Experimentation of an Expectation Maximization Algorithm for Probabilistic Logic Programs

Elena Bellodi Fabrizio Riguzzi*
ENDIF-Dipartimento di Ingegneria, Università di Ferrara
Via Saragat 1, 44122 Ferrara, Italy
elena.bellodi@unife.it
fabrizio.riguzzi@unife.it

Abstract

Statistical Relational Learning and Probabilistic Inductive Logic Programming are two emerging fields that use representation languages able to combine logic and probability. In the field of Logic Programming, the distribution semantics is one of the prominent approaches for representing uncertainty and underlies many languages such as ICL, PRISM, ProbLog and LPADs. Learning the parameters for such languages requires an Expectation Maximization algorithm since their equivalent Bayesian networks contain hidden variables. EMBLEM (EM over BDDs for probabilistic Logic programs Efficient Mining) is an EM algorithm for languages following the distribution semantics that computes expectations directly on the Binary Decision Diagrams that are built for inference. In this paper we present experiments comparing EMBLEM with LeProbLog, Alchemy, CEM, RIB and LFI-ProbLog on six real world datasets. The results show that EMBLEM is able to solve problems on which the other systems fail and it often achieves significantly higher areas under the Precision Recall and the ROC curves in a similar time.

Keywords Statistical Relational Learning, Probabilistic Inductive Logic Programming, Probabilistic Logic Programming, Expectation Maximization, Binary Decision Diagrams, Logic Programs with Annotated Disjunctions

1 Introduction

In Statistical Relational Learning [8] and Probabilistic Inductive Logic Programming [6], logical-statistical languages are used to effectively learn in complex domains involving relations and uncertainty. They have been successfully applied to many domains such as text classification [19], entity recognition [36] or information extraction [26].

In Logic Programming, the *distribution semantics* [34] is a prominent approach for combining logic and probability. It underlies for example Probabilistic Logic Programs [4], PRISM [34], the Independent Choice Logic [24], Logic Programs with Annotated Disjunctions (LPADs) [41] and ProbLog [7]. The approach is appealing because practical inference algorithms appeared [7, 31, 17], which adopt Binary Decision Diagrams (BDD).

Investigations on techniques for learning languages under the distribution semantics represent a very promising research direction. Various works have started to appear on the subject: [28, 29, 30] adopt constraint optimization techniques to learn a subclass of ground programs, [2, 21, 22] proposed to use the Expectation Maximization (EM) algorithm for inducing parameters and the Structural EM algorithm for inducing the structure of ground LPADs, LeProblog [9, 10] learns the parameters of ProbLog programs by using gradient descent, RIB [32] performs parameter learning using the information bottleneck and LFI-ProbLog [11] computes the expectations needed for EM directly on BDDs.

Recently EMBLEM (EM over BDDs for probabilistic Logic programs Efficient Mining) has been proposed [1], that learns parameters of probabilistic logic programs under the distribution semantics by using an EM algorithm. The main characteristic of EMBLEM is the computation of expectations using BDDs, similarly to LFI-ProbLog. EMBLEM differs from the latter in the construction of BDDs, that is done for queries rather than for whole interpretations. EMBLEM has been tested [1] on the IMDB, Cora and UW-CSE datasets and compared with RIB [32], LeProbLog [7], Alchemy [27] and CEM, an implementation of EM based on the `cplint` interpreter [31]. For all algorithms the Area Under the Curve has been computed for both the Precision Recall curve (AUCPR) and the Receiver Operating

*Corresponding author, Tel/Fax +390532974836

Characteristics curve (AUCROC). EMBLEM achieves higher or equal AUCPR and AUCROC with respect to all other systems, except one case that however is non-statistically significant.

In this paper we extend this experimentation by considering three more datasets - WebKB, MovieLens and Mutagenesis - and one more system, LFI-ProbLog.

The results of this new experimentation confirm those of [1]: the AUCPR and AUCROC for EMBLEM are higher than or equal to those of the other systems in nearly all cases. Differences between EMBLEM and the other systems are statistically significant in favor of EMBLEM in 31 out of 64 cases and against EMBELM in only 1 case.

The paper is organized as follows. Section 2 presents Probabilistic Logic Programming and Section 3 describes EMBLEM. Section 4 discusses related works. Section 5 represents the main contribution of the paper, the experimental results. Section 6 concludes the paper.

2 Probabilistic Logic Programming

A probabilistic logic program under the distribution semantics [34] defines a probability distribution over ground normal logic programs called *worlds*. This distribution is then extended to queries and the probability of a query is obtained by marginalizing the joint distribution of the query and the programs.

The distribution semantics has been defined both for programs that do not contain function symbols, and thus have a finite set of worlds W, and for programs that contain them, that have an infinite set of worlds. We review here the first case for the sake of simplicity. Details of the semantics with function symbols can be found in [34, 25, 33]. The probability of a query Q given a world w is $P(Q|w) = 1$ if $w \models Q$ and 0 otherwise, where $\models$ is truth in the well-founded model [39]. Thus the probability of a query Q is given by $P(Q) = \sum_{w \in W} P(Q, w) = \sum_{w \in W} P(Q|w)P(w) = \sum_{w \in W: w \models Q} P(w)$.

The languages following the distribution semantics differ in the way they define the distribution over logic programs. Each language allows probabilistic choices among atoms in clauses: Probabilistic Logic Programs, PHA, ICL, PRISM, and ProbLog allow probability distributions over facts, while LPADs allow probability distributions over the heads of disjunctive clauses. All these languages have the same expressive power: there are transformations with linear complexity that can convert each one into the others [40, 5]. In this paper we will use LPADs because the syntactic constructs of the other languages can be directly encoded in LPADs.

2.1 Logic Programs with Annotated Disjunctions

In LPADs the alternatives are encoded in the head of clauses in the form of a disjunction in which each atom is annotated with a probability. Each grounding of an annotated disjunctive clause represents a probabilistic choice between a number of ground normal clauses. By choosing a head atom for each grounding of each clause we get a world. The probability of the world is given by the product of the annotations of the atoms selected.

Formally a *Logic Program with Annotated Disjunctions* [41] 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 $X_{ij} = k$. We assume that each random variable is independent of the others. However, this does not limit the set of joint distributions that can be defined on the atoms of the Herbrand base seen as Boolean random variables.A set of atomic choices κ is *consistent* if $(C_i, \theta_j, k) \in \kappa, (C_i, \theta_j, l) \in \kappa \Rightarrow k = l$, 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, i.e., a two-valued model.

