Learning Logic Programs with Annotated Disjunctions

Fabrizio Riguzzi

Dipartimento di Ingegneria, Università di Ferrara, Via Saragat 1
44100 Ferrara, Italy,
friguzzi@ing.unife.it

Abstract. Logic Programs with Annotated Disjunctions (LPADs) provide a simple and elegant framework for integrating probabilistic reasoning and logic programming. In this paper we propose an algorithm for learning LPADs. The learning problem we consider consists in starting from a sets of interpretations annotated with their probability and finding one (or more) LPAD that assign to each interpretation the associated probability. The learning algorithm first finds all the disjunctive clauses that are true in all interpretations, then it assigns to each disjunct in the head a probability and finally decides how to combine the clauses to form an LPAD by solving a constraint satisfaction problem. We show that the learning algorithm is correct and complete.

1 Introduction

There has been recently a growing interest in the field of probabilistic logic programming: a number of works have appeared that combine logic programming or relational representations with probabilistic reasoning. Among these works, we cite: Probabilistic Logic Programs [14], Bayesian Logic Programs [9, 10], Probabilistic Relational Models [7], Context-sensitive Probabilistic Knowledge Bases [15], Independent Choice Logic (ICL) [17] and Stochastic Logic Programs [13, 3].

One of the most recent approaches is Logic Programs with Annotated Disjunctions (LPADs) presented in [22, 21]. In this approach, the clauses of an LPAD can have a disjunction in the head and each disjunct is annotated with a probability. The sum of the probabilities for all the disjuncts in the head of a clause must be one. Clauses with disjunction in the head express a form of uncertain knowledge, for example the clause

$$heads(Coin) \vee tails(Coin) \leftarrow toss(Coin)$$

expresses the fact that, if a coin is tossed, it can land on heads or tails but we don't know which. By annotating the disjuncts with a probability, we can express probabilistic knowledge that we have regarding the facts in the head, for example the clause

$$(heads(Coin) : 0.5) \vee (tails(Coin) : 0.5) \leftarrow toss(Coin), \neg biased(Coin)$$

expresses the fact that, if the coin is not biased, it has equal probability of landing on heads or on tails.

The semantics of LPADs is given in terms of a function π_P^* that, given an LPAD P, assigns a probability to each interpretation that is a subset of the Herbrand Base of P. Moreover, given the function π_P^*, a probability function for formulas can be defined.

This formalism is interesting for the intuitive reading of its formulae that makes the writing of LPADs simpler than other formalisms. Moreover, also the semantics is simple and elegant. The formalism that is closest to LPADs is ICL: in fact, in [21] the authors show that ICL programs can be translated into LPADs and acyclic LPADs can be translated into ICL programs. Therefore, ICL programs are equivalent to a large class of LPADs. However, ICL is more suited for representing problems where we must infer causes from effects, like diagnosis or theory revision problems, while LPADs are more suited for reasoning on the effects of certain actions.

In this paper we propose the algorithm LLPAD (Learning LPADs) that learns a large subclass of LPADs. We consider a learning problem where we are given a set of interpretations together with their probabilities and a language bias and we want to find an LPAD that assigns to each input interpretation its probability according to the semantics.

LLPAD is able to learn LPADs that are sound and such that a couple of clauses sharing a disjunct have mutually exclusive bodies, i. e., bodies that are never both true in an interpretation I that has a non-zero probability.

LLPAD exploits techniques from the learning from interpretations setting: it searches first for the definite clauses that are true in all the input interpretations and are true in a non-trivial way in at least one interpretation, i. e., they have the body true in the interpretation, and then it searches for disjunctive clauses that are true in all the input interpretations, are non-trivially true in at least one interpretations and have mutually exclusive disjuncts in the head. Once the disjunctive clauses have been found, the probability of each disjunct in the head is computed.

Finally, we must decide which of the found clauses belong to the target program. To this purpose, we assign to each annotated disjunctive clause a Boolean variable that represents the presence or absence of the clause in a solution. Then, for each input interpretation, we impose a constraint over the variables of the clauses that have the body true in the interpretation. The constraint is based on the semantics of LPADs and ensures that the probability assigned to the interpretation by the final program is the one given as input for that interpretation.

The paper is organized as follows. In section 2 we provide some preliminary notions regarding LPADs together with the semantics of LPADs as given in [22]. In section 3 we discuss two properties of LPADs that are exploited by LLPAD. In section 4 we introduce the learning problem we have defined and we describe LLPAD. In section 5 we discuss related works and finally in section 6 we conclude and present directions for future work.

2 LPADs

2.1 Preliminaries

A disjunctive logic program [12] is a set of disjunctive clauses. A disjunctive clause is a formula of the form

$$h_1 \vee h_2 \vee \ldots \vee h_n \leftarrow b_1, b_2, \ldots, b_m$$

where h_i are logical atoms and b_i are logical literals. The disjunction $h_1 \vee h_2 \vee \ldots \vee h_n$ is called the *head* of the clause and the conjunction $b_1 \wedge b_2 \wedge \ldots \wedge b_m$ is the called the *body*. Let us define the two functions $head(c)$ and $body(c)$ that, given a disjunctive clause c, return respectively the head and the body of c. In some cases, we will use the functions $head(c)$ and $body(c)$ to denote the set of the atoms in the head or of the literals of the body respectively. The meaning of $head(c)$ and $body(c)$ will be clear from the context.

