Learning the Structure of Probabilistic Logic Programs

Elena Bellodi and Fabrizio Riguzzi

ENDIF – University of Ferrara, Via Saragat 1, I-44122, Ferrara, Italy
{elena.bellodi,fabrizio.riguzzi}@unife.it

Abstract. There is a growing interest in the field of Probabilistic Inductive Logic Programming, which uses languages that integrate logic programming and probability. Many of these languages are based on the distribution semantics and recently various authors have proposed systems for learning the parameters (PRISM, LeProbLog, LFI-ProbLog and EMBLEM) or both the structure and the parameters (SEM-CP-logic) of these languages. EMBLEM for example uses an Expectation Maximization approach in which the expectations are computed on Binary Decision Diagrams. In this paper we present the algorithm SLIPCASE for "Structure LearnIng of ProbabilistiC logic progrAmS with Em over bdds". It performs a beam search in the space of the language of Logic Programs with Annotated Disjunctions (LPAD) using the log likelihood of the data as the guiding heuristics. To estimate the log likelihood of theory refinements it performs a limited number of Expectation Maximization iterations of EMBLEM. SLIPCASE has been tested on three real world datasets and compared with SEM-CP-logic and Learning using Structural Motifs, an algorithm for Markov Logic Networks. The results show that SLIPCASE achieves higher areas under the precision-recall and ROC curves and is more scalable.

Keywords: Probabilistic Inductive Logic Programming, Statistical Relational Learning, Structure Learning, Distribution Semantics.

1 Introduction

The ability to model both complex and uncertain relationships among entities is very important for learning accurate models in many domains. This originated a growing interest in the field of Probabilistic Inductive Logic Programming, which uses languages that integrate logic programming and probability. Many of these languages are based on the distribution semantics [24], which underlies, e.g., Probabilistic Logic Programs [5], Probabilistic Horn Abduction [21], PRISM [24], Independent Choice Logic [22], pD [11], Logic Programs with Annotated Disjunctions (LPADs) [28], ProbLog [8] and CP-logic [26]. All these languages have the same expressive power: there are linear transformations from one to the others. Efficient inference algorithms have started to appear for them, which in many cases find explanations for queries and compute their probability by building a Binary Decision Diagram (BDD).

Recently, various approaches for learning the parameters of these languages have been proposed: LeProbLog [12] uses gradient descent while LFI-ProbLog [14] and EMBLEM [2] use an Expectation Maximization approach in which the expectations are computed directly using BDDs.

In this paper we consider the problem of learning both the structure and the parameters of languages based on the distribution semantics. To the best of our knowledge, the only works for learning the structure of languages based on the distribution semantics are [23], where the authors propose an algorithm for theory compression for ProbLog, and [19] where the system SEM-CP-logic for learning ground LPADs is presented.

We propose the algorithm SLIPCASE for "Structure LearnIng of ProbabilistiC logic progrAmS with Em over bdds". It performs a beam search in the space of LPADs using the log likelihood of the data as the guiding heuristics. To estimate the log likelihood of theory refinements it performs a limited number of Expectation Maximization iterations of EMBLEM. SLIPCASE can learn general LPADs including non-ground programs.

The paper is organized as follows. Section 2 presents Probabilistic Logic Programming, concentrating on LPADs. Section 3 discusses related works while Section 4 describes EMBLEM more in detail. Section 5 illustrates SLIPCASE. In Section 6 we present the results of the experiments performed. Section 7 concludes the paper and proposes directions for future work.

2 Probabilistic Logic Programming

The distribution semantics [24] is one of the most interesting approaches to the integration of logic programming and probability. It was introduced for the PRISM language but is shared by many other languages. A program in one of these languages defines a probability distribution over 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 languages following the distribution semantics differ in the way they define the distribution over logic programs but have the same expressive power: there are transformations with linear complexity that can convert each one into the others [27, 7]. In this paper we will use LPADs for their general syntax. We review here the semantics for the case of no function symbols for the sake of simplicity.

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.

