An Expectation Maximization Algorithm 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 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 tools - 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. 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 various datasets and showed good performances both in terms of speed and memory.

1 Introduction

In the field of Statistical Relational Learning (SRL) logical-statistical languages are used to effectively learn in complex domains involving relations and uncertainty. They have been successfully applied in social networks analysis, entity recognition, information extraction, etc.

Similarly, a large number of works in Logic Programming has attempted to combine logic and probability, among which the *distribution semantics* [11] is a prominent approach. It underlies for example PRISM [11], the Independent Choice Logic, Logic Programs with Annotated Disjunctions (LPADs) [15], ProbLog [3] and CP-logic. The approach is appealing because efficient inference algorithms appeared [3,9], which adopt Binary Decision Diagrams (BDD).

In this paper we present the EMBLEM system for "EM over BDDs for probabilistic Logic programs Efficient Mining" that learns parameters of probabilistic logic programs under the distribution semantics by using an Expectation Maximization (EM) algorithm: it is an iterative method to estimate some unknown parameters Θ of a model, given a dataset where some of the data is missing,to find maximum likelihood estimates of Θ. The translation of these programs into graphical models requires the use of hidden variables and therefore of EM: the main characteristic of our system is the computation of expectations using BDDs. Since there are transformations with linear complexity that can convert a program in a language into the others[2], we will use LPADs for their general syntax.

EMBLEM has been tested on the IMDB, Cora and UW-CSE datasets and compared with RIB [10], LeProbLog [3], Alchemy [8] and CEM, an implementation of EM based on the `cplint` interpreter [9].

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

2 Logic Programs with Annotated Disjunctions

Formally a *Logic Program with Annotated Disjunctions* [15] 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 $body(C_i)$. 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 the same 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. Subsequently we will write $w_\sigma \models Q$ to mean that the query Q is true in the well-founded model of the program w_σ.

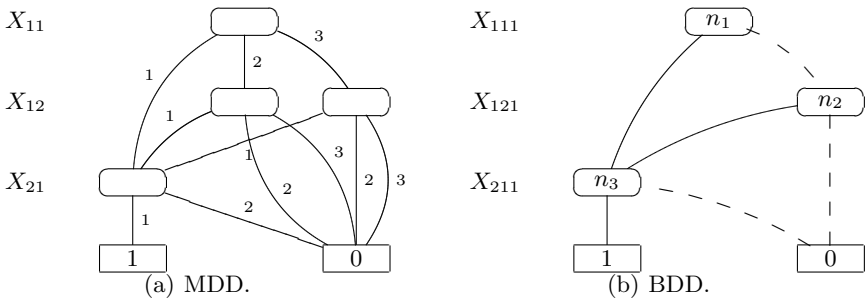
The probability of a query Q according to an LPAD T is given by $P(Q) = \sum_{\sigma\in E(Q)} P(\sigma)$ where $E(Q)$ is $\{\sigma \in \mathcal{S}_T, w_\sigma \models Q\}$, i.e., the set of selections corresponding to worlds where the query is true. To reduce the computational cost of answering queries 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}.

The possible worlds in which a query is true can be represented using a Multi-valued Decision Diagram (MDD). 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 variable X_{ij} associated to $C_i\theta_j$ 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 possible world where Q is true. While building MDDs simplification operations can be applied that delete or merge nodes. 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: nodes at first and second level have three outgoing edges, corresponding to the values of X_{11} and X_{12}, 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*), hence the associated node has two outgoing edges.



(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, the probability of the query can be computed.

Most packages for the manipulation of a decision diagram are however restricted to work on Binary Decision Diagrams (BDD), 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 $\overline{X_{ij1}} \wedge \overline{X_{ij2}} \wedge \ldots \wedge \overline{X_{ijn_i-1}}$. BDDs obtained in this way can be used as well 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 $g(i) = \{j|\theta_j \text{ is a substitution grounding } C_i\}$ 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: $\pi_{i1} = \Pi_{i1}, \ldots \pi_{ik} = \frac{\Pi_{ik}}{\prod_{j=1}^{k-1}(1-\pi_{ij})}$ up to $k = n_i - 1$. Figure 1(b) shows the reduced BDD corresponding to the MDD on the left, with binary variables for each level.