The Herbrand base $H_B(P)$ of a disjunctive logic program P is the set of all the atoms constructed with the predicate, constant and functor symbols appearing in P. A Herbrand interpretation is a subset of $H_B(P)$. Let us denote the set of all Herbrand interpretations by $\mathcal{I}_P$. In this paper we will consider only Herbrand interpretations and in the following we will drop the word Herbrand. A disjunctive clause c is true in an interpretation I if for all grounding substitution θ of c: $body(c)\theta \subset I \rightarrow head(c)\theta \cap I \neq \emptyset$. As was observed by [4], the truth of a clause c in an interpretation I can be tested by running the query $? - body(c), not\ head(c)$ on a database containing I. If the query succeeds c is false in I. If the query finitely fails c is true in I. A clause c θ-subsumes a clause d if and only if there exists a substitution θ such that $c\theta \subseteq d$ and we write $c \geq_\theta d$.

A Logic Program with Annotated Disjunctions consists of a set of formulas of the form

$$(h_1 : p_1) \vee (h_2 : p_2) \vee \ldots \vee (h_n : p_n) \leftarrow b_1, b_2, \ldots b_m$$

called *annotated disjunctive clauses*. In such a clause the h_i are logical atoms, the b_i are logical literals and the p_i are real numbers in the interval $[0, 1]$ such that $\sum_{i=1}^{n} p_i = 1$. For a clause c of the form above, we define $head(c)$ as the set $\{(h_i : p_i) | 1 \leq i \leq n\}$ and $body(c)$ as the set $\{b_i | 1 \leq i \leq m\}$. If $head(c)$ contains a single element $(a : 1)$ we will simply denote the head as a. The set of all ground LPAD defined over a first order alphabet is denoted by $\mathcal{P}_\mathcal{G}$.

Let us see an example of LPAD taken from [22].

$$\begin{aligned}
(heads(Coin) : 0.5) \vee (tails(Coin) : 0.5) &\leftarrow toss(Coin), \neg biased(Coin).\\
(heads(Coin) : 0.6) \vee (tails(Coin) : 0.4) &\leftarrow toss(Coin), biased(Coin).\\
(fair(Coin) : 0.9) \vee (biased(Coin) : 0.1).&\\
toss(Coin).&
\end{aligned}$$

2.2 Semantics of LPADs

The semantics of an LPAD was given in [22]. We report it here for the sake of completeness. It is given in terms of its grounding. Therefore we restrict our attention to ground LPADs, i.e., LPADs belonging to $\mathcal{P}_\mathcal{G}$. For example, the grounding of the LPAD given in the previous section is

$$\begin{aligned}(heads(coin) : 0.5) \vee (tails(coin) : 0.5) &\leftarrow toss(coin), \neg biased(coin).\\ (heads(coin) : 0.6) \vee (tails(coin) : 0.4) &\leftarrow toss(coin), biased(coin).\\ (fair(coin) : 0.9) \vee (biased(coin) : 0.1).&\\ toss(coin).&\end{aligned}$$

Each annotated disjunctive clause represents a probabilistic choice between a number of non-disjunctive clauses. By choosing a head for each clause of an LPAD we get a normal logic program called an *instance* of the LPAD. For example, the LPAD above has $2 \cdot 2 \cdot 2 \cdot 1 = 8$ possible instances one of which is

$$\begin{aligned}heads(coin) &\leftarrow toss(coin), \neg biased(coin).\\ heads(coin) &\leftarrow toss(coin), biased(coin).\\ fair(coin).&\\ toss(coin).&\end{aligned}$$

A probability is assigned to all instances by assuming independence between the choices made for each clause. Therefore, the probability of the instance above is $0.5 \cdot 0.6 \cdot 0.9 \cdot 1 = 0.27$.

An instance is identified by means of a selection function.

Definition 1 (Selection function). *Let P be a program in $\mathcal{P}_\mathcal{G}$. A* selection σ *is a function which selects one pair $(h : \alpha)$ from each rule of P, i.e. $\sigma : P \rightarrow (H_B(P) \times [0, 1])$ such that, for each r in P, $\sigma(r) \in head(r)$. For each rule r, we denote the atom h selected from this rule by $\sigma_{atom}(r)$ and the probability α selected by $\sigma_{prob}(r)$. Furthermore, we denote the set of all selections σ by $\mathcal{S}_P$.*

Let us now give the formal definition of an instance.

Definition 2 (Instance). *Let P be a program in $\mathcal{P}_\mathcal{G}$ and σ a selection in $\mathcal{S}_P$. The* instance P_σ *chosen by σ is obtained by keeping only the atom selected for r in the head of each rule $r \in P$, i.e. $P_\sigma = \{$ "$\sigma_{atom}(r) \leftarrow body(r)$"$|r \in P\}$.*

We now assign a probability to a selection function σ and therefore also to the associated program P_σ.

Definition 3 (Probability of a selection). *Let P be a program in $\mathcal{P}_\mathcal{G}$. The* probability of a selection σ *in $\mathcal{S}_P$ is the product of the the probability of the individual choices made by that selection, i.e.*

$$C_\sigma = \prod_{r \in P} \sigma_{prob}(r)$$

The instances of an LPAD P are normal logic program. Their semantics can be given by any of the semantics defined for normal logic programs (e.g. Clark's completion [2], Fitting semantics [5], stable models [6], well founded semantics [20]). In this paper we will consider only the well founded semantics, the most skeptical one. Since in LPAD the uncertainty is modeled by means of the annotated disjunctions, the instances of an LPAD should contain no uncertainty, i.e. they should have a single two-valued model. Therefore, given an instance P_σ, its semantics is given by its well founded model $WFM(P_\sigma)$ and we require that it is two-valued.

Definition 4 (Sound LPAD). *An LPAD P is called* sound *iff for each selection σ in $\mathcal{S}_P$, the well founded model $WFM(P_\sigma)$ of the program P_σ chosen by σ is two-valued.*

For example, if the LPAD is acyclic (meaning that all its instances are acyclic) then the LPAD is sound. We denote with $P_\sigma \models_{WFM} F$ the fact that the formula F is true in the well founded model of P_σ.