Formally a *Logic Program with Annotated Disjunctions* [28] consists of a finite set of annotated disjunctive clauses. An annotated disjunctive clause C_i is of the form $h_{i1} : \Pi_{i1}; \ldots ; h_{in_i} : \Pi_{in_i} :- b_{i1}, \ldots , b_{im_i}$. In such a clause $h_{i1}, \ldots , h_{in_i}$ are logical atoms and $b_{i1}, \ldots , b_{im_i}$ are logical literals, $\{\Pi_{i1}, \ldots , \Pi_{in_i}\}$ are real numbers in the interval $[0, 1]$ such that $\sum_{k=1}^{n_i} \Pi_{ik} \leq 1$. $b_{i1}, \ldots , b_{im_i}$ is called the *body* and is indicated with *body*(C_i). Note that, if $n_i = 1$ and $\Pi_{i1} = 1$, the clause corresponds to a non-disjunctive clause. If $\sum_{k=1}^{n_i} \Pi_{ik} < 1$, the head of the annotated disjunctive clause implicitly contains an extra atom *null* that does not appear in the body of any clause and whose annotation is $1 - \sum_{k=1}^{n_i} \Pi_{ik}$. We denote by *ground*$(T)$ the grounding of an LPAD T.

An *atomic choice* is a triple (C_i, θ_j, k) where $C_i \in T$, θ_j is a substitution that grounds C_i and $k \in \{1, \ldots, n_i\}$. 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 only one head is selected for a ground clause. A *composite choice* κ is a consistent set of atomic choices. The *probability* $P(\kappa)$ *of a composite choice* κ is the product of the probabilities of the individual atomic choices, i.e. $P(\kappa) = \prod_{(C_i,\theta_j,k)\in\kappa} \Pi_{ik}$.

A *selection* σ is a composite choice that, for each clause $C_i\theta_j$ in *ground*(T), contains an atomic choice (C_i, θ_j, k). 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 worlds: $P(w_\sigma) = P(\sigma) = \prod_{(C_i,\theta_j,k)\in\sigma} \Pi_{ik}$. We denote the set of all worlds of a program by W.

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

Let $P(W)$ be the distribution over worlds. The probability of a query Q given a world w is $P(Q|w) = 1$ if $w \models Q$ and 0 otherwise. The probability of a query Q is given by

$$P(Q) = \sum_{w\in W} P(Q, w) = \sum_{w\in W} P(Q|w)P(w) = \sum_{w\in W: w\models Q} P(w) \quad (1)$$

Sometimes a simplification of this semantics can be used to reduce the computational cost of answering queries. In this simplified semantics random variables are directly associated to clauses in the programs rather than to their ground instantiations. So, for a clause C_i, possibly non ground, there is a single random variable X_i. In this way the number of random variables may be significantly reduced and atomic choices take the form (C_i, k), meaning that head h_{ik} is selected from program clause C_i, i.e., that $X_i = k$. In the experiments on the WebKB dataset in Section 6 we use this simplification to reduce the computational cost.

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

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

This 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. 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}. Clause C_2 instead has only one grouding $C_2\emptyset$ so there is one random variable X_{21}. T has 18 worlds, the query *epidemic* is true in 5 of them and its probability is
$P(epidemic) = 0.6\cdot0.6\cdot0.7+0.6\cdot0.3\cdot0.7+0.6\cdot0.1\cdot0.7+0.3\cdot0.6\cdot0.7+0.1\cdot0.6\cdot0.7 = 0.588$. In the simplified semantics C_1 is associated with 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 $P(epidemic) = 0.6 \cdot 0.7 = 0.42$.

It is often unfeasible to find all the worlds where the query is true so inference algorithms find instead *explanations* for the query, i.e. composite choices such that the query is true in all the worlds whose selections are a superset of them. The problem of computing the probability of a query Q can thus be reduced to computing the probability of the function

$$f_Q(\mathbf{X}) = \bigvee_{\kappa\in E(Q)} \bigwedge_{(C_i,\theta_j,k)\in\kappa} X_{ij} = k \tag{2}$$

where $E(Q)$ is the set of explanations for Q. Explanations however, differently from worlds, are not necessarily mutually exclusive with respect to each other, so the probability of the query can not be computed by a summation as in (1). The explanations have first to be made disjoint so that a summation can be computed.

To this purpose Multivalued Decision Diagrams (MDD) [25] are used. A 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 with 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. 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, often with a much smaller number of nodes with respect to the original MDD.

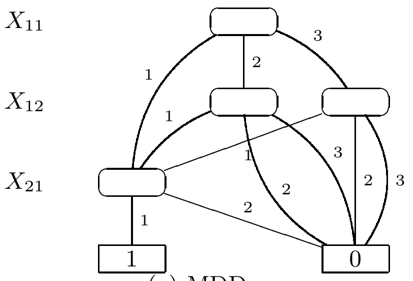
An MDD can be used to represent $f_Q(\mathbf{X})$ and, since MDDs split paths on the basis of the values of a variable, the branches are mutually disjoint so a dynamic programming algorithm can be applied for computing the probability.