3 EMBLEM

EMBLEM applies the algorithm for performing EM over BDDs, proposed in [14,6], 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 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 with a target predicate in the head 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. Given a program containing 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*.

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. 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 $k = 1, \ldots, n_i - 1$: $\pi_{ik} = \frac{\mathbf{E}[c_{ik1}|Q]}{\mathbf{E}[c_{ik0}|Q] + \mathbf{E}[c_{ik1}|Q]}$

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

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

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 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

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

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

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

where $N(Q)$ is the set of nodes of the BDD, $v(n)$ 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.

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

where π_{ikx} is π_{ik} if x=1 and $(1-\pi_{ik})$ if x=0. $F(n)$ is the *forward probability* [6], the probability mass of the paths from the root to n, while $B(n)$ is the *backward probability* [6], 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

$$P(X_{ijk}=x,Q)=\sum_{n\in N(Q),v(n)=X_{ijk}} e^x(n) \tag{1}$$

For the case of a BDD, i.e., a diagram obtained by applying also the deletion rule, Formula 1 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 n along the 0-branch and whose outgoing edges both point to $child_0(n)$. The correction of formula (1) to take into account of this aspect is applied in the Expectation step.

We now describe EMBLEM in detail. EMBLEM's main procedure consists of a cycle in which the procedures Expectation and Maximization are repeatedly called. The first one returns the log likelihood LL of the data that is used in the stopping criterion: EMBLEM stops when the difference between LL 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 LL.

Procedure Expectation takes as input a list of BDDs, one for each example, and computes the expectation for each one, i.e. $P(Q,X_{ijk}=x)$ for all variables X_{ijk} in the BDD. In the procedure we use $\eta^x(i,k)$ to indicate $\sum_{j\in g(i)}P(Q,X_{ijk}=x)$. 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 $k=1$ to n_i-1 $\mathbf{E}[c_{ikx}]=\mathbf{E}[c_{ikx}]+\eta^x(i,k)/P(Q)$,where $P(Q)$ is the backward probability of the root. Procedure Maximization computes the parameters values for the next EM iteration.

Procedure GetForward traverses the diagram one level at a time starting from the root level, where F(root)=1, and for each node n computes its contribution to the forward probabilities of its children. Function GetBackward

computes the backward probability of nodes by traversing recursively the tree from the root to the leaves. More details can be found in [1].

4 Experiments

EMBLEM has been tested over three real world datasets: IMDB[1], UW-CSE[2] and Cora[3]. We implemented EMBLEM in Yap Prolog[4] and we compared it with RIB [10]; CEM, an implementation of EM based on the `cplint` inference library [9]; LeProblog [4], and Alchemy [8]. All experiments were performed on Linux machines with an Intel Core 2 Duo E6550 (2333 MHz) and 4 GB of RAM.

To compare our results with LeProbLog and Alchemy we exploited the translations of LPADs into ProbLog [2] and MLN [10] respectively.

For the probabilistic logic programming systems (EMBLEM, RIB, CEM and LeProbLog) we consider various options: associating a distinct random variable to each grounding of a probabilistic clause or a single random variable to a non-ground clause, to express whether the clause is used or not (the latter case makes the problem easier); putting a limit on the depth of derivations, thus eliminating explanations associated to derivations exceeding the limit (necessary for problems that contain cyclic clauses, such as transitive closure clauses); 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 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 and AUCROC). 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 [10]. We ran EMBLEM on it with the following settings: no depth bound (theory is acyclic), random variables associated to instantiations of the clauses (learning time is very low) and a number of restarts chosen to match the execution time of EMBLEM with that of the fastest other algorithm.

The queries that LeProbLog take 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` obtained by random sampling. 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 was done also in all the subsequent experiments. For Alchemy we used the preconditioned rescaled conjugate

[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`

gradient discriminative algorithm for every dataset and in this case we specified `sameperson/2` as the only non-evidence predicate.

A second LPAD, also taken from [10], has been created to evaluate the performance of the algorithms when some atoms are unseen. The 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 and AUCROC 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` [10]. We ran EMBLEM on them with no depth bound, one variable for each instantiation of a rule 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 and AUCROC for EMBLEM, LeProblog and Alchemy. For RIB and CEM we obtained a lack of memory error (indicated with "me").