We now define the probability of interpretations.

Definition 5 (Probability of an interpretation). *Let P be a sound LPAD in $\mathcal{P}_\mathcal{G}$. For each of its interpretations I in $\mathcal{I}_P$, the probability $\pi^*_P(I)$ assigned by P to I is the sum of the probabilities of all selections which lead to I, i.e. with $S(I)$ being the set of all selection σ for which $WFM(P_\sigma) = I$:*

$$\pi^*_P(I) = \sum_{\sigma \in S(I)} C_\sigma$$

For example, consider the interpretation $\{toss(coin), fair(coin), heads(coin)\}$. This interpretation is the well founded model of two instance of the example LPAD, one is the instance shown above and the other is the instance:

$$\begin{aligned} heads(coin) &\leftarrow toss(coin), \neg biased(coin). \\ tails(coin) &\leftarrow toss(coin), biased(coin). \\ fair(coin). & \\ toss(coin). & \end{aligned}$$

The probability of this instance is $0.5 \cdot 0.4 \cdot 0.9 \cdot 1 = 0.18$. Therefore, the probability of the interpretation above is $0.5 \cdot 0.4 \cdot 0.9 \cdot 1 + 0.5 \cdot 0.6 \cdot 0.9 \cdot 1 = 0.5 \cdot (0.4 + 0.6) \cdot 0.9 \cdot 1 = 0.45$.

3 Properties of LPADs

We now give a definition and two theorems that will be useful in the following.

The first theorem states that, under certain conditions, the probabilities of the head disjuncts of a rule can be computed from the probabilities of the interpretations. In particular, the probability of a disjunct h_i is given by the sum

of the probabilities of interpretations where the body of the clause and h_i are true divided by the sum of the probabilities of interpretations where the body is true.

The second theorem states that, given an interpretation, under certain conditions, all the selection σ in the set $S(I)$ agree on all the rules with the body true and that the probability of I can be computed by multiplying the probabilities of the head disjuncts selected by a $\sigma \in S(I)$ for all the clauses with the body true.

Definition 6 (Mutually exclusive bodies). *Clauses $H_1 \leftarrow B_1$ and $H_2 \leftarrow B_2$ have mutually exclusive bodies over a set of interpretations J if, $\forall I \in J$, B_1 and B_2 are not both true in I.*

Theorem 1. *Consider a sound LPAD P and a clause $c \in P$ of the form*

$$c = h_1 : p_1 \vee h_2 : p_2 \vee \ldots h_m : p_m \leftarrow B.$$

Suppose you are given the function π_P^* and suppose that all the couples of clauses of P that share an atom in the head have mutually exclusive bodies over the set of interpretations $J = \{I | \pi_\sigma^*(I) > 0\}$. The probabilities p_i can be computed with the following formula:

$$p_i = \frac{\sum_{I \in \mathcal{I}_P, I \models B, h_i} \pi_P^*(I)}{\sum_{I \in \mathcal{I}_P, I \models B} \pi_P^*(I)}$$

Proof. Let us first expand the numerator:

$$\sum_{I \in \mathcal{I}_P, I \models B, h_i} \pi_P^*(I) = \sum_{I \in J, I \models B, h_i} \sum_{\sigma \in S(I)} \prod_{r \in P} \sigma_{prob}(r)$$

A selection σ such that $WFM(P_\sigma) = I$ for an I such that $I \models B, h_i$ is a selection such that $P_\sigma \models_{WFM} B, h_i$. Therefore the above expression can be written as

$$\sum_{\sigma \in T} \prod_{r \in P} \sigma_{prob}(r)$$

where $T = \{\sigma | P_\sigma \models_{WFM} B, h_i\}$. Since clause c has a mutually exclusive body over the set of interpretations J with all the other clauses of P that contain h_i in the head, the truth of h_i in P_σ can be obtained only if $\sigma(c) = (h_i : p_i)$ for all $\sigma \in T$, therefore the numerator becomes

$$\sum_{\sigma \in T} p_i \prod_{r \in P \setminus \{c\}} \sigma_{prob}(r) = p_i \sum_{\sigma \in T} \prod_{r \in P \setminus \{c\}} \sigma_{prob}(r)$$

Let us expand the denominator in a similar way

$$\sum_{I \in \mathcal{I}_P, I \models B} \pi_P^*(I) = \sum_{\sigma \in Q} \prod_{r \in P} \sigma_{prob}(r)$$

where $Q = \{\sigma | P_\sigma \models_{WFM} B\}$. Clause c expresses the fact that, if B is true, then either h_1, h_2, ... or h_m is true, i.e., these cases are exhaustive. Moreover, they are also exhaustive. Therefore we can write Q in the following way:

$$Q = \{\sigma | P_\sigma \models_{WFM} B, h_1\} \cup \{\sigma | P_\sigma \models_{WFM} B, h_2\} \cup \ldots \cup \{\sigma | P_\sigma \models_{WFM} B, h_m\}$$

Let $Q_j = \{\sigma | P_\sigma \models_{WFM} B, h_j\}$, then $Q_j \cap Q_k = \emptyset$ for all $j, k = 1, \ldots m, j \neq k$.