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 with the node.

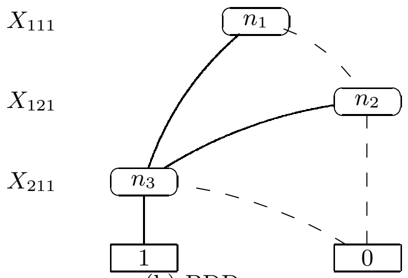
Most packages for the manipulation of decision diagrams are however restricted to work on Binary Decision Diagrams (BDD), i.e., decision diagrams where all the variables are Boolean. These packages offer Boolean operators among BDDs and apply simplification rules to the result of operations in order to reduce as much as possible the BDD size, obtaining a reduced one.

A node n in a BDD has two children: the 1-child, indicated with $child_1(n)$, and the 0-child, indicated with $child_0(n)$. When drawing BDDs, rather than using edge labels, the 0-branch - the one going to the 0-child - is distinguished from the 1-branch by drawing it 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 [7], gives the best performance. For a multi-valued 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 $\overline{X_{ij1}} \wedge \ldots \wedge \overline{X_{ijn_i-1}}$. The BDD corresponding to the MDD of Figure 1(a) is shown in Figure 1(b). 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 $P(X_{ijk} = 1)$. The parameters are obtained from those of multivalued variables in this way: $\pi_{i1} = \Pi_{i1}, \ldots \pi_{ik} = \frac{\Pi_{ik}}{\prod_{j=1}^{k-1}(1-\pi_{ij})}, \ldots$ up to $k = n_i - 1$.



(a) MDD.



(b) BDD.

Fig. 1. Decision diagrams for Example 1.

3 Related Work

SLIPCASE is based on EMBLEM [2], that learns LPAD parameters by using an Expectation Maximization algorithm where the expectations are computed directly on BDDs. EMBLEM is described in detail in Section 4. EMBLEM is similar to LFI-ProbLog [14] but differs in the construction of BDDs: while LFI-ProbLog builds BDDs that represent the application of the whole theory to an interpretation, EMBLEM focuses on a target predicate, the one for which we want to obtain good predictions, and builds BDDs starting from atoms for the target predicate. Moreover, while EMBLEM computes the contributions of deleted paths with a simple post-processing phase (see Section 4), LFI-ProbLog treats missing nodes as if they were there when traversing the tree and updates the counts accordingly.

Previous work on learning the structure of probabilistic logic programs includes [23], that presents an algorithm for performing theory compression on ProbLog programs. Theory compression means removing as many clauses as possible from the theory in order to maximizes the likelihood w.r.t. a set of positive and negative examples. No new clause can be added to the theory.

The system most related to SLIPCASE is SEM-CP-logic [19], which learns parameters and structure of ground CP-logic programs. It performs learning by considering the Bayesian network equivalent to the CP-logic program and by applying techniques for learning Bayesian networks. In particular, it applies the Structural Expectation Maximization (SEM) algorithm [10]: it iteratively generates refinements of the equivalent Bayesian network and it greedily chooses the one that maximizes the BIC score. SLIPCASE differs from this work because it uses log likelihood as a score, it keeps a beam of refinements and it refines a theory by applying standard ILP refinement operators, which allows to learn non ground theories.

Structure learning has been thoroughly investigated for Markov Logic: in [16] the authors proposed two approaches. The first is a beam search that adds a clause at a time to the theory using weighted pseudo-likelihood as a scoring function. The second is called shortest-first search and adds the k best clauses of length l before considering clauses with length $l+1$.

[20] proposed a bottom-up algorithm for learning Markov Logic Networks called BUSL that is based on relational pathfinding: paths of true ground atoms that are linked via their arguments are found and generalized into first-order rules.

In [17] the structure of Markov Logic theories is learned by applying a generalization of relational pathfinding. A database is viewed as a hyper-graph with constants as nodes and true ground atoms as hyperedges. Each hyperedge is labeled with a predicate symbol. First a hypergraph over clusters of constants is found, then pathfinding is applied on this ‘lifted’ hypergraph. The resulting algorithm is called LHL.

In [18] the algorithm Learning Markov Logic Networks using Structural Motifs (LSM) is presented: it is based on the observation that relational data frequently contain recurring patterns of densely connected objects called structural motifs. LSM limits the search to these patterns. As LHL, LSM views a database as a hyper-graph and groups nodes that are densely connected by many paths and the hyperedges connecting the nodes into a motif. Then it evaluates whether the motif appears frequently enough in the data and finally it applies relational pathfinding to find rules. LSM was experimented on various datasets and found to be superior to other methods, thus representing the state of the art in Markov Logic Networks' structure learning and in Statistical Relational Learning in general. We compare SLIPCASE with LSM in Section 6.