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 $E(Q)$ 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 = $flu(robert).$

The program models the fact that if somebody has the flu and the climate is cold, there is the possibility that an epidemic or a pandemic arises. We are uncertain whether the climate is cold but we know for sure that David and Robert have the flu. 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 it generates two random variables X_{11} and X_{12} with three values each, since C_1 has three head atoms (epidemic, pandemic and null). Similarly, clause C_2 has a single grounding $C_2\emptyset$ so it generates a single random variable X_{21} with two values, since C_2 has two head atoms (cold and null). Clauses C_3 and C_4 have a single head atom with probability one, which is omitted, and an empty body: they represent true facts.

T has 18 worlds, the query epidemic is true in 5 of them and its probability is $P(epidemic) = 0.6 \cdot 0.6 \cdot 0.7 + 0.6 \cdot 0.3 \cdot 0.7 + 0.6 \cdot 0.1 \cdot 0.7 + 0.3 \cdot 0.6 \cdot 0.7 + 0.1 \cdot 0.6 \cdot 0.7 = 0.588$.

In the simplified semantics C_1 is associated to a single random variable X_1. In this case T has 6 instances, the query epidemic is true in 1 of them and its probability is $P(epidemic) = 0.6 \cdot 0.7 = 0.42$.

2.2 Decision Diagrams

The possible worlds in which a query is true can be represented using a Multivalued 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. An MDD can be used to represent the set $E(Q)$ by considering the multivalued variables X_{ij} associated to the $C_i\theta_j$s of $ground(T)$. 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 each possible world where Q is true can be associated to a path to a 1-leaf in the MDD. MDDs can be built by combining simpler MDDs using Boolean operators. 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 source node: nodes at the first and second levels have three outgoing edges, corresponding to the values of X_{11} and X_{12}, while nodes at the third level have two outgoing edges, corresponding to the values of X_{21}.

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, so the probability of the query cannot be computed by a summation as in Formula 1. The explanations have first to be made disjoint so that a summation can be computed. Since MDDs split paths on the basis of the values of a variable, the branches are mutually exclusive so a dynamic programming algorithm can be applied for computing the probability [17].

Most packages for the manipulation of decision diagrams are however restricted to work on Binary Decision Diagrams (BDD), i.e., decision diagrams where all the variables are Boolean. These packages offer Boolean operators between BDDs and apply simplification rules to the result of operations in order to reduce them as much as possible, obtaining reduced BDDs. Usually reduced BDDs have a much smaller number of nodes than the equivalent Binary

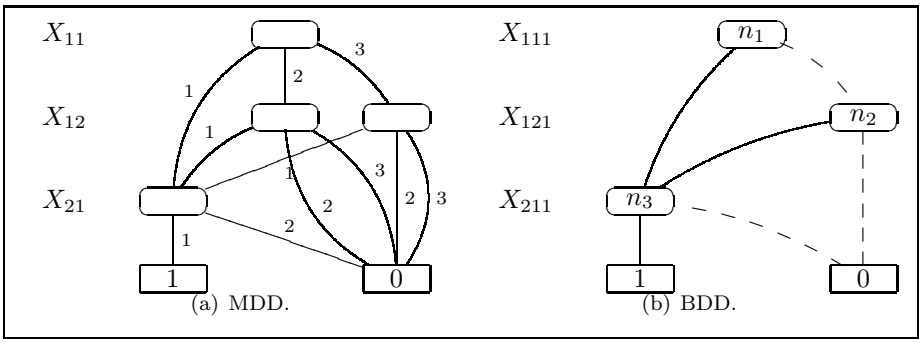

Figure 1: Decision diagrams for Example 1.

Decision Tree (BDT). 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. Various options are possible, we found that the following, proposed in [5], gives the best performance. 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 $\overline{X_{ij1}} \wedge \ldots \wedge \overline{X_{ijk-1}} \wedge X_{ijk}$, and the equation $X_{ij} = n_i$ by means of the conjunction $\overline{X_{ij1}} \wedge \ldots \wedge \overline{X_{ijn_i-1}}$. BDDs obtained in this way can be used for computing the probability of queries as well 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: $\pi_{i1} = \Pi_{i1}$, ..., $\pi_{ik} = \frac{\Pi_{ik}}{\prod_{j=1}^{k-1}(1-\pi_{ij})}$, ..., up to $k = n_i - 1$. Figure 1(b) shows the reduced BDD corresponding to the MDD of Figure 1(a).

3 EMBLEM

EMBLEM applies the algorithm for performing EM over BDDs, proposed in [38, 14], to the problem of learning the parameters of an LPAD. EMBLEM takes as input a number of goals that represent the examples and it generates for each one a BDD encoding its explanations. The typical input for EMBLEM will be a set of interpretations, i.e., sets of ground facts, each describing a portion of the domain of interest. Among the predicates for the input facts the user has to indicate which are target predicates: the facts for these predicates will then form the queries for which the BDDs are built. We assume that also false facts are provided for target predicates, representing negative examples, for which a negated goal is used when finding explanations. The target 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. Using open-world predicates introduces an asymmetry with respect to the target predicates, since when trying to prove a negative example, EMBLEM does not use other negative examples to prove the falsity of a positive body literal, it only checks for the absence of matching positive facts and for failure of matching rules. We did not introduce this feature in EMBLEM to keep the algorithm simple but we intend to investigate in the future if this feature can help in improving performances.

If the second 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.

Let us now present the formulas for the expectation and maximization phases in the case of a single example Q:

- Expectation: compute $\mathbf{E}[c_{ik0}|Q]$ 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 $x \in \{0, 1\}$, and $j \in g(i)$. $\mathbf{E}[c_{ikx}|Q]$ is given by $\sum_{j \in g(i)} P(X_{ijk} = x|Q)$.
- Maximization: compute 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 all examples simply sum up when computing $\mathbf{E}[c_{ijx}]$.