Since clause c has a mutually exclusive body over the set of interpretations J with all the other clauses of P, the truth of h_j in P_σ can be obtained only if $\sigma(c) = h_j : p_j$ for all $\sigma \in Q_j$, therefore the denominator becomes

$$\sum_{j=1}^{m} p_j \sum_{\sigma \in Q_j} \prod_{r \in P \setminus \{c\}} \sigma_{prob}(r)$$

Given a selection σ^T in T, consider a selection σ^{Q_j} that differs from σ^T only over clause c, i.e., $\sigma^T(c) = (h_i : p_i)$ while $\sigma^{Q_j}(c) = (h_j : p_j)$. From $P_{\sigma^T} \models_{WFM} B$ follows that $P_{\sigma^{Q_j}} \models_{WFM} B$ because B can not depend on the truth of literal h_i since otherwise there would be a loop and B would not be true in the well-founded model of P_{σ^T}. From $P_{\sigma^{Q_j}} \models_{WFM} B$ follows that $P_{\sigma^{Q_j}} \models_{WFM} B, h_j$ because B can not depend on $\neg h_j$ since otherwise there would be a loop through negation and the LPAD P would not be sound, in contradiction with the hypothesis. Therefore σ^{Q_j} is in Q_j. The same reasoning can be applied in the opposite direction. As a consequence

$$\sum_{\sigma \in T} \prod_{r \in P \setminus \{c\}} \sigma_{prob}(r) = \sum_{\sigma \in Q_j} \prod_{r \in P \setminus \{c\}} \sigma_{prob}(r)$$

for all $j = 1, \ldots, m$. Thus, the fraction becomes

$$\frac{p_i \sum_{\sigma \in T} \prod_{r \in P \setminus \{c\}} \sigma_{prob}(r)}{\left(\sum_{j=1}^{m} p_j\right) \sum_{\sigma \in T} \prod_{r \in P \setminus \{c\}} \sigma_{prob}(r)} = p_i$$

□

Theorem 2. *Consider an interpretation I and an LPAD P such that all the couples of clauses that share an atom in the head have mutually exclusive bodies with respect to the set of interpretations $\{I\}$. Then all the selection $\sigma \in S(I)$ agree on the clauses with body true in I and*

$$\pi_P^*(I) = \prod_{r \in P, I \models body(r)} \sigma_{prob}(r)$$

where σ is any element of $S(I)$.

Proof. We prove the theorem by induction on the number n of clauses with the body false in I.

Case $n = 0$. For each atom $A \in I$, there is only one clause c that has it in the head for the assumption of mutual exclusion. Therefore, for A to be in I, σ

must select atom A for clause c. Moreover, all the clauses have the body true, therefore for each clause one atom in the head must be in I. Therefore there is a single σ in $S(I)$ and the theorem holds.

We assume that the theorem holds for a program P^{n-1} with $n-1$ clauses with the body false in I. We have to prove that the theorem holds for a program P^n obtained from P^{n-1} by adding a clause r_n with the body false. Suppose that the r_n is

$$h_1 : p_1 \vee h_2 : p_2 \vee \ldots \vee h_m : p_m \leftarrow B$$

Let $S_{n-1}(I)$ ($S_n(I)$) be the set of all selections σ such that $WFM(P^{n-1}_\sigma) = I$ ($WFM(P^n_\sigma = I)$). Moreover, let $S_{n-1}(I)$ be $\{\sigma^1, \sigma^2, \ldots, \sigma^k\}$.

Since B is false in I, any head disjunct in r_n can be selected and r_n will be true anyway in I. Therefore, for each $\sigma^i \in S_{n-1}(I)$, there are m $\sigma^{i,j}$ in $S_n(I)$. Each $\sigma^{i,j}$ agrees with σ^i on all the rules of P^{n-1}. $\sigma^{i,j}$ extends σ^i by selecting the jth disjunct in clause r_n, i.e $\sigma^{i,j}(r_n) = (h_j : p_j)$. We can write:

$$\begin{aligned}
\pi^*_{P^n}(I) = & \sum_{\sigma \in S_n(I)} \prod_{r \in P^n} \sigma_{prob}(r) = \\
& = \sum_{\sigma \in S_n(I)} \prod_{r \in P^{n-1}} \sigma_{prob}(r)\sigma_{prob}(r_n) = \\
& = \prod_{r \in P^{n-1}} \sigma^{1,1}_{prob}(r)p_1 + \prod_{r \in P^{n-1}} \sigma^{1,2}_{prob}(r)p_2 + \ldots + \prod_{r \in P^{n-1}} \sigma^{1,m}_{prob}(r)p_m + \\
& + \ldots \\
& + \prod_{r \in P^{n-1}} \sigma^{k,1}_{prob}(r)p_1 + \prod_{r \in P^{n-1}} \sigma^{k,2}_{prob}(r)p_2 + \ldots + \prod_{r \in P^{n-1}} \sigma^{k,m}_{prob}(r)p_m = \\
& = p_1 \left(\prod_{r \in P^{n-1}} \sigma^{1,1}_{prob}(r) + \prod_{r \in P^{n-1}} \sigma^{2,1}_{prob}(r) + \ldots + \prod_{r \in P^{n-1}} \sigma^{k,1}_{prob}(r) \right) + \\
& + \ldots \\
& + p_m \left(\prod_{r \in P^{n-1}} \sigma^{1,m}_{prob}(r) + \prod_{r \in P^{n-1}} \sigma^{2,m}_{prob}(r) + \ldots + \prod_{r \in P^{n-1}} \sigma^{k,m}_{prob}(r) \right)
\end{aligned}$$

Since $\sigma^{i,j}$ extends σ^i only on clause r_n, then $\prod_{r \in P^{n-1}} \sigma^{i,j}_{prob}(r) = \prod_{r \in P^{n-1}} \sigma^i_{prob}(r)$. Therefore

$$\begin{aligned}
\pi^*_{p^n}(I) = & (p_1 + p_2 + \ldots + p_m) \times \\
& \times \left(\prod_{r \in P^{n-1}} \sigma^1_{prob}(r) + \prod_{r \in P^{n-1}} \sigma^2_{prob}(r) + \ldots + \prod_{r \in P^{n-1}} \sigma^k_{prob}(r) \right) = \\
& = \sum_{\sigma \in S_{n-1}(I)} \prod_{r \in P^{n-1}} \sigma_{prob}(r)
\end{aligned}$$

which, for the hypothesis for $n-1$, becomes

$$\prod_{r \in P^n, I \models body(r)} \sigma_{prob}(r)$$

□

The hypothesis of mutual exclusion of the bodies is fundamental for this theorem to hold. In fact, consider the following example:

$$P_1 = a : a_1 \vee b : b_1.$$
$$a : a_2 \vee b : b_2.$$

Then $\pi^*_{P_1}(\{a, b\}) = a_1 b_2 + a_2 b_1$.