A different approach is taken in [3] where the algorithm DSL is presented that performs discriminative structure learning, i.e., it maximizes the conditional likelihood of a set of outputs given a set of inputs. DSL repeatedly adds a clause to the theory by iterated local search, which performs a walk in the space of local optima. We share with this approach the discriminative nature of the algorithm and the scoring function.

4 EMBLEM

EMBLEM [2] applies the algorithm for performing Expectation Maximization (EM) over BDDs proposed in [15] to the problem of learning the parameters of an LPAD. EMBLEM takes as input a number of goals that represent the examples. For each goal it generates the 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, and a theory. The user has to indicate which, among the predicates of the input facts, are target predicates: the facts for these predicates will then form the queries for which the BDDs are built. The predicates can be treated as closed-world or open--world. In the first case the body of clauses is resolved only with facts in the interpretation. In the second case the body of clauses is resolved both with facts in the interpretation and with clauses in the theory. If the last option is set and the theory is cyclic, we use a depth bound on SLD-derivations to avoid going into infinite loops, as proposed by [13]. 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. Expectations are computed directly over BDDs using the algorithm of [15]. The procedure of EMBLEM can be viewed in Algorithm 4 without taking into consideration the N and $Nmax$ variables, used later for a "bounded version" of the algorithm.

The Expectation (see Algorithm 1) phase 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 $x \in \{0, 1\}$ and for all $j \in g(i) := \{j|\theta_j$ is a substitution grounding $C_i\}$, i.e., $\mathbf{E}[c_{ikx}|Q]$ is given by

$$\sum_{j \in g(i)} P(X_{ijk} = x|Q)$$

In the Maximization phase (see Algorithm 2), π_{ik} is computed for all rules C_i and $k = 1, \ldots, n_i - 1$ as

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

Algorithm 1 Procedure Expectation

```
 1: function EXPECTATION($BDDs$)
 2:     $LL = 0$
 3:     for all $BDD \in BDDs$ do
 4:         for all $i \in Rules$ do
 5:             for $k = 1$ to $n_i - 1$ do
 6:                 $\eta^0(i,k) = 0;\ \ \eta^1(i,k) = 0$
 7:             end for
 8:         end for
 9:         for all variables X do
10:             $\varsigma(X) = 0$
11:         end for
12:         GETFORWARD($root(BDD)$)
13:         $Prob$=GETBACKWARD($root(BDD)$)
14:         $T = 0$
15:         for $l = 1$ to $levels(BDD)$ do
16:             Let $X_{ijk}$ be the variable associated to level $l$
17:             $T = T + \varsigma(X_{ijk})$
18:             $\eta^0(i,k) = \eta^0(i,k) + T \times (1 - \pi_{ik})$
19:             $\eta^1(i,k) = \eta^1(i,k) + T \times \pi_{ik}$
20:         end for
21:         for all $i \in Rules$ do
22:             for $k = 1$ to $n_i - 1$ do
23:                 $\mathbf{E}[c_{ik0}] = \mathbf{E}[c_{ik0}] + \eta^0(i,k)/Prob$
24:                 $\mathbf{E}[c_{ik1}] = \mathbf{E}[c_{ik1}] + \eta^1(i,k)/Prob$
25:             end for
26:         end for
27:         $LL = LL + \log(Prob)$
28:     end for
29:     return $LL$
30: end function
```

Algorithm 2 Procedure Maximization

```
1: procedure MAXIMIZATION
2:     for all $i \in Rules$ do
3:         for $k = 1$ to $n_i - 1$ do
4:             $\pi_{ik} = \frac{\mathbf{E}[c_{ik1}]}{\mathbf{E}[c_{ik0}]+\mathbf{E}[c_{ik1}]}$
5:         end for
6:     end for
7: end procedure
```

The values of $P(X_{ijk} = x|Q)$ for all i, j, k, x are computed directly on the BDDs. First, we must compute the probability mass of each path passing through each node associated with X_{ijk} and going down its x-branch. This is done by calculating forward and backward probabilities of each node, i.e., the probability mass of paths from the root to the node and that of the paths from the node to the leaves respectively. $P(X_{ijk} = x|Q)$ is computed from $P(X_{ijk} = x, Q)$, given by

$$P(X_{ijk} = x, Q) = \sum_{n \in N(Q), v(n) = X_{ijk}} F(n)B(child_x(n))\pi_{ikx}$$

where $N(Q)$ is the set of BDD nodes for query Q, $v(n)$ is the variable associated with node n, $F(n)$ is the forward probability of node n and $B(n)$ is the backward probability of node n. 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. By indicating with $e^x(n)$ such an expression we get

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

Computing the forward probability and the backward probability of BDDs' nodes requires two traversals of the graph, so its cost is linear in the number of nodes.