3.1 Expectation Computation

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

$$\begin{aligned} P(X_{ijk} = x, Q) &= \sum_{\sigma \in E(Q)} P(Q, X_{ijk} = x, \sigma) = \sum_{\sigma \in E(Q)} P(Q|\sigma) P(X_{ijk} = x|\sigma) P(\sigma) \\ &= \sum_{\sigma \in E(Q)} P(X_{ijk} = x|\sigma) P(\sigma) = \sum_{\sigma \in E(Q)} P(X_{ijk} = x|\sigma) \prod_{(C_o,\theta,p)\in\sigma} \pi_{op} \end{aligned}$$

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 BDT,

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

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 BDT 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, 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-branch from the 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. So

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

where π_{ikx} is π_{ik} if $x = 1$ and $(1 - \pi_{ik})$ if $x = 0$, and $F(n) = \sum_{\rho_n \in R_n(Q)} \prod_{d \in \rho_n} \pi(d)$ is the *forward probability* [14], the probability mass of the paths from the root to n, and $B(n) = \sum_{\rho^n \in R^n(Q)} \prod_{d \in \rho^n} \pi(d)$ is the *backward probability* [14], the probability mass of paths from n to the 1 leaf. Here $R^n(Q)$ is the set 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

$$P(X_{ijk} = x, Q) = \sum_{n \in N(Q) \wedge v(n) = X_{ijk}} e^x(n) \tag{2}$$

For the case of a BDD, i.e., a diagram obtained by applying also the deletion rule, Formula 2 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 (2) to take into account this aspect is applied in the Expectation step.

In fact, suppose that a node n associated to variable Y has a level higher than variable X_{ijk} and suppose that $child_0(n)$ is associated to variable W that has a level lower than variable X_{ijk}. The nodes associated to variable X_{ijk} have been deleted from the paths from n to $child_0(n)$. One can imagine that the current BDD has 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 original BDD can be re-obtained by applying a deletion operation that merges the two paths passing through m. The probability mass of the two paths that were merged was $e^0(n)(1-\pi_{ik})$ and $e^0(n)\pi_{ik}$ for the paths passing through the 0-child and 1-child of m respectively.

Formally, let $Del^x(X)$ be the set of nodes n such that the level of X is below that of n and is above that of $child_x(n)$, i.e., X is deleted between n and $child_x(n)$. For the BDD in Figure 1(b), for example, $Del^1(X_{111}) = \{\}$, $Del^0(X_{111}) = \{\}$, $Del^1(X_{121}) = \{n_1\}$, $Del^0(X_{121}) = \{\}$, $Del^1(X_{211}) = \{\}$, $Del^0(X_{211}) = \{n_2\}$. Then

$$P(X_{ijk}=0,Q) = \sum_{n\in N(Q)\wedge v(n)=X_{ijk}} e^x(n) + (1-\pi_{ik})\left(\sum_{n\in Del^0(X_{ijk})} e^0(n) + \sum_{n\in Del^1(X_{ijk})} e^1(n)\right)$$

$$P(X_{ijk}=1,Q) = \sum_{n\in N(Q)\wedge v(n)=X_{ijk}} e^x(n) + \pi_{ik}\left(\sum_{n\in Del^0(X_{ijk})} e^0(n) + \sum_{n\in Del^1(X_{ijk})} e^1(n)\right)$$

We now describe EMBLEM in detail.

3.2 EMBLEM's Algorithm

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 the LL of the current iteration and that of the previous iteration drops below a threshold ϵ or when this difference is below a fraction δ of the previous 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. 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 account of 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 leaves to the root. The values of the forward and backward probabilities for the BDD of Figure 1(b) are shown in Figure 2. More details can be found in [1].

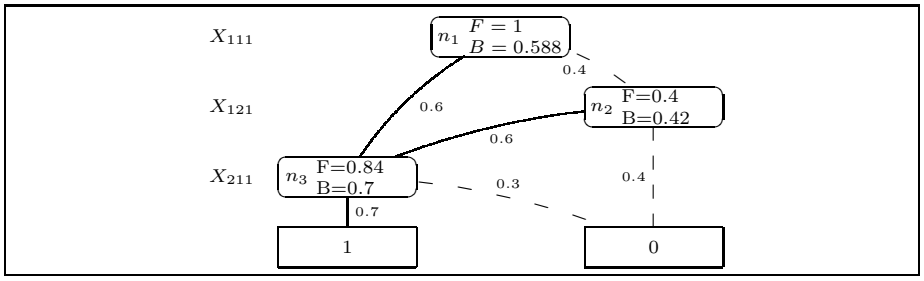

Figure 2: Forward and backward probabilities on a BDD. F indicates the forward probability and B the backward probability of each node.

4 Related Work

Our work has close connection with various other works. [15, 14] proposed an EM algorithm for learning 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 [38] also proposed an EM algorithm over BDD to learn parameters for the CPT-L language, that is less expressive than LPADs. [13] applies the algorithm of [15, 14] to the problem of computing the probabilistic parameters of abductive explanations. [11] presented the LFI-ProbLog algorithm that performs EM for the ProbLog language by learning from partial interpretations and computing the expectations over BDDs. We differ from this work in the construction of BDDs: they build a BDD for a whole partial 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 LFI-ProbLog treats missing nodes as if they were there and updates the counts accordingly.

Other approaches for learning probabilistic logic programs can be classified into three categories: those that employ constraint techniques, those that use EM and those that adopt gradient descent. In the first class, the algorithms of [28, 29, 30] learn a subclass of ground programs and then apply mixed integer linear programming to identify a subset of the clauses that form a solution. Among the approaches that use EM, [22] first proposed to use it to induce parameters of LPADs. However, their approach requires generating the underlying Bayesian network, which can be costly, specially if the program is not originally ground. Moreover [22] proposed to use the Structural EM algorithm to induce ground LPADs structures; again their algorithm works on the underlying Bayesian network. RIB [32] performs parameter learning using the information bottleneck approach, which is an extension of EM targeted especially towards hidden variables; it works best when interpretations have the same Herbrand base, which is not always the case. The PRISM system [34] is one of the first learning algorithms based on EM: it exploits Logic Programming techniques for computing expectations but imposes strong restrictions on the language. In [18] the authors use EM to learn the structure of first-order rules with associated probabilistic uncertainty parameters. Their approach involves generating the underlying graphical model using a Knowledge-Based Model Construction approach; EM is then applied on the graphical model. Among the works that use a gradient descent technique, LeProbLog [9] tries to find the parameters of a ProbLog program that minimize the mean squared error of the queries' probability and uses BDDs to compute the gradient.

Alchemy [27] is a state of the art SRL system that offers various tools for inference, weight learning and structure learning of Markov Logic Networks (MLNs). [19] discusses how to perform weight learning by applying gradient descent of the conditional likelihood of queries for target predicates. 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.

