cplint` package in the source tree of Yap Prolog and information on its use can be found at `http://sites.google.com/a/unife.it/ml/emblem`.

In the future we plan to extend EMBLEM for learning the structure of LPADs by combining the standard Expectation Maximization algorithm, which optimizes parameters, with structure search for model selection.

References

1. Bellodi, E., Riguzzi, F.: EM over binary decision diagrams for probabilistic logic programs. Tech. Rep. CS-2011-01, ENDIF, Università di Ferrara, Italy (2011)
2. Davis, J., Goadrich, M.: The relationship between Precision-Recall and ROC curves. In: Cohen, W.W., Moore, A. (eds.) Proceedings of the 23rd International Conference on Machine Learning. ACM International Conference Proceeding Series, vol. 148, pp. 233–240. ACM (2006)
3. De Raedt, L., Demoen, B., Fierens, D., Gutmann, B., Janssens, G., Kimmig, A., Landwehr, N., Mantadelis, T., Meert, W., Rocha, R., Santos Costa, V., Thon, I., Vennekens, J.: 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, in NIPS (2008)
4. De Raedt, L., Kimmig, A., Toivonen, H.: ProbLog: A probabilistic prolog and its application in link discovery. In: Veloso, M.M. (ed.) Proceedings of the 20th International Joint Conference on Artificial Intelligence. pp. 2462–2467. AAAI Press (2007)
5. Gutmann, B., Kimmig, A., Kersting, K., Raedt, L.D.: Parameter learning in probabilistic databases: A least squares approach. In: Daelemans, W., Goethals, B., Morik, K. (eds.) Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases. LNCS, vol. 5211, pp. 473–488. Springer (2008)
6. Gutmann, B., Kimmig, A., Kersting, K., Raedt, L.: Parameter estimation in ProbLog from annotated queries. Tech. Rep. CW 583, Department of Computer Science, Katholieke Universiteit Leuven, Belgium (2010)
7. Gutmann, B., Thon, I., De Raedt, L.: Learning the parameters of probabilistic logic programs from interpretations. Tech. Rep. CW 584, Department of Computer Science, Katholieke Universiteit Leuven, Belgium (June 2010)
8. Inoue, K., Sato, T., Ishihata, M., Kameya, Y., Nabeshima, H.: Evaluating abductive hypotheses using an em algorithm on bdds. In: Boutilier, C. (ed.) Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI). pp. 810–815. Morgan Kaufmann Publishers Inc. (2009)
9. Ishihata, M., Kameya, Y., Sato, T., Minato, S.: Propositionalizing the em algorithm by bdds. In: Zelezn, F., Lavra, N. (eds.) Late Breaking Papers of the 18th International Conference on Inductive Logic Programming. pp. 44–49 (2008)
10. Ishihata, M., Kameya, Y., Sato, T., Minato, S.: Propositionalizing the em algorithm by bdds. Tech. Rep. TR08-0004, Dept. of Computer Science, Tokyo Institute of Technology (2008)
11. Lowd, D., Domingos, P.: Efficient weight learning for Markov logic networks. In: Kok, J.N., Koronacki, J., de Mántaras, R.L., Matwin, S., Mladenic, D., Skowron, A. (eds.) Proceedings of the 18th European Conference on Machine Learning. LNCS, vol. 4702, pp. 200–211. Springer (2007)
12. Meert, W., Struyf, J., Blockeel, H.: Learning ground CP-Logic theories by leveraging Bayesian network learning techniques. Fundamenta Informaticae 89(1), 131–160 (2008)

13. Neapolitan, R.: Learning Bayesian Networks. Prentice Hall, Upper Saddle River, NJ (2003)
14. Poole, D.: The Independent Choice Logic for modelling multiple agents under uncertainty. Artificial Intelligence 94(1-2), 7–56 (1997)
15. Richardson, M., Domingos, P.: Markov logic networks. Machine Learning 62(1-2), 107–136 (2006)
16. Riguzzi, F.: ALLPAD: Approximate learning of logic programs with annotated disjunctions. In: Muggleton, S., Otero, R.P., Tamaddoni-Nezhad, A. (eds.) Proceedings of the 16th International Conference on Inductive Logic Programming. LNCS, vol. 4455, pp. 43–45. Springer (2007)
17. Riguzzi, F.: A top-down interpreter for LPAD and CP-Logic. In: Basili, R., Pazienza, M.T. (eds.) Proceedings of the 10th Congress of the Italian Association for Artificial Intelligence. LNCS, vol. 4733, pp. 109–120. Springer (2007)
18. Riguzzi, F.: ALLPAD: approximate learning of logic programs with annotated disjunctions. Machine Learning 70(2-3), 207–223 (2008)
19. Riguzzi, F.: Extended semantics and inference for the Independent Choice Logic. Logic Journal of the IGPL 17(6), 589–629 (2009)
20. Riguzzi, F., Mauro, N.D.: Applying the information bottleneck to statistical relational learning. Machine Learning (2011), to appear
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 (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. Singla, P., Domingos, P.: Discriminative training of Markov logic networks. In: Veloso, M.M., Kambhampati, S. (eds.) Proceedings of the 20th National Conference on Artificial Intelligence and the 17th Innovative Applications of Artificial Intelligence Conference. pp. 868–873. AAAI Press/The MIT Press (2005)
24. Singla, P., Domingos, P.: Entity resolution with Markov logic. In: Proceedings of the 6th IEEE International Conference on Data Mining. pp. 572–582. IEEE Computer Society (2006)
25. Thayse, A., Davio, M., Deschamps, J.P.: Optimization of multivalued decision algorithms. In: International Symposium on Multiple-Valued Logic. pp. 171–178. IEEE Computer Society Press (1978)
26. Thon, I., Landwehr, N., Raedt, L.D.: A simple model for sequences of relational state descriptions. In: Daelemans, W., Goethals, B., Morik, K. (eds.) Proceedings of the European conference on Machine Learning and Knowledge Discovery in Databases (ECML/PKDD 2008)- Part II. Lecture Notes in Computer Science, vol. 5212, pp. 506–521. Springer-Verlag (2008)
27. Vennekens, J., Denecker, M., Bruynooghe, M.: Cp-logic: A language of causal probabilistic events and its relation to logic programming. Theory and Practice of Logic Programming 9(3), 245–308 (2009)
28. Vennekens, J., Verbaeten, S.: Logic programs with annotated disjunctions. Tech. Rep. CW386, Department of Computer Science, Katholieke Universiteit Leuven, Belgium (2003)
29. Vennekens, J., Verbaeten, S., Bruynooghe, M.: Logic programs with annotated disjunctions. In: Demoen, B., Lifschitz, V. (eds.) Proceedings of the 20th International Conference on Logic Programming. LNCS, vol. 3131, pp. 195–209. Springer (2004)