Formula (3) is correct if, when building the BDD, no node has been deleted, i.e., if a node for every variable appears on each path. If this is not the case, the contribution of deleted paths must be taken into account. This is done in the algorithm of [15], by keeping an array ς with an entry for every level l that stores an algebraic sum of $e^x(n)$: those for nodes in upper levels that do not have a descendant in level l minus those for nodes in upper levels that have a descendant in level l. In this way it is possible to add the contributions of the deleted paths by starting from the root level and accumulating $\varsigma(l)$ for the various levels in a variable T: an $e^x(n)$ value which is added to the accumulator T for level l means that n is an ancestor for nodes in this level. When the x-branch from n reaches a node in a level $l' \leq l$ $e^x(n)$ is subtracted from the accumulator, as it is not relative to a deleted node on the path anymore. This is implemented in a post processing phase in Algorithm 1.

5 SLIPCASE

SLIPCASE learns an LPAD by starting from an initial theory and by performing a beam search in the space of refinements of the theory guided by the log likelihood of the data.

First the parameters of the initial theory are computed using EMBLEM and the theory is inserted in the beam (see Algorithm 3). Then an iterative phase begins, where at each step the theory with the highest log likelihood is removed from the beam. Such a theory is the first of the beam since the theories are kept ordered. Then SLIPCASE finds the set of refinements of the selected theory that are allowed by the language bias. `modeh` and `modeb` declarations in Progol style are used to this purpose. The admitted refinements are: the addition of a literal to a clause, the removal of a literal from a clause, the addition of a clause with an empty body and the removal of a clause. The refinements must respect the input-output modes of the bias declarations and the resulting clauses must be connected. For each refinement, an estimate of the log likelihood of the data is computed by running the procedure BoundedEMBLEM (see Algorithm 4) that performs a limited number of Expectation-Maximization steps. BoundedEMBLEM differs from EMBLEM only in line 10, where it imposes that iterations are at most $NMax$. Once the log likelihood for each refinement is computed, the best theory found so far is possibly updated and each refinement is inserted in order in the beam.

At the end the parameters of the best theory found so far are computed with EMBLEM and the resulting theory is returned. We followed this approach rather than using a scoring function as proposed in [10] for SEM because we found that using the log likelihood was giving better results with a limited additional cost. We think that is due to the fact that, while in SEM the number of incomplete or unseen variables is fixed, in SLIPCASE the revisions can introduce or remove unseen variables from the underlying Bayesian network.

Algorithm 3 Procedure SLIPCASE

```
1: function SLIPCASE($Th, MaxSteps, \epsilon, \epsilon 1, \delta, b, NMax$)
2:     Build $BDDs$
3:     $(LL, Th)$ =EMBLEM($Th, \epsilon, \delta$)
4:     $Beam = [(Th, LL)]$
5:     $BestLL = LL$
6:     $BestTh = Th$
7:     $Steps = 1$
8:     repeat
9:         Remove the first couple $(Th, LL)$ from $Beam$
10:        Find all refinements $Ref$ of $Th$
11:        for all $Th'$ in $Ref$ do
12:            $(LL'', Th'')$ =BoundedEMBLEM($Th', \epsilon, \delta, NMax$)
13:            if $LL'' > BestLL$ then
14:                Update $BestLL, BestTh$
15:            end if
16:            Insert $(Th'', LL'')$ in $Beam$ in order of $LL''$
17:            if $size(Beam) > b$ then
18:                Remove the last element of $Beam$
19:            end if
20:        end for
21:        $Steps = Steps + 1$
22:    until $Steps > MaxSteps$ or $Beam$ is empty or $(BestLL - Previous\_BestLL) < \epsilon 1$
23:    $(LL, ThMax)$ =EMBLEM($BestTh, \epsilon, \delta$)
24:    return $ThMax$
25: end function
```

6 Experiments

We implemented SLIPCASE in Yap Prolog and we tested it on three real world datasets: HIV [1], UW-CSE[1] [16] and WebKB[2] [4]. We compared SLIPCASE with SEM-CP-logic [19] and with LSM [18]. All experiments were performed on