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 [13][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. 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 ran EMBLEM on this LPAD with no depth bound (theory is acyclic), a single variable for each instantiation of a rule (learning time is reasonable) 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 the transitive rules for the predicates `samebib/2`, `samevenue/2`, `sametitle/2`, for a total of 563 rules. In this case we had to run EMBLEM with a depth bound equal to two (theory becomes cyclic and with higher values of depth learning time was overlong) and a single

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

variable for each non-ground rule (LPAD too complex to be treated with a variable for each instantiation); 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, which were contained in the dataset provided with the MLN. 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 and AUCROC obtained by training on four mega-examples and testing on the remaining one. CEM and Alchemy on CoraT gave a 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 applying the closed world assumption to `advisedBy/2`. The theory used was obtained from the MLN of [12][6] and contains 86 rules. We ran EMBLEM on it with a single variable for each instantiation of a rule, a depth bound of two (cyclic theory) and one random restart (to limit time, in comparison with the other faster algorithms).

The annotated queries that LeProbLog takes as input have been created by annotating with 1.0 each positive example and with 0.0 each negative example for `advisedBy/2`. As for Alchemy, we learned weights with `advisedBy/2` as the only non-evidence predicate. Table 1 shows the AUCPR and AUCROC 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 and AUCROC between EMBLEM and RIB/LeProbLog/CEM/Alchemy (significant differences in bold).

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(see AUCPR), with a learning time equal 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 (especially AUCPR) and time. Over UW-CSE it has significant better performances with respect to all the algorithms. Longer learning times are needed for EMBLEM on IMDBu-SM and UW-CSE datasets, but in both cases AUCPR achieves significantly higher values. LeProblog reveals itself to be the closest

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

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, "no" means that the algorithm was not applicable. AUCPR is the area under the Precision-Recall curve, AUCROC is the area under the ROC curve, both averaged over the five folds. E is EMBLEM, R is RIB, L is LeProbLog, C is CEM, A is Alchemy.

Dataset	AUCPR					AUCROC				
	E	R	L	C	A	E	R	L	C	A
IMDB-SP	0.202(500r)	0.199	0.096	0.202	0.107	0.931(500r)	0.929	0.870	0.930	0.907
IMDBu-SP	0.175(40r)	0.166	0.134	0.120	0.020	0.900(40r)	0.897	0.921	0.885	0.494
IMDB-SM	1.000	me	0.933	0.537	0.820	1.000	me	0.983	0.709	0.925
IMDBu-SM	1.000	me	0.933	0.515	0.338	1.000	me	0.983	0.442	0.544
Cora	0.995(120r)	0.939	0.905	0.995	0.469	1.000(120r)	0.992	0.994	0.999	0.704
CoraT	0.991	no	0.970	me	me	0.999	no	0.998	me	me
UW-CSE	0.883	me	0.270	0.644	0.294	0.993	me	0.932	0.873	0.961

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	me	1.49	0.53	1.95

Table 3. Results of t-test on all datasets, relative to AUCPR and AUCROC. 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 - AUCPR				p - AUCROC			
	E-R	E-L	E-C	E-A	E-R	E-L	E-C	E-A
IMDB-SP	0.2167	**0.0126**	0.3739	**0.0134**	0.3436	**0.0012**	0.3507	**0.015**
IMDBu-SP	0.1276	0.1995	**0.001**	**4.5234e-5**	0.2176	0.1402	**0.0019**	**1.01e-5**
IMDB-SM	me	0.3739	**0.0241**	0.1790	me	0.3739	**0.018**	0.2556
IMDBu-SM	me	0.3739	0.2780	**2.2270e-4**	me	0.3739	0.055	**6.54e-4**
Cora	**0.011**	0.0729	1	**0.0068**	**0.0493**	0.0686	0.4569	**0.0327**
CoraT	no	**0.0464**	me	me	no	0.053	me	me
UW-CSE	me	**1.5017e-4**	**0.0088**	**4.9921e-4**	me	**0.0048**	0.2911	**0.0048**

system to EMBLEM from the point of view of performances, able in addition to always complete learning, but with longer times (except for IMDBu-SM and UW-CSE). Looking at the overall results, AUCPR and AUCROC are higher or equal for EMBLEM than the other systems except on IMDBu-SP, where LeProbLog achieves a non-statistically significant higher AUCROC. Differences between EMBLEM and the other systems are statistically significant in 22 out of 43 cases.

