Learning Ground ProbLog Programs from Interpretations

Fabrizio Riguzzi

ENDIF, Università di Ferrara, Via Saragat 1, 44100 Ferrara, Italy
fabrizio.riguzzi@unife.it

Abstract. The relations between ProbLog and Logic Programs with Annotated Disjunctions imply that Boolean Bayesian networks can be represented as ground ProbLog programs and acyclic ground ProbLog programs can be represented as Boolean Bayesian networks. This provides a way of learning ground acyclic ProbLog programs from interpretations: first the interpretations are represented in tabular form, then a Bayesian network learning algorithm is applied and the learned network is translated into a ground ProbLog program. The program is then further analyzed in order to identify noisy or relations in it. The paper proposes an algorithm for such identification and presents an experimental analysis of its computational complexity.

Keywords: Probabilistic Logical Models, ProbLog, LPAD, Noisy Or.

1 Introduction

ProbLog [1] is a recent formalism that combines logic and probability. It is interesting for the simplicity of its semantics and for the availability of an efficient top-down interpreter. Logic Programs with Annotated Disjunctions (LPADs) [2] is another formalism for integrating probability and logic in a clear and elegant way.

In this paper, the relations between ProbLog and LPADs are investigated. We show that ground ProbLog programs can be represented as LPADs in a way that preserves the semantics. This allow us to apply results obtained for LPADs to ProbLog programs. In particular, we show that a Bayesian network with binary variables can be represented as a ground ProbLog program and that a ground acyclic [3] ProbLog program can be translated into a Boolean Bayesian network. Representing a Bayesian network as a ground ProbLog program has the advantage of a more compact representation in the case in which noisy or relations are present.

Ground ProbLog programs that encode a Bayesian network take a special form that we call Bayesian in which all the bodies of the rules for an atom contain the same set of atoms and all the possible combinations of signs of literals in the bodies are present. We present a method for transforming a general

ground ProbLog program into a program in Bayesian form. The method involves applying the noisy or law to obtain the probability of clauses.

The possibility of representing a Boolean Bayesian network with a ground ProbLog program provides a method for learning it from interpretations: we first transform the input interpretations into tabular form, then we apply a Bayesian network learning algorithm and we translate the learned network into a ground ProbLog program in Bayesian form. In order to obtain general ground acyclic ProbLog programs, we propose an algorithm that identifies the noisy or relations in the program.

The paper is organized as follows. In section 2 we introduce ProbLog and LPADs. In section 3 we present some properties of ground ProbLog programs. Section 4 discusses the learning problem and the algorithm. Section 5 presents experiments and section 6 discusses related works. Finally, section 7 concludes the paper.

2 Preliminaries

A ProbLog program [1] P is a finite set of clauses of the form

$$\alpha : h \leftarrow b_1, \ldots, b_m \tag{1}$$

where α is a real number between 0 and 1, h and $b_1, \ldots, b_m$ are atoms. We will call $H_B(P)$ the Herbrand base of P.

In this paper we consider an extension of the original ProbLog language where $b_1, \ldots, b_m$ are literals.

The semantics of the extended ProbLog is defined in terms of instances: an instance is a normal logic program obtained by selecting a subset of the clauses. Its probability is given by the product of the α factor for all the clauses that are included in the instance and of $1 - \alpha$ for all the clauses not included. The probability $\pi^P_{PB}(\phi)$ of a query ϕ according to program P is given by the sum of the probabilities of the instances that have the query as a consequence according to a chosen semantics, e.g. Clark's completion [4], stable models [5] or well founded [6]. In this paper we consider only the well founded semantics because it is the one used by LPADs.

A Logic Program with Annotated Disjunctions (LPAD) P [2] consists of a set of formulas of the form $h_1 : \alpha_1 \vee h_2 : \alpha_2 \vee \ldots \vee h_n : \alpha_n \leftarrow b_1, b_2, \ldots b_m$ called *annotated disjunctive clauses*. In such a clause the h_i are logical atoms, the b_i are logical literals and the α_i are real numbers in the interval $[0, 1]$ such that $\sum_{i=1}^{n} \alpha_i = 1$.

The semantics of LPADs is given as well in terms of instances: an instance is a ground normal program obtained by selecting for each clause of the grounding of P one of the heads. The probability of the instance is given by the product of the probabilities associated with the heads selected. The probability $\pi^P_{LP}(\phi)$ of a formula ϕ according to program P is given by the sum of the probabilities of the instances that have the formula as a consequence according to the well founded semantics.

3 Properties of ProbLog

A ground ProbLog program P can be syntactically transformed into an LPAD P' by substituting each clause of the form (1) with the LPAD clause

$$h : \alpha \vee none : (1 - \alpha) \leftarrow b_1, \ldots, b_m$$

where *none* is a special atom that does not appear in the body of any clause. This is similar to the way in which a CP-logic program [7] can be transformed into an LPAD.

Theorem 1. *Given a ground ProbLog program P and a query ϕ, $\pi^P_{PB}(\phi) = \pi^{P'}_{LP}(\phi)$.*

Proof. (Skecth) There is a one to one correspondence between instances of P and instances of P': the clauses excluded from an instance of P are present in the corresponding instance of P' with *none* in the head. A query (that does not involve the special atom *none*) is true in an instance of P if and only if it is true in the corresponding instance of P'.

In fact, for well-founded models, it can be shown that the sequence of interpretations I_α for an instance of P are equal to $I'_\alpha \setminus \{none\}$ where I'_α is the sequence of interpretations for the corresponding instance of P'.

Therefore a ground program defines the same probability distribution when interpreted as a ProbLog program and as an LPAD. For non ground programs the semantics differ, because the instances of ProbLog programs are obtained by selecting clauses from the non-ground program while the instances of LPADs are selected from a grounding of the program.

From the equivalence between ground ProbLog programs and LPADs follows that Bayesian networks with binary variables can be translated into ProbLog programs [2]. For example, the classic burglary Bayesian network (see Figure 2 in [2]) can be translated into the following ProbLog program

$0.1 : burglary \qquad 0.2 : earthquake$
$0.1 : alarm \leftarrow \neg burglary, \neg earthquake$
$0.8 : alarm \leftarrow \neg burglary, earthquake$
$0.8 : alarm \leftarrow burglary, \neg earthquake$
$1.0 : alarm \leftarrow burglary, earthquake$

We will say that a ProbLog program like the one above is in Bayesian form.

Definition 1 (Bayesian form). *A ground ProbLog program P is in Bayesian form if, for every atom a of $H_B(P)$, all the clauses for a have bodies built over the same set of atoms D_a and there is a clause for every possible combination of signs of literals built over D_a.*

It was shown in [2] that a ground, finite and acyclic [3] LPAD can be translated into a Bayesian Logic Program (BLP) preserving the semantics. Since BLP encode Bayesian networks, this provide a way of translating such type of LPADs

into Bayesian networks. Thus we can translate a ground acyclic ProbLog program into a Bayesian network. The class of acyclic programs is an important one because the ProbLog proof procedure may not terminate for cyclic programs.

We present here the technique. Given a ground acyclic ProbLog program P, we build a Bayesian network by associating each atom a in $H_B(P)$ with a binary variable a with values *true* and *false*. Moreover, for each rule r of the form

$\alpha : h \leftarrow b_1, \ldots, b_m, \neg c_1, \ldots, \neg c_l$

we add to the Bayesian network a new variable V_r that has $b_1, \ldots, b_m, c_1, \ldots, c_l$ as parents and has the two values h and *none*, where *none* is a special atom that does not appear anywhere in the body of rules. The conditional probability table (CPT) of