One may think that is enough to have mutually exclusive bodies only when two clauses have the same disjunct in the head. But this is not true as the following example shows:

$$P_2 = a : a_1 \vee b : b_1 \vee c : c_1.$$
$$a : a_2 \vee c : c_2 \vee d : d_2.$$
$$a : a_3 \vee b : b_3 \vee d : d_3.$$

Then $\pi^*_{P_2}(\{a, b, c\}) = a_1 c_2 b_3 + b_1 c_2 a_3 + c_1 a_2 b_3$.

4 Learning LPADs

We consider a learning problem of the following form:

Given:

- a set E of examples that are couples $(I, Pr(I))$ where I is an interpretation and $Pr(I)$ is its associated probability
- a space of possible LPAD S (described by a language bias LB)

Find:

- an LPAD $P \in S$ such that $\forall (I, Pr(I)) \in E \;\; \pi^*_P(I) = Pr(I)$

Instead of a set of couples $(I, Pr(I))$, the input of the learning problem can be a multiset E' of interpretations. From this case we can obtain a learning problem of the form above by computing a probability for each interpretation in E'. The probability can be computed in the obvious way, by dividing the number of occurrences of the interpretation by the total number of interpretations in E'.

Before discussing the learning algorithm, let us first provide some preliminaries.

Definition 7 (Clause non-trivially true in an interpretation (adapted from [4])). *A clause c is* non-trivially true *in an interpretation I if c is true in I and there exist at least one grounding substitution θ of c such that both $body(c)\theta$ and $head(c)\theta$ are true in I.*

Definition 8 (Refinement of a body). *The* refinement of a body B *of a clause is a body B' such that $B \geq_\theta B'$ and there does not exist a body B'' such that $B \geq_\theta B''$ and $B'' \geq_\theta B'$.*

Refining the body of a clause c can make c non-trivially true in less interpretations.

Definition 9 (Refinement of a head). *The* refinement of a head H *of a clause is a head* H' *such that* $H' \geq_\theta H$ *and there does not exist a head* H'' *such that* $H' \geq_\theta H''$ *and* $H'' \geq_\theta H$*.*

Definition 10 (Mutually exclusive disjuncts). *The disjuncts in the head of a clause are* mutually exclusive with respect to a set of interpretations J *if there is no interpretation* $I \in J$ *such that two or more disjuncts are true in* I*.*

Let us suppose that the language bias LB is given in the form of set of couples (ALH, ALB) where ALH is the set of literals allowed in the head and ALB is the set of literals allowed in the body.

The algorithm (see figure 1) proceeds in three stages. In the first, it searches for all the definite clauses allowed by the language bias

- that are true in all interpretations,
- that are non-trivially true in at least one interpretation.

The reason for searching separately for definite clauses will be explained in the following.

In the second stage, the algorithm searches for all the non-annotated disjunctive clauses allowed by the language bias

- that are true in all interpretations,
- that are non-trivially true in at least one interpretation,
- whose disjuncts in the head are mutually exclusive.

When it finds one such clause, it annotates the head disjuncts with a probability.

In the third stage, a constraint satisfaction problem is solved in order to find subsets of the annotated disjunctive clauses that form programs that assign to each interpretation the associated probability.

The search for clauses in the first stage is repeated for each couple (ALH, ALB) in LB. For each literal L in ALH, a search is started from clause $L \leftarrow$ true (function Search_Definite, not shown for brevity). The body is refined until it is true in 0 interpretations, in which case the search stops, or in the case where the head is true in all the interpretations in which the body is true, in which case the clause is returned.

The search for clauses in the second stage is repeated for each couple (ALH, ALB) in LB. The search is performed by first searching breadth-first for bodies that are true in at least one interpretation (function Search_Body, see figure 2). Every time a body is true in at least one interpretation, the algorithm searches breadth-first for all the heads that are true in the interpretations where the body is true (set EB) and whose disjuncts are mutually exclusive with respect to EB (function Search_Head, see figure 3).

Search_Body is initially called with a body equal to true. Since such a body is true in all interpretations, the function Search_Head is called with an initial head

$Head$ containing all the literals allowed by the bias (ALH). The clauses returned by Search_Head are then added to the current set of clauses and Search_Body is called recursively over all the refinements of true. This is done because different bodies may have different heads.

In Search_Head the head is tested to find out the interpretations of EB where the head is true. If the head is not true in all the interpretations of EB, the search is stopped and the empty set of clauses is returned because there is no way to refine the head in order to make the clause true. Instead, if the head is true in all the interpretations of EB, $Head$ is tested to see whether the disjuncts are mutually exclusive. If so, the head disjuncts that are false in all the interpretations are removed, the probabilities of the remaining head disjuncts are computed and the clause is returned by Search_Head. If the disjuncts are not mutually exclusive, all the refinements of $Head$ are considered and Search_Head is called recursively on each of them.

The probabilities of the disjuncts in the head are computed according to theorem 1: the probability of a disjunct is given by the sum of the probabilities of the interpretations in EB where the disjunct is true divided by the sum of probabilities of the interpretations in EB. The function Compute_Probabilities takes a disjunction of atoms and returns an annotated disjunction.