5 Experiments

EMBLEM has been tested[1] over several real world datasets: IMDB, Cora, UW-CSE, WebKB, MovieLens and Mutagenesis. EMBLEM has been compared with LeProbLog [9], LFI-ProbLog [11], Alchemy [27], RIB [32] and CEM, an implementation of EM based on the `cplint` inference library [31].

To compare our results with LeProbLog we exploited the translation of LPADs into ProbLog proposed in [5], in which a disjunctive clause with k head atoms and vector of variables $\mathbf{X}$ is modeled with k ProbLog clauses and $k-1$ probabilistic facts with variables $\mathbf{X}$. For Alchemy we exploited the translation between LPADs and MLN used in [32] and inspired by the translation between ProbLog and MLN proposed in [10]. An MLN clause is translated into an LPAD clause in which the head atoms of the LPAD clause are the *null* atom plus the positive literals of the MLN clause while the body atoms are the negative literals.

For the probabilistic logic programming systems (EMBLEM, LeProbLog, CEM, RIB and LFI-ProbLog), 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 (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), and setting the number of restarts for EM based algorithms.

All experiments except one (see below) for the probabilistic logic programming systems have been performed using open-world predicates, meaning that, when resolving a literal for a target predicate, both facts in the database and rules are used to prove it. As far as the choice to associate random variables either to ground or non-ground clauses is concerned, all experiments have been run first with the most difficult setting (a single random variable for each grounding) and have been re-run with the second easy setting only if EMBLEM failed to terminate under the first one. All experiments used the same value of the thresholds ϵ and δ for stopping the EM cycle.

The datasets are partitioned into four, five or ten mega-examples, where each mega-example contains a connected group of facts and individual mega-examples are independent of each other. A cross-validation approach has been adopted in the experiments: of the four (five, ten) mega-examples, a single example in turn is retained for testing and

[1] All experiments were performed on Linux machines with an Intel Core 2 Duo E6550 (2333 MHz) and 4 GB of RAM.

the remaining ones are used as training data; at the end average values of AUCPR and AUCROC are reported. The terms fold and mega-example will be used interchangeably in the following. The datasets are described in Table 1 in terms of target predicates, number of different constants, number of different predicates, number of examples for the target predicate(s), number of tuples (ground atoms) in the interpretations, number of folds (mega-examples).

Table 1: Characteristics of the six datasets for the experiments: target predicates, number of constants, of predicates, of tuples, of tuples for target predicates only and of folds.

Dataset	Target Preds	Constants	Predicates	Tuples	Facts for target predicates	Folds
IMDB	`sameperson(per1,per2)`(SP)/ `samemovie(mov1,mov2)`(SM)	316	10	1540	1072	5
Cora	`samebib(cit1,cit2)` `sameauthor(aut1,aut2)` `samevenue(ven1,ven2)` `sametitle(tit1,tit2)`	3079	10	378589	63262	5
UW-CSE	`advisedBy(per1,per2)`	1158	22	3212	4191	5
WEBKB	`coursePage(page)` `facultyPage(page)` `studentPage(page)` `researchProjectPage(page)`	4942	8	290973	16668	4
Movielens	`rating(user,movie,rating)`	2627	7	169124	129779	5
Mutagenesis	`active(drug)`	7045	18	15249	184	10

The IMDB dataset [23][2] regards movies, their directors and the actors who appear in them. Each director is associated to genres based on the genres of the movies he or she directed; it is divided into five mega-examples each containing data regarding four movies.

We defined 4 different LPADs, two for predicting the target predicate `sameperson`(`per1`, `per2`), and two for predicting `samemovie`(`movie1`, `movie2`), on the basis of the relations among actors, their movies and their directors. We have 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 [32]:

```
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 indicating a tunable parameter to be learned by EMBLEM. We ran EMBLEM, LeProbLog, RIB, CEM and LFI-ProbLog on it with the following settings: no depth bound (the theory is acyclic) and random variables associated to instantiations of clauses. EMBLEM was run with a number of restarts chosen to match its execution time with that of the fastest other algorithm.

The queries taken as input by LeProbLog were obtained by annotating with 1.0 each positive example and with 0.0 each negative example, those taken as input by LFI-ProbLog by annotating with *true* each positive example and with *false* each negative example. The same procedure has been adopted in all experiments. We ran LeProbLog and LFI-ProbLog for a maximum of 100 iterations or until the difference in Mean Squared Error (MSE) (for LeProbLog) or log likelihood (for LFI-ProbLog) between two iterations got smaller than 10^{-5}. Except where otherwise noted, we used these parameters for all experiments.

For Alchemy we used the preconditioned rescaled conjugate gradient discriminative algorithm [19] for every dataset and in this case we specified `sameperson/2` as the only non-evidence predicate.

A second LPAD, also taken from [32], 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).
```