[1] `http://alchemy.cs.washington.edu/data/uw-cse`

[2] `http://alchemy.cs.washington.edu/data/webkb`

Algorithm 4 Procedure BoundedEMBLEM

```
function BoundedEMBLEM($Theory, \epsilon, \delta, NMax$)
    Build $BDDs$
    $LL = -inf$
    $N = 0$
    repeat
        $LL_0 = LL$
        $LL$ = Expectation($BDDs$)
        Maximization
        $N = N + 1$
    until $LL - LL_0 < \epsilon \vee LL - LL_0 < -LL \cdot \delta \vee N > NMax$
    Update the parameters of $Theory$
    return $LL, Theory$
end function
```

Linux machines with an Intel Core 2 Duo E6550 (2333 MHz) processor and 4 GB of RAM.

SLIPCASE offers the following options: putting a limit on the depth of derivations, necessary for problems that contain cyclic clauses; setting the number of iterations $NMax$ for BoundedEMBLEM; setting the size of the beam, setting the greatest number of variables in a learned rule (*max_var*) and of rules (*max_rules*) in the learned theory.

For all experiments with SLIPCASE we used a beam size of 5, *max_var*=5, *max_rules*=10 and $NMax = +\infty$ since we observed that EMBLEM usually converged quickly. For testing, we drew a Precision-Recall curve and a Receiver Operating Characteristics curve, and computed the Area Under the Curve (AUCPR and AUCROC respectively) using the methods reported in [6, 9].

The HIV dataset records mutations in HIV's reverse transcriptase gene in patients that are treated with the drug zidovudine. It contains 364 examples, each of which specifies the presence or not of six classical zidovudine mutations, denoted with the predicates (without arguments): `41L`, `67N`, `70R`, `210W`, `215FY` and `219EQ`. The goal is to discover causal relations between the occurrences of mutations in the virus, so all the predicates were set as target. The input initial theory was composed of six probabilistic clauses of the form *target_mutation* : 0.2. The language bias allows each atom to appear in the head and in the body (for this reason some clauses may contain the same atom both in the head and in the body). We used a five-fold cross-validation approach, by considering a single fold as the grouping of 72 or 73 examples.

We ran SLIPCASE with a depth bound equal to three and obtained a final structure with the following rules for each fold (the programs obtained from the various folds differ only in the learned probabilities):

```
70R:0.402062.
41L:0.682637 :- 215FY.
67N:0.824176 :- 219EQ.
219EQ:0.75 :- 67N.
215FY:0.948452 ; 41L:0.0488947 :- 41L.
210W:0.380175 ; 41L:0.245964 :- 41L, 215FY.
210W:4.73671e-11.
```

```
210W:5.14295e-06.
```

For testing, we computed the probability of each mutation in each example given the value of the remaining mutations. The presence of a mutation in an example is considered as a positive example (positive atom), while its absence as a negative example (negated atom).

For SEM-CP-logic, we tested the learned theory reported in [19] over each of the five folds, with the same method applied for SLIPCASE. For LSM, we used the generative training algorithm to learn weights, because all the predicates were considered as target, with the option `-queryEvidence` (to mean that all the atoms of the query predicates not in the database are assumed false evidence), and the MC-SAT algorithm for inference over the test fold, by specifying all the six mutations as query atoms. Table 1 shows the AUCPR and AUCROC averaged over the five folds for the algorithms (graphs are missing for lack of space).

Two observations can be made with regard to the previous results:

- SLIPCASE is able to achieve higher AUCPR and AUCROC with respect to LSM and SEM-CP-logic;
- a comparison among (1) the theory learned by SLIPCASE, (2) the theory learned by SEM-CP-logic and (3) the mutagenetic tree for the development of zidovudine resistance, reported in [1], where nodes correspond to target mutations and edges to hypothesized causal relations between them, shows that:
  1. The clause `67N :- 219EQ.` is present in all three models;
  2. The clause `41L :- 215FY.` is present in our theory and in the mutagenetic tree, while is present with the opposite direction again in our theory (in the fifth clause) and in the CP-theory;
  3. The relation that links the `210W`'s occurrence to `41L` and `215FY` is found both in the clause `210W ; 41L :-41L, 215FY.` of our theory and in the mutagenic tree's right branch, which specifies the causal relations `41L :- 215FY` and `210W :- 41L.`