5 Related Work

Our work has close connection with various other works. [6] 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 [14] also proposed an EM algorithm over BDD to learn parameters for the CPT-L language. [5] presented the CoPrEM algorithm that performs EM for the ProbLog language. We differ from this work in the construction of BDDs: they build a BDD for an interpretation while we build it for single ground atoms for the specified target predicate(s), the one(s) for which we are interested in good predictions. Moreover CoPrEM treats missing nodes as if they were there and updates the counts accordingly.

Other approaches for learning probabilistic logic programs employ constraint techniques, or use EM, or adopt gradient descent. Among the approaches that use EM, [7] 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 [10] performs parameter learning using the information bottleneck approach, which is an extension of EM targeted especially towards hidden variables. Among the works that use a gradient descent technique we remind LeProbLog [4], which tries to find the parameters of a ProbLog program that minimize the MSE of the query probability and uses BDD to compute the gradient.

Alchemy [8] 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 significantly differ from the languages under the distribution semantics since they extend first-order logic by attaching weights to logical formulas, but do not allow to exploit logic programming techniques.

6 Conclusions

We have proposed a technique which applies an EM algorithm to BDDs 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 four

other systems - through the AUCPR. These results show that EMBLEM uses less memory than RIB, CEM and Alchemy, allowing it to solve larger problems. Moreover its speed allows to perform a high number of restarts making it escape local maxima. In the future we plan to extend EMBLEM for learning LPADs structure.

References

1. Bellodi, E., Riguzzi, F.: EM over binary decision diagrams for probabilistic logic programs. Tech. Rep. CS-2011-01, ENDIF, Università di Ferrara (2011)
2. 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: NIPS Workshop on Probabilistic Programming (2008)
3. De Raedt, L., Kimmig, A., Toivonen, H.: ProbLog: A probabilistic prolog and its application in link discovery. In: International Joint Conference on Artificial Intelligence. pp. 2462–2467 (2007)
4. Gutmann, B., Kimmig, A., Kersting, K., Raedt, L.D.: Parameter learning in probabilistic databases: A least squares approach. In: European Conference on Machine Learning. LNCS, vol. 5211, pp. 473–488. Springer (2008)
5. 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)
6. Ishihata, M., Kameya, Y., Sato, T., Minato, S.: Propositionalizing the em algorithm by bdds. Tech. Rep. TR08-0004, CS Dept., Tokyo Institute of Technology (2008)
7. Meert, W., Struyf, J., Blockeel, H.: Learning ground CP-Logic theories by leveraging Bayesian network learning techniques. Fund. Inf. 89(1), 131–160 (2008)
8. Richardson, M., Domingos, P.: Markov logic networks. Mach. Learn. 62(1-2), 107–136 (2006)
9. Riguzzi, F.: Extended semantics and inference for the Independent Choice Logic. Log. J. IGPL 17(6), 589–629 (2009)
10. Riguzzi, F., Mauro, N.D.: Applying the information bottleneck to statistical relational learning. Mach. Learn. (2011), to appear
11. Sato, T.: A statistical learning method for logic programs with distribution semantics. In: International Conference on Logic Programming. pp. 715–729. MIT Press (1995)
12. Singla, P., Domingos, P.: Discriminative training of Markov logic networks. In: National Conference on Artificial Intelligence. pp. 868–873. AAAI Press/The MIT Press (2005)
13. Singla, P., Domingos, P.: Entity resolution with Markov logic. In: International Conference on Data Mining. pp. 572–582. IEEE Computer Society (2006)
14. Thon, I., Landwehr, N., Raedt, L.D.: A simple model for sequences of relational state descriptions. In: European conference on Machine Learning. LNCS, vol. 5212, pp. 506–521. Springer (2008)
15. Vennekens, J., Verbaeten, S., Bruynooghe, M.: Logic programs with annotated disjunctions. In: International Conference on Logic Programming. LNCS, vol. 3131, pp. 195–209. Springer (2004)