[2]Available at `http://alchemy.cs.washington.edu/data/imdb`.

```
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. Alchemy was run with the `−withEM` option that turns on EM learning. The other parameters for all systems were set as before.

Then we drew the Precision-Recall and the Receiver Operating Characteristics curves and computed the Area Under the Curve (AUCPR and AUCROC) for both LPADs. Tables 2 and 3 show the AUCPR and AUCROC averaged over the five folds for EMBLEM, LeProbLog, Alchemy, RIB, CEM and LFI-ProbLog. Results are shown respectively in the IMDB-SP and IMDBu-SP rows. Table 4 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 behavior 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, LeProbLog, RIB, CEM and LFI-ProbLog on these two programs with no depth bound (the theory is acyclic) and one variable for each instantiation of each rule. For EMBLEM we used one random restart (since we already obtained an AUCPR and AUCROC of 1). We ran LeProbLog, Alchemy and LFI-ProbLog with the same settings as IMDB-SP and IMDBu-SP, by replacing `sameperson/2` with `samemovie/2`. Tables 2 and 3 show, in the IMDB-SM and IMDBu-SM rows, the average AUCPR and AUCROC for EMBLEM, LeProbLog, Alchemy and CEM. RIB and LFI-ProbLog in this case gave a memory error (indicated with "me"), due to the exhaustion of the available stack space during the execution of the algorithm.

The Cora dataset [20] contains citations to computer science research papers from the Cora Computer Science Research Paper Engine. We used the version of the dataset of [35][3]. For each citation we know the title, authors, venue and the words that appear in it. The dataset encodes a problem of information integration from multiple sources and in particular an entity resolution problem. Citations of the same paper often appear differently and the task is to determine which citations are referring to the same paper, by predicting the predicate `samebib(cit1,cit2)`. The database is composed of five mega-examples and contains facts for the predicates `samebib(cit1,cit2)`, `sameauthor(aut1,aut2)`, `sametitle(tit1,tit2)`, `samevenue(ven1,ven2)` - set as target predicates - and `haswordauthor(author,word)`, `haswordtitle(title,word)`, `haswordvenue(venue,word)`.

From the MLN proposed in [36][4] 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).
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. Positive and negative examples for the four target predicates were already available in the version of the dataset we used. We ran EMBLEM, LeProLog, RIB, CEM and LFI-ProbLog on this LPAD with no depth bound (the theory is acyclic) and a single variable for each instantiation of a rule. We ran EMBLEM with a number of restarts chosen to match its execution time with that of the fastest other algorithm and LFI-ProbLog for a maximum of only 10 iterations (or until the difference in MSE between two iterations got smaller than 10^{-5}) due to its long learning time.

[3] Available at `http://alchemy.cs.washington.edu/data/cora`.

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

The second LPAD adds to the previous one the transitive rules for the predicates `samebib/2`, `samevenue/2`, `sametitle/2` and `sameauthor/2`:

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

for a total of 563 rules. In this case we had to run EMBLEM, LeProbLog, CEM and LFI-ProbLog with a depth bound equal to two (the theory is cyclic and larger depth values led to excessive learning time), a single variable for each non-ground rule and one restart. For LeProbLog, we separately learned the four predicates because learning the whole theory at once gave a lack of memory error. This is equivalent to using a closed-world setting. For Alchemy, we learned weights with the four predicates as the non-evidence predicates. Tables 2 and 3 show in the Cora and CoraT (Cora Transitive) rows the average AUCPR and AUCROC. On CoraT Alchemy, CEM and LFI-ProbLog gave a memory error, for a segmentation fault the first one (by the `learnwts` command) and for memory exhaustion the others, 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 [27][5] contains information about the Computer Science Department of the University of Washington regarding students, professors, courses, quarters, professors' publications, etc. The database is split into five mega-examples, each with facts for a particular departmental area (AI, graphics, programming languages, systems and theory). The interest in this dataset has emerged in the context of social network analysis, where one seeks to reason about a group of people: in particular link prediction tackles the problem of predicting relationships from people's attributes, and UW-CSE represents a benchmark in that direction if we try to predict which professors advise which graduate students. Hence, our target predicate was set to `advisedBy(per1,per2)`.

The theory used has been obtained from the MLN of [35][6] and contains 86 rules. We ran EMBLEM, LeProbLog, CEM, RIB and LFI-ProbLog on it with a single variable for each non-ground rule and a depth bound of two (the theory is cyclic). For EMBLEM we used one random restart (to make the time comparable with that of the other fastest algorithm). The parameters for LeProbLog and LFI-ProbLog were set as for the IMDB experiments. For Alchemy, we learned weights with `advisedBy/2` as the only non-evidence predicate. Tables 2 and 3 show the AUCPR and AUCROC averaged over the five folds for all algorithms. RIB and LFI-ProbLog in this case exhausted the available memory.

The WebKB dataset [3][7] consists of labeled web pages from the computer science departments of four universities, along with the words on the web pages and the links among them. We used the same set of training examples of [3] which differs from the one used in [19, 11]. Each web page is labeled with some subset of the following categories: student, faculty, research project and course. This dataset may be seen as a text classification problem, since we wish to infer the page's class given the information about words and links.

The dataset is split into four mega-examples, one for each university. Our goal is to predict the four predicates `coursePage(page)`, `facultyPage(page)`, `studentPage(page)` and `researchProjectPage(page)`, representing the various possible page's classes, for which the dataset contains both positive and negative examples.

The theory was obtained by translating the MLN of [19][8] into an LPAD. The theory contains a rule of the form

```
<class1>Page(Page1):p :- linkTo(Page2,Page1),<class2>Page(Page2).
```

for each couple of classes `(<class1>,<class2>)`, and rules of the form

```
<class>Page(Page):p :- has(<word>,Page).
```

for each possible class and word. The resulting LPAD contains 3112 rules. Examples of rules are:

```
coursePage(Page1):p :- linkTo(Page2,Page1),coursePage(Page2).
coursePage(Page):p :- has('abstract',Page).
```

Running EMBLEM with a depth bound equal to the lowest value (two) and an open-world setting gave a lack of memory error, so we used a closed-world setting for the target predicates in the body of clauses, resolving target predicates only with facts in the database. Moreover, we used a single variable for each rule instantiation and one random restart to match the execution time with that of the other fastest algorithm. For Alchemy, we learned weights

[5] Available at `http://alchemy.cs.washington.edu/data/uw-cse`.
[6] Available at `http://alchemy.cs.washington.edu/mlns/uw-cse`.
[7] Available at `http://alchemy.cs.washington.edu/data/webkb`.
[8] Available at `http://alchemy.cs.washington.edu/mlns/webkb`.

with the target predicates specified as non-evidence predicates. We also set the flag `-noAddUnitClauses` to 1 (unit predicates with attached weights are not added) since otherwise inference would give a lack of memory error. For LeProbLog and LFI-ProbLog, we ran them on a sample containing respectively 5% and 1% of the training set since the complete set of training examples led to exceedingly long learning times, and in addition LFI-ProbLog was run for a maximum of 10 iterations. Despite that, LFI-ProbLog spent anyway a considerable time. Tables 2 and 3 show the AUCPR and AUCROC averaged over the four mega-examples. RIB and CEM in this case terminated for lack of memory.

The MovieLens [12] dataset contains information about movies, users and ratings that users expressed about movies. We used the version of the dataset of [16][9] containing 82623 ratings and 170143 total ground atoms. For each movie the dataset records the genres to which it belongs, by means of predicates of the form `<genre>(movie, <gen_value>)`, where `<genre>` can be either `drama`, `action` or `horror` and `gen_value` is either `<genre>_0`, if the movie does not belong to the genre, or `<genre>_1`, if the movie belongs to the genre. The age, gender and occupation of users are recorded. Ratings from users on the movies range from 1 to 5. This dataset can be used to build a recommender system, i.e. a system that suggests items of interest to users based on their previous preferences, the preferences of other users, and attributes of users and items. Hence the target predicate is `rating(user,movie,rating)`: we wish to predict the rating on a movie by a user.

The dataset is split into five mega-examples. The theory we used contains 4 rules:

```
rating(A,B,R):p:- rating(A,C,R),B\==C,drama(B,drama_1),drama(C,drama_1).
rating(A,B,R):p:- rating(A,C,R),B\==C,action(B,action_1),action(C,action_1).
rating(A,B,R):p:- rating(A,C,R),B\==C,horror(B,horror_1),horror(C,horror_1).
rating(A,B,R):p:- rating(C,B,R),A\==C.
```

We ran EMBLEM, LeProbLog, RIB, CEM and LFI-Problog on it with a single variable for each non-ground rule and a depth bound of two (the theory is cyclic). EMBLEM was run with one random restart to match the execution time with that of the other fastest algorithm. Tables 2 and 3 show the AUCPR and AUCROC averaged over the five folds for all the algorithms. RIB, CEM, Alchemy and LFI-ProbLog gave a memory error.

The Mutagenesis dataset [37] contains information about a number of aromatic and heteroaromatic nitro drugs, including their chemical structures in terms of atoms, bonds and a number of molecular substructures such as six and five membered rings, benzenes, phenantrenes and others. The problem here is to predict the mutagenicity of the drugs. The prediction of mutagenesis is important as it is relevant to the understanding and prediction of carcinogenesis, and not all compounds can be empirically tested for mutagenesis, e.g. antibiotics.

Of the compounds, those having positive levels of log mutagenicity are labeled "active" and constitute the positive examples, the remaining ones are "inactive" and constitute the negative examples. The goal is to predict if a drug is active, so the target predicate was `active(drug)`.

We split the dataset into ten mega-examples. The theory has been obtained by running Aleph[10] on it with a ten-fold cross-validation and choosing randomly one of the ten theories for parameters learning. The selected theory contains 17 rules, such as:

```
active(A):p :-  bond(A,B,C,2), bond(A,C,D,1), ring_size_5(A,E).
```

We ran EMBLEM, LeProbLog, RIB, CEM and LFI-ProbLog on it with a single variable for each instantiation of a rule and no depth bound (acyclic theory). EMBLEM was run with one random restart, to match its execution time with that of the fastest other algorithm.

The predicates representing molecular substructures use function symbols to represent lists of atoms, for example, in `ring_size_5(drug,ring)`, `ring` is a list of atoms composing a five membered ring in the drug's structure. Alchemy was not applicable to this dataset because it does not handle function symbols. Tables 2 and 3 show the AUCPR and AUCROC averaged over the ten mega-examples for all the algorithms.

Table 5 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 LeProbLog/Alchemy/RIB/CEM/LFI-ProbLog on all datasets (significant differences in favor of EMBLEM in bold)[11]. From the results we can observe that over IMDB-SP EMBLEM has better performances than the other systems in both AUCPR and AUCROC (except for CEM/AUCPR), with six differences out of ten statistically significant. Moreover, EMBLEM takes less time than all other systems except CEM. Over IMDBu-SP EMBLEM has the best performances except for the AUCROC of LeProbLog, again with six significant differences and a low execution time. Over IMDB-SM, it reaches the highest area value in less time (only one restart is

[9] Available at `http://www.cs.sfu.ca/~oschulte/jbn/dataset.html`.

[10] `http://www.cs.ox.ac.uk/activities/machlearn/Aleph/aleph.html`

[11] Please note that, even if many datasets have a small number of folds, the t-test takes this into account by "spreading" the distribution.

Table 2: Results of the experiments on all datasets in terms of Area Under the Curve PR. 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.

Dataset	AUCPR					
	EMBLEM	LeProbLog	Alchemy	RIB	CEM	LFI-ProbLog
IMDB-SP	0.202(500r)	0.096	0.107	0.199	0.202	0.139
IMDBu-SP	0.175(40r)	0.134	0.020	0.166	0.120	0.019
IMDB-SM	1.000	0.933	0.820	me	0.537	me
IMDBu-SM	1.000	0.933	0.338	me	0.515	me
Cora	0.995(120r)	0.905	0.469	0.939	0.995	0.996
CoraT	0.991	0.975	me	no	me	me
UW-CSE	0.749	0.279	0.294	me	0.644	me
WebKB	0.341	0.065	0.495	me	me	0.072
MovieLens	0.869	0.820	me	me	me	me
Mutagenesis	0.992	0.991	no	0.949	0.961	0.928

Table 3: Results of the experiments on all datasets in terms of Area Under the Curve ROC. 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.

Dataset	AUCROC					
	EMBLEM	LeProbLog	Alchemy	RIB	CEM	LFI-ProbLog
IMDB-SP	0.931(500r)	0.870	0.907	0.929	0.930	0.890
IMDBu-SP	0.900(40r)	0.921	0.494	0.897	0.885	0.500
IMDB-SM	1.000	0.983	0.925	me	0.709	me
IMDBu-SM	1.000	0.983	0.544	me	0.442	me
Cora	1.000(120r)	0.994	0.704	0.992	0.999	0.999
CoraT	0.999	0.998	me	no	me	me
UW-CSE	0.993	0.939	0.961	me	0.873	me
WebKB	0.853	0.512	0.884	me	me	0.545
MovieLens	0.841	0.772	me	me	me	me
Mutagenesis	0.986	0.984	no	0.903	0.919	0.895

needed) and two out of six differences are significant; on IMDBu-SM, it still reaches the highest area with one restart but with a longer execution time and one out of six differences are significant. RIB and LFI-ProbLog are not able to terminate on this dataset. Over Cora, EMBLEM shows the best performances along with CEM and LFI-ProbLog, but in much less time, with four significant differences out of ten. Over CoraT, EMBLEM has slightly better performances (with a much lower learning time) than LeProbLog - the only other system able to complete learning on this more complex theory. Over UW-CSE, it has better performances with respect to all the algorithms for AUCPR and AUCROC with five out of six differences significant. Again RIB and LFI-ProbLog are not able to terminate. Over WebKB, EMBLEM shows significantly better areas with respect to LeProbLog and LFI-ProbLog, and worse areas with respect to Alchemy, with the difference in AUCPR being statistically significant. We remind that this dataset is one of the most problematic: it is the only one where EMBLEM has been run with a closed-world setting for the target predicates (simpler than an open-world setting, which failed in this case) and we reduced the size of the training set for LeProbLog and LFI-ProbLog to contain the computation time. Over MovieLens, EMBLEM achieved significantly better areas with respect to LeProbLog in less time, with the differences statistically significant, while all the others systems were not able to complete. Over Mutagenesis, EMBLEM has better performances than all other systems, with the differences being statistically significant in three out of eight cases. Moreover, it is the fastest.

LeProbLog seems to be the closest system to EMBLEM from the point of view of performances, being able to always complete learning as EMBLEM, but with longer times (except for two cases). On the contrary, RIB and LFI-ProbLog incurred in many difficulties in treating the datasets.

Looking at the overall results, EMBLEM's AUCPR and AUCROC are higher or equal than those of the other systems except on IMDBu-SP, where LeProbLog achieves a non-statistically significant higher AUCROC, and WebKB, where Alchemy achieves a higher AUCROC and a statistically significant higher AUCPR. Differences between EMBLEM and the other systems are statistically significant in favor of EMBLEM in 34 out of 64 cases at the 5% significance level and in 21 out of 64 cases at the 1% significance level.

Table 4: Execution time in hours of the experiments, corresponding to the average over the folds, on all datasets.

Dataset	Time(h)					
	EMBLEM	LeProbLog	Alchemy	RIB	CEM	LFI-ProbLog
IMDB-SP	0.010	0.350	1.540	0.016	0.010	0.037
IMDBu-SP	0.0100	0.2300	1.5400	0.0098	0.0120	0.0570
IMDB-SM	0.00036	0.0050	0.0026	me	0.0051	me
IMDBu-SM	3.2200	0.0121	0.0108	me	0.0467	me
Cora	2.48	13.25	1.30	2.49	11.95	44.07
CoraT	0.38	5.67	me	no	me	me
UW-CSE	2.81	2.92	1.95	me	0.53	me
WebKB	0.048	0.114	0.052	me	me	11.32
MovieLens	0.07	20.01	me	me	me	me
Mutagenesis	2.49e-5	0.130	no	0.040	0.040	0.019

Table 5: Results of t-test on all datasets, relative to AUCPR and AUCROC. p is the p-value of a paired two-tailed t-test between EMBLEM and the other systems (significant differences in favor of EMBLEM at the 5% level in bold). LeP is LeProbLog, A is Alchemy, C is CEM, LFI is LFI-ProbLog.

Dataset	*p*-AUCPR					*p*-AUCROC				
	E-LeP	E-A	E-R	E-C	E-LFI	E-LeP	E-A	E-R	E-C	E-LFI
IMDB-SP	**0.0126**	**0.0134**	0.2167	0.3739	**0.0038**	**0.0012**	**0.015**	0.3436	0.3507	**0.0061**
IMDBu-SP	0.1995	**4.5e-5**	0.1276	**0.001**	**5.5e-5**	0.1402	**1e-5**	0.2176	**0.0019**	**5.3e-7**
IMDB-SM	0.3739	0.1790	me	**0.0241**	me	0.3739	0.2556	me	**0.018**	me
IMDBu-SM	0.3739	**2.2e-4**	me	0.2780	me	0.3739	0.2556	me	0.055	me
Cora	0.0729	**0.0068**	**0.011**	1.000	0.6807	0.0686	**0.0327**	**0.0493**	0.4569	0.7661
CoraT	0.104	me	no	me	me	0.131	me	no	me	me
UW-CSE	**2.6e-4**	**4.9e-4**	me	**0.0088**	me	**0.0276**	**0.0048**	me	0.2911	me
WebKB	**0.0177**	0.0012	me	me	**0.0157**	**0.002**	0.1709	me	me	**0.0053**
MovieLens	**8e-7**	me	me	me	me	**7.6e-7**	me	me	me	me
Mutagenesis	0.16	no	**0.0457**	0.0969	**0.0011**	0.1993	no	0.1088	0.1956	**5.4e-4**

6 Conclusions

EMBLEM applies an EM algorithm for learning the parameters of probabilistic logic programs under the distribution semantics. 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 experimented the algorithm over the real datasets IMDB, Cora, UW-CSE, WebKB, MovieLens and Mutagenesis and evaluated its performances by means of the AUCPR and the AUCROC. These results show that EMBLEM achieves higher areas in all cases except two and that the improvements are statistically significant in 34 out of 66 cases. Moreover, EMBLEM uses less memory than Alchemy, RIB, CEM and LFI-ProbLog allowing it to solve larger problems, and often in less time than LeProbLog.