The UW-CSE dataset contains information about the Computer Science department of the University of Washington, and is split into five mega-examples, each containing facts for a particular research area. The goal is to predict the `advisedby/2` predicate, namely the fact that a person is advised by another person: this was our target predicate. The input theory for SLIPCASE was composed by two clauses of the form `advisedby(X,Y):0.5.` and the language bias allowed `advisedby/2` to appear only in the head (`modeh` declaration) and all the other predicates only in the body (`modeb`); we ran it with no depth bound. We used a five-fold cross-validation approach. For LSM, we used the preconditioned rescaled conjugate gradient discriminative training algorithm for learning the weights, by specifying `advisedby/2` as the only non-evidence predicate plus the option `-queryEvidence`, and the MC-SAT algorithm for inference over the test fold, by specifying `advisedby/2` as the query predicate. For SEM-CP-logic we could not test any CP-logic theory learned from this dataset - as we did for HIV

- since the implementation of SEM-CP-logic only learns ground theories. For this reason performance data are missing in Table 1. Table 1 shows the average AUCPR and AUCROC for SLIPCASE and LSM: SLIPCASE is able to achieve higher AUCPR and AUCROC with respect to LSM.

The WebKB dataset describes web pages from the computer science departments of four universities. We used the version of the dataset from [4] that contains 4,165 web pages and 10,935 web links, along with words on the web pages. Each web page is labeled with some subset of the categories: student, faculty, research project and course. The goal is to predict these categories from the web pages' words and link structures. We trained on data from three universities and tested on the remaining one. For SLIPCASE, we used a single random variable for each clause instead of one for each grounding of each clause. The language bias allowed predicates representing the four categories both in the head and in the body of clauses. Moreover, the body can contain the atom `linkTo(_Id,Page1,Page2)` (linking two pages) and the atom `has(word,Page)` with `word` a constant. This dataset is quite large, with input files of 15 MB on average. LSM failed on this dataset because the weight learning phase quickly exhausted the available memory on machines with 4 GB of RAM. For the reasons explained above we did not experiment SEM-CP-Logic on this dataset.

Table 1. Results of the experiments in terms of the Area Under the PR Curve and under the ROC Curve averaged over the folds.

Dataset	AUCPR			AUCROC		
	Slipcase	LSM	SEM-CP-logic	Slipcase	LSM	SEM-CP-logic
HIV	0.777	0.381	0.579	0.926	0.652	0.721
UW-CSE	0.034	0.017	-	0.894	0.546	-
WebKB	0.395	-	-	0.712	-	-

Table 2 shows the learning times in hours for the three datasets. We could not include the times for SEM-CP-logic on HIV since they are not mentioned in [19]. The similarity in learning times between HIV and UW-CSE for SLIPCASE despite the difference in the number of predicates for the two domains is due to the different specifications in the language bias for the theory refinements' generation: every predicate in HIV can be used in the clauses' body and in the head, while in UW-CSE only one is allowed for the head.

The table highlights that SLIPCASE has better scalability than LSM.

7 Conclusions

We have presented a technique for learning both the structure (clauses) and the parameters (probabilities in the head of the clauses) of Logic Programs with Annotated Disjunctions, by exploiting the EM algorithm over Binary Decision Diagrams proposed in [2]. It can be applied to all languages that are based on the distribution semantics.

Table 2. Execution time in hours of the experiments on all datasets.

Dataset	Time(h)	
	Slipcase	LSM
HIV	0.010	0.003
UW-CSE	0.018	2.574
WebKB	5.689	-

The code of SLIPCASE is available in the source code repository of the development version of Yap and is included in the `cplint` suite. More information on the system, including a user manual, can be found at `http://sites.unife.it/ml/slipcase`.

We have tested the algorithm over the real datasets HIV, UW-CSE and WebKB, and evaluated its performances - in comparison with LSM and SEM-CP-logic - through the AUCPR and AUCROC. From the results one can note that SLIPCASE has better performances (highest area values) under both metrics.

In the future we plan to test SLIPCASE over other datasets and to experiment with other search strategies, such as using bottom clauses to guide refinements, local search in the space of refinements or bottom-up search such as in [20, 17, 18].