Example 1. Consider the coin problem presented in section 2.1. The set of couples $(I, Pr(I))$ is:

$$\begin{array}{ll} I_1 = \{heads(coin), toss(coin), fair(coin)\} & Pr(I_1) = 0.45 \\ I_2 = \{tails(coin), toss(coin), fair(coin)\} & Pr(I_2) = 0.45 \\ I_3 = \{heads(coin), toss(coin), biased(coin)\} & Pr(I_3) = 0.06 \\ I_4 = \{tails(coin), toss(coin), biased(coin)\} & Pr(I_4) = 0.04 \end{array}$$

Given the above set E and the language bias $LB = \{(\{heads(coin), tails(coin), toss(coin), biased(coin), fair(coin)\}, \{toss(coin), biased(coin), fair(coin)\})\}$, the algorithm generates the following definite clause:

$$d_1 = toss.$$

and the following set of annotated disjunctive clauses SC:

$$\begin{array}{l} c_1 = biased(coin) : 0.1 \vee fair(coin) : 0.9. \\ c_2 = heads(coin) : 0.51 \vee tails(coin) : 0.49. \\ c_3 = biased(coin) : 0.1 \vee fair(coin) : 0.9 \leftarrow toss(coin). \\ c_4 = heads(coin) : 0.51 \vee tails(coin) : 0.49 \leftarrow toss(coin). \\ c_5 = heads(coin) : 0.6 \vee tails(coin) : 0.4 \leftarrow toss(coin), biased(coin). \\ c_6 = heads(coin) : 0.5 \vee tails(coin) : 0.5 \leftarrow toss(coin), fair(coin). \\ c_7 = heads(coin) : 0.6 \vee tails(coin) : 0.4 \leftarrow biased(coin). \\ c_8 = heads(coin) : 0.5 \vee tails(coin) : 0.5 \leftarrow fair(coin). \end{array}$$

In the third stage, we have to partition the found disjunctive clauses in subsets that are solutions of the learning problem. This is done by assigning to

function LLPAD(
inputs : E : set of couples $(I, Pr(I))$,
A language bias LB in the form of a set of couples (ALH, ALB),
where ALH is a list of literals allowed in the head,
and ALB is a list of literals allowed in the body)
returns : SP : a set of learned LPAD

$SD := \emptyset$
$ES := \{(\emptyset, I) | (I, Pr(I)) \in E\}$
$ES' := \{(\emptyset, I, Pr(I)) | (I, Pr(I)) \in E\}$
for all couples (ALH, ALB) do
for all literals L in ALH
$SD := SD \cup$Search_Definite$(ALB, L \leftarrow$ true,$ES)$
end for
end for
$SC := \emptyset$
for all couples (ALH, ALB) do
$Body :=$true
$SC := SC \cup$Search_Body$(ALH, ALB, Body, ES')$
end for
for all $c_i \in SC$
assert the constraint $x_i \in [0, 1]$
end for
for all couples (c_i, c_j) of clauses of SC
if c_i and c_j do not have mutually exclusive bodies over
the set of interpretations E then
assert the constraint $x_i + x_l \leq 1$
end if
end for
for all couples $(I, Pr(I)) \in E$
assert the following constraint
$\sum_{c_i \in SC(I)} x_i \log p_i = \log Pr(I)$
where $SC(I)$ is the set of clauses of SC whose body is
true in I and p_i is the probability associated with
the head disjunct of c_i that is true in I
end for
$SP := \emptyset$
for all solutions of the resulting CSP
let P be the LPAD that contains all the clauses c_i for which
$x_i = 1$
$SP := SP \cup \{P \cup SD\}$
end for
return SP

Fig. 1. Function LLPAD

```
function Search_Body(
    inputs : ALH : set of literals allowed in the head,
        ALB : set of literals allowed in the body,
        Body : current body,
        E: set of triples (Θ, I, Pr(I)))
    returns : SC : a set of disjunctive clauses

SC := ∅
Test Body over E
let EB be a set containing elements of the form (Θ, I, Pr(I)) where I is
    an interpretation where Body is true and Θ is the set of substitutions with
    which Body is true in I
if Body is true in 0 interpretations then
    return ∅
else
    Head := ALH
    SC := SC∪Search_Head(ALH, Head, Body, EB)
    for all refinements Body′ of Body
        SC := SC∪Search_Body(ALH, ALB, Body′, EB)
    end for
end if
return SC
```

Fig. 2. Function Search_Body

each clause c_i found in the second stage a variable x_i that is 0 if the clause is absent from a solution P and is 1 if the clause is present in a solution. Then we must ensure that the couples of clauses that share a literal in the head have mutually exclusive bodies over the set of interpretations E. This is achieved by testing, for each couple of clauses, if they share a literal in the head and, if so, if the intersections of the two sets of interpretations where their body is true is non-empty. In this case, we must ensure that the two clauses are not both included in a solution. To this purpose, we assert the constraint $x_i + x_j \leq 1$ for all such couples of clauses (c_i, c_j). Finally we must ensure that P assigns the correct probability to each interpretation. For each interpretation we thus have the constraint:

$$\prod_{c_i \in SC(I)} p_i^{x_i} = Pr(I)$$

where $SC(I)$ is the subset of SC of all the disjunctive clauses whose body is true in I and p_i is the probability of the single head of c_i that is true in I. This constraint is based on theorem 2.