Work is currently under way to extend EMBLEM for learning the structure of probabilistic logic programs.

References

[1] E. Bellodi and F. Riguzzi. Expectation Maximization over binary decision diagrams for probabilistic logic programs. *Intelligent Data Analysis*, 16(6):1–52, 2012.

[2] H. Blockeel and W. Meert. Towards learning non-recursive LPADs by transforming them into Bayesian networks. In *International Conference on Inductive Logic Programming*, volume 4894 of *LNCS*, pages 94–108. Springer, 2007.

[3] M. Craven and S. Slattery. Relational learning with statistical predicate invention: Better models for hypertext. *Machine Learning*, 43(1-2):97–119, 2001.

[4] E. Dantsin. Probabilistic logic programs and their semantics. In *Russian Conference on Logic Programming*, volume 592 of *LNCS*, pages 152–164. Springer, 1991.

[5] L. De Raedt, B. Demoen, D. Fierens, B. Gutmann, G. Janssens, A. Kimmig, N. Landwehr, T. Mantadelis, W. Meert, R. Rocha, V. Santos Costa, I. Thon, and J. Vennekens. Towards digesting the alphabet-soup of statistical relational learning. In *NIPS Workshop on Probabilistic Programming*, pages 1–3, 2008.

[6] L. De Raedt, P. Frasconi, K. Kersting, and S. Muggleton, editors. *Probabilistic Inductive Logic Programming - Theory and Applications*, volume 4911 of *LNCS*. Springer, 2008.

[7] L. De Raedt, A. Kimmig, and H. Toivonen. ProbLog: A probabilistic prolog and its application in link discovery. In *International Joint Conference on Artificial Intelligence*, pages 2462–2467, 2007.

[8] L. Getoor and B. Taskar, editors. *Introduction to Statistical Relational Learning*. MIT Press, 2007.

[9] B. Gutmann, A. Kimmig, K. Kersting, and L. De Raedt. Parameter learning in probabilistic databases: A least squares approach. In *European Conference on Machine Learning*, volume 5211 of *LNCS*, pages 473–488. Springer, 2008.

[10] B. Gutmann, A. Kimmig, K. Kersting, and L. De Raedt. Parameter estimation in ProbLog from annotated queries. Technical Report CW 583, Department of Computer Science, Katholieke Universiteit Leuven, Belgium, 2010.

[11] B. Gutmann, I. Thon, and L. De Raedt. Learning the parameters of probabilistic logic programs from interpretations. In *European Conference on Machine Learning and Knowledge Discovery in Databases*, volume 6911 of *LNCS*, pages 581–596. Springer, 2011.

[12] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl. An algorithmic framework for performing collaborative filtering. In *International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 230–237. ACM, 1999.

[13] K. Inoue, T. Sato, M. Ishihata, Y. Kameya, and H. Nabeshima. Evaluating abductive hypotheses using an EM algorithm on BDDs. In *International Joint Conference on Artificial Intelligence*, pages 810–815. Morgan Kaufmann, 2009.

[14] M. Ishihata, Y. Kameya, T. Sato, and S. Minato. Propositionalizing the EM algorithm by BDDs. Technical Report TR08-0004, Department of Computer Science, Tokyo Institute of Technology, Tokyo, Japan, 2008.

[15] M. Ishihata, Y. Kameya, T. Sato, and S. Minato. Propositionalizing the EM algorithm by BDDs. In *Late Breaking Papers of the International Conference on Inductive Logic Programming*, pages 44–49, 2008.

[16] H. Khosravi, O. Schulte, T. Man, X. Xu, and B. Bina. Structure learning for Markov logic networks with many descriptive attributes. In *AAAI Conference on Artificial Intelligence*, pages 1–493. AAAI Press, 2010.

[17] A. Kimmig, B. Demoen, L. De Raedt, V. Santos Costa, and R. Rocha. On the implementation of the probabilistic logic programming language problog. *Theory and Practice of Logic Programming*, 11(2-3):235–262, 2011.

[18] D. Koller and A. Pfeffer. Learning probabilities for noisy first-order rules. In *International Joint Conference on Artifical Intelligence*, volume 2, pages 1316–1321. Morgan Kaufmann, 1997.

[19] D. Lowd and P. Domingos. Efficient weight learning for Markov logic networks. In *European Conference on Machine Learning*, volume 4702 of *LNCS*, pages 200–211. Springer, 2007.

[20] A. McCallum, K. Nigam, J. Rennie, and K. Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3(2):127–163, 2000.

[21] W. Meert, J. Struyf, and H. Blockeel. Learning ground CP-logic theories by means of bayesian network techniques. In *International Workshop on Multi-Relational Data Mining*, pages 93–104, 2007.

[22] W. Meert, J. Struyf, and H. Blockeel. Learning ground CP-Logic theories by leveraging Bayesian network learning techniques. *Fundumanta Informaticae*, 89(1):131–160, 2008.

[23] L. Mihalkova and R. J. Mooney. Bottom-up learning of Markov logic network structure. In *International Conference on Machine Learning*, volume 227 of *ACM International Conference Proceeding Series*, pages 625–632. ACM, 2007.

[24] D. Poole. The Independent Choice Logic for modelling multiple agents under uncertainty. *Artificial Intelligence*, 94(1-2):7–56, 1997.

[25] D. Poole. Abducing through negation as failure: stable models within the independent choice logic. *Journal of Logic Programming*, 44(1-3):5–35, 2000.

[26] H. Poon and P. Domingos. Joint inference in information extraction. In *AAAI Conference on Artificial Intelligence*, pages 913–918. AAAI Press, 2007.

[27] M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 62(1-2):107–136, 2006.

[28] F. Riguzzi. Learning logic programs with annotated disjunctions. In *International Conference on Inductive Logic Programming*, volume 3194 of *LNCS*, pages 270–287. Springer, 2004.

[29] F. Riguzzi. ALLPAD: Approximate learning of logic programs with annotated disjunctions. In *International Conference on Inductive Logic Programming*, volume 4455 of *LNCS*, pages 43–45. Springer, 2007.

[30] F. Riguzzi. ALLPAD: Approximate learning of logic programs with annotated disjunctions. *Machine Learning*, 70(2-3):207–223, 2008.

[31] F. Riguzzi. Extended semantics and inference for the Independent Choice Logic. *Logic Journal of the IGPL*, 17(6):589–629, 2009.

[32] F. Riguzzi and N. Di Mauro. Applying the information bottleneck to statistical relational learning. *Machine Learning*, 86(1):89–114, 2011.

[33] F. Riguzzi and T. Swift. Well-definedness and efficient inference for probabilistic logic programming under the distribution semantics. *Theory and Practice of Logic Programming, 25-esimo Convegno Italiano di Logica Computazionale Special Issue*, 2012. to appear.

[34] T. Sato. A statistical learning method for logic programs with distribution semantics. In *International Conference on Logic Programming*, pages 715–729. MIT Press, 1995.

[35] P. Singla and P. Domingos. Discriminative training of Markov logic networks. In *National Conference on Artificial Intelligence*, pages 868–873. AAAI Press/MIT Press, 2005.

[36] P. Singla and P. Domingos. Entity resolution with Markov logic. In *IEEE International Conference on Data Mining*, pages 572–582. IEEE Computer Society, 2006.

[37] A. Srinivasan, S. Muggleton, M. J. E. Sternberg, and R. D. King. Theories for mutagenicity: A study in first-order and feature-based induction. *Artificial Intelligence*, 85(1-2):277–299, 1996.

[38] I. Thon, N. Landwehr, and L. De Raedt. A simple model for sequences of relational state descriptions. In *European conference on Machine Learning*, volume 5212 of *LNCS*, pages 506–521. Springer, 2008.

[39] A. Van Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.

[40] J. Vennekens and S. Verbaeten. Logic programs with annotated disjunctions. Technical Report CW386, Department of Computer Science, Katholieke Universiteit Leuven, Belgium, 2003.

[41] J. Vennekens, S. Verbaeten, and M. Bruynooghe. Logic programs with annotated disjunctions. In *International Conference on Logic Programming*, volume 3131 of *LNCS*, pages 195–209. Springer, 2004.