References

1. Beerenwinkel, N., Rahnenführer, J., Däumer, M., Hoffmann, D., Kaiser, R., Selbig, J., Lengauer, T.: Learning multiple evolutionary pathways from cross-sectional data. J. Comput. Biol. 12(6), 584–598 (2005)
2. Bellodi, E., Riguzzi, F.: Expectation Maximization over binary decision diagrams for probabilistic logic programs. Intel. Data Anal. 16(6) (2012)
3. Biba, M., Ferilli, S., Esposito, F.: Discriminative structure learning of markov logic networks. In: International Conference on Inductive Logic Programming. LNCS, vol. 5194, pp. 59–76. Springer (2008)
4. Craven, M., Slattery, S.: Relational learning with statistical predicate invention: Better models for hypertext. Mach. Learn. 43(1/2), 97–119 (2001)
5. Dantsin, E.: Probabilistic logic programs and their semantics. In: Russian Conference on Logic Programming. LNCS, vol. 592, pp. 152–164. Springer (1991)
6. Davis, J., Goadrich, M.: The relationship between Precision-Recall and ROC curves. In: International Conference on Machine Learning. ACM International Conference Proceeding Series, vol. 148, pp. 233–240. ACM (2006)
7. 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: Universal Languages, Systems and Applications (2008)
8. 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. AAAI Press (2007)

9. Fawcett, T.: An introduction to roc analysis. Patt. Recog. Lett. 27(8), 861–874 (2006)
10. Friedman, N.: The Bayesian structural EM algorithm. In: Conference on Uncertainty in Artificial Intelligence. pp. 129–138. Morgan Kaufmann (1998)
11. Fuhr, N.: Probabilistic datalog: Implementing logical information retrieval for advanced applications. J. Am. Soc. Inf. Sci. 51(2), 95–110 (2000)
12. Gutmann, B., Kimmig, A., Kersting, K., Raedt, L.D.: Parameter learning in probabilistic databases: A least squares approach. In: European Conference on Machine Learning and Knowledge Discovery in Databases. LNCS, vol. 5211, pp. 473–488. Springer (2008)
13. Gutmann, B., Kimmig, A., Kersting, K., Raedt, L.: Parameter estimation in ProbLog from annotated queries. Tech. Rep. CW 583, KU Leuven (2010)
14. Gutmann, B., Thon, I., Raedt, L.D.: Learning the parameters of probabilistic logic programs from interpretations. In: European Conference on Machine Learning and Knowledge Discovery in Databases. LNCS, vol. 6911, pp. 581–596. Springer (2011)
15. Ishihata, M., Kameya, Y., Sato, T., Minato, S.: Propositionalizing the em algorithm by bdds. In: Late Breaking Papers of the International Conference on Inductive Logic Programming. pp. 44–49 (2008)
16. Kok, S., Domingos, P.: Learning the structure of markov logic networks. In: International Conference on Machine Learning. pp. 441–448. ACM (2005)
17. Kok, S., Domingos, P.: Learning markov logic network structure via hypergraph lifting. In: International Conference on Machine Learning. p. 64. ACM (2009)
18. Kok, S., Domingos, P.: Learning markov logic networks using structural motifs. In: International Conference on Machine Learning. pp. 551–558. Omnipress (2010)
19. Meert, W., Struyf, J., Blockeel, H.: Learning ground CP-Logic theories by leveraging Bayesian network learning techniques. Fundam. Inform. 89(1), 131–160 (2008)
20. Mihalkova, L., Mooney, R.J.: Bottom-up learning of markov logic network structure. In: International Conference on Machine Learning. pp. 625–632. ACM (2007)
21. Poole, D.: Logic programming, abduction and probability - a top-down anytime algorithm for estimating prior and posterior probabilities. New Gener. Comput. 11(3-4), 377–400 (1993)
22. Poole, D.: The Independent Choice Logic for modelling multiple agents under uncertainty. Artif. Intell. 94(1-2), 7–56 (1997)
23. Raedt, L.D., Kersting, K., Kimmig, A., Revoredo, K., Toivonen, H.: Compressing probabilistic prolog programs. Mach. Learn. 70(2-3), 151–168 (2008)
24. Sato, T.: A statistical learning method for logic programs with distribution semantics. In: International Conference on Logic Programming. pp. 715–729. MIT Press (1995)
25. Thayse, A., Davio, M., Deschamps, J.P.: Optimization of multivalued decision algorithms. In: International Symposium on Multiple-Valued Logic. pp. 171–178. IEEE Computer Society Press (1978)
26. Vennekens, J., Denecker, M., Bruynooghe, M.: CP-logic: A language of causal probabilistic events and its relation to logic programming. The. Pra. Log. Program. 9(3), 245–308 (2009)
27. Vennekens, J., Verbaeten, S.: Logic programs with annotated disjunctions. Tech. Rep. CW386, KU Leuven (2003)
28. 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)