The definite clauses are not considered in the constraints because they would contribute only with a factor 1^{x_j} that has no effect on the constraint for any value of x_j. Therefore, for each assignment of the other variables, x_j can be either 0 or 1. This is the reason why we have learned the definite clauses separately.

```
function Search_Head(
    inputs : ALH : set of literals allowed in the head,
        Head : current head,
        Body : current body,
        E: set of triples (Θ, I, Pr(I)))
    returns : SC : a set of disjunctive clauses

SC := ∅
if Head and Body share one or more literals then
    % the clause is a tautology
    for all refinements Head' of Head
        SC := SC∪Search_Head(ALH, Head', Body, E)
    end for
    return SC
else
    test Head over E
    if, for all (Θ, I, Pr(I)) ∈ E, Head is true in I with every substitution θ ∈ Θ
        then
        if the disjuncts are mutually exclusive then
            % a good clause has been found
            obtain Head' by removing from Head all the literals that are
                false in all the interpretations
            Head'' :=Compute_Probabilities(Head', Body, EB)
            return {Head'' ← Body}
        else
            % Head has to be refined
            for all refinements Head' of Head
                SC := SC∪Search_Head(ALH, Head', Body, E)
            end for
            return SC
        end if
    else
        % Head is false in some interpretations
        return ∅
    end if
end if
```

Fig. 3. Function Search_Head

If we take the logarithm of both members we get the following linear constraint:

$$\sum_{c_i \in SC(I)} x_i \log p_i = \log Pr(I)$$

We can thus find the solutions of the learning problem by solving the above constraint satisfaction problem.

Even if the variables have finite domains, it is not possible to use a CLP(FD) solver because the coefficients of the linear equations are irrational. Therefore we solve a relaxation of the above problem where the variables are real numbers belonging to the interval $[0, 1]$.

Example 2 (Continuation of example 1). In the third stage, we use the CLP(R) solver [8] of Sicstus Prolog and we obtain the following answer:

$$x_2 = 0, x_4 = 0, x_1 = 1 - x_3, x_5 = 1 - x_7, x_6 = 1 - x_8$$

meaning that c_2 and c_4 are absent, if c_1 is present c_3 must be absent and viceversa, if c_5 is present c_7 must be absent and viceversa and if c_6 is present c_8 must be absent and viceversa. By labeling the variables in all possible ways we get eight solutions, which is exactly the number that can be obtained by considering the three double choices. The original program is among the eight solutions.

We now show that the algorithm is correct and complete.

The algorithm is correct because all the disjunctive clauses that are found are true in all the interpretations, are non-trivially true in at at least one interpretation and have mutually exclusive disjuncts in the head. The probabilities of the head literals are correct because of theorem 1 and a solution of the constraint satisfaction problem is a program for which clauses which share a literal have mutually exclusive bodies for all the interpretations in E and that assigns to each interpretation in E the correct probability because of theorem 2.

The algorithm is also complete, i.e. if there is an LPAD in the language bias that satisfies the above conditions, then LLPAD will find it, since it searches the space of possible clauses in a breadth first way.

The algorithm can be made heuristic by relaxing some of the constraints imposed: for example, we can require that the clauses must be non-trivially true in more than one interpretation. In this way, we can prune a clause as soon as its bodies is true in less than the minimum number of interpretations. However, in this case the constraint satisfaction problem must not be solved in an exact way but rather in an approximate way.

5 Related Works

To the best of our knowledge, this paper is the first published attempt to learn LPADs. Therefore, the only works related to ours are those that deal with the learning of other forms of probabilistic models.

In [21] the authors compare LPAD with Bayesian Logic Programs (BLP) [9, 10]. They show that every BLP can be expressed as an LPAD with a semantics that matches that of the BLP. Moreover, they also show that a large subset of LPADs can be translated into BLPs in a way that preserves the semantics of the LPADs. Such a subset is the set of all ground LPADs that are acyclic and where each ground atom depends only on a finite number of atoms. Therefore, the techniques developed in [11] for learning BLPs can be used for learning this class of LPADs as well. However, the technique proposed in this paper are not based on a greedy algorithm as that in [11], therefore it should be less likely to incur in a local maxima.

In [7] the authors propose a formalism called Probabilistic Relational Models (PRM) that extends the formalism of Bayesian networks to be able to model domains that are described by a multi-table relational database. Each attribute of a table is considered as a random variable and its set of parents can contain other attributes of the same table or attributes of other tables connected to the attribute table by foreign key connections. The relationship between PRM and LPADs is not clear. For sure they have a non empty intersection: the PRM that do not contain attributes that depend on aggregate functions of attributes of other tables can be expressed as LPADs. Moreover, LPADs are not a subset of PRM since they can express partial knowledge regarding the dependence of an attribute from other attributes, in the sense that with LPADs it is possible to specify only a part of a conditional probability table. Therefore the learning techniques developed in [7] can not be used in substitution of the techniques proposed in this paper.

[1] and [18] propose two logic languages for encoding in a synthetic way complex Bayesian networks. They are both very much related to PRM. Differently from PRM they offer some of the advantages of logic programming: the combination of function symbols and recursion, non-determinism and the applicability of ILP techniques for learning. LPADs offer as well this features. Moreover, with LPADs partial decision tables can be encoded.

Stochastic Logic Programs (SLPs) [3, 13] are another formalism integrating logic and probability. In [21] the authors have shown that a SLP can be translated in LPAD, while whether the opposite is possible is not known yet. Therefore, it is not clear at the moment whether the techniques used for learning SLPs can be used for learning LPADs.

PRISM [19] is a logic programming language in which a program is composed of a set of facts and a set of rules such that no atom in the set of facts appears in the head of a rule. In a PRISM program, each atom is seen as a random variable taking value true or false. A probability distribution for the atoms appearing in the head of rules is inferred from a given probability distribution for the set of facts. PRISM differs from LPADs because PRISM assigns a probability distribution to ground facts while LPADs assign a probability distribution to the literals in the head of rules. PRISM programs resemble programs of ICL, in the sense that PRISM facts can be seen as ICL abducibles. In [19] the author also proposes an algorithm for learning the parameters of the probability distribution

of facts from a given probability distribution for the observable atoms (atoms in the head of rules). However, no algorithm for learning the rules of a PRISM program has been defined. Inferring the parameters of the distribution is performed in LLPAD analytically by means of theorem 1 rather than by means of the EM algorithm as in PRISM.

We have already observed that a large class of LPADs is equivalent to ICL, namely the class of acyclic LPADs. Another formalism very related to ICL and LPADs is Probabilistic Disjunctive Logic Programs (PDLP) [16]. PDLP are not required to be acyclic and therefore all LPADs can be translated into PDLP and all PDLP can be translated into an LPAD. However, it remains to be checked whether the transformation preserves the semantics. Moreover, in [16] the author does not propose an algorithm for learning PDLP.

6 Conclusion and Future Works

We have defined a problem of learning LPADs and an algorithm that is able to solve it by means of learning from interpretations and constraint solving techniques. The algorithm has been implemented in Sicstus Prolog 3.9.0 and is available on request from the author.

In the future we plan to adopt more sofisticated language bias, for example the $\mathcal{D}$LAB formalism or the mode predicates of Progol and Aleph. Moreover, we plan to use extra information in order to be able to select among the set of learned clauses. An example of this extra information is negative interpretations, i.e., interpretations where the clauses have to be false.

7 Acknowledgements

This work was partially funded by the IST programme of the EC, FET under the IST-2001-32530 SOCS project, within the Global Computing proactive initiative and by the Ministero dell'Istruzione, della Ricerca e dell'Università under the COFIN2003 project "La gestione e la negoziazione automatica dei diritti sulle opere dell'ingegno digitali: aspetti giuridici e informatici". The author would like to thank Evelina Lamma and Luìs Moniz Pereira for many interesting discussions on the topic of this paper.

References

1. Hendrik Blockeel. Prolog for first-order bayesian networks: A meta-intepreter approach. In *Multi-Relational Data Mining (MRDM03)*, 2003.
2. K. L. Clark. Negation as failure. In *Logic and Databases*. Plenum Press, 1978.
3. James Cussens. Stochastic logic programs: Sampling, inference and applications. In *Sixteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-2000)*, pages 115–122, San Francisco, CA, 2000. Morgan Kaufmann.
4. L. De Raedt and L. Dehaspe. Clausal discovery. *Machine Learning*, 26(2–3):99–146, 1997.

5. M. Fitting. A kripke-kleene semantics for logic programs. *Journal of Logic Programming*, 2(4):295–312, 1985.
6. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. Kowalski and K. A. Bowen, editors, *Proceedings of the 5th Int. Conf. on Logic Programming*, pages 1070–1080. MIT Press, 1988.
7. L. Getoor, N. Friedman, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In Saso Dzeroski and Nada Lavrac, editors, *Relational Data Mining*. Springer-Verlag, Berlin, 2001.
8. C. Holzbaur. OFAI clp(q,r) manual, edition 1.3.3. Technical Report TR-95-09, Austrian Research Institute for Artificial Intelligence, Vienna, 1995.
9. K. Kersting and L. De Raedt. Bayesian logic programs. In *Work-in-Progress Reports of the Tenth International Conference on Inductive Logic Programming (ILP2000), London, UK*, 2000.
10. K. Kersting and L. De Raedt. Bayesian logic programs. Technical Report 151, Institute for Computer Science, University of Freiburg, Freiburg, Germany, April 2001.
11. K. Kersting and L. De Raedt. Towards combining inductive logic programming and bayesian networks. In C. Rouveirol and M. Sebag, editors, *Eleventh International Conference on Inductive Logic Programming (ILP-2001), Strasbourg, France, September 2001*, number 2157 in LNAI. Springer-Verlag, 2001.
12. J. Lobo, J. Minker, and A. Rajasekar. *Foundations of Disjunctive Logic Programming*. MIT Press, Cambridge, Massachusetts, 1992.
13. S. H. Muggleton. Learning stochastic logic programs. *Electronic Transactions in Artificial Intelligence*, 4(041), 2000.
14. R. T. Ng and V. S. Subrahmanian. Probabilistic logic programming. *Information and Computation*, 101(2):150–201, 1992.
15. L. Ngo and P. Haddaway. Answering queries from context-sensitive probabilistic knowledge bases. *Theoretical Computer Science*, 171(1–2):147–177, 1997.
16. Liem Ngo. Probabilistic disjunctive logic programming. In *Proceedings of the 12th Annual Conference on Uncertainty in Artificial Intelligence (UAI-96)*, pages 397–404, San Francisco, CA, 1996. Morgan Kaufmann Publishers.
17. D. Poole. The Independent Choice Logic for modelling multiple agents under uncertainty. *Artificial Intelligence*, 94(1–2):7–56, 1997.
18. V. Santos Costa, D. Page, M. Qazi, and J. Cussens. Clp($\mathcal{BN}$): Constraint logic programming for probabilistic knowledge. In *Uncertainty in Artificial Intelligence (UAI03)*, 2003.
19. T. Sato. A statistical learning method for logic programs with distribution semantics. In *12th International Conference on Logic Programming (ICLP95)*, pages 715–729, 1995.
20. 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.
21. J. Vennekens and S. Verbaeten. Logic programs with annotated disjunctions. Technical Report CW386, K. U. Leuven, 2003. http://www.cs.kuleuven.ac.be/~joost/techrep.ps.
22. J. Vennekens, S. Verbaeten, and M. Bruynooghe. Logic programs with annotated disjunctions. In *The 20th International Conference on Logic Programming (ICLP04)*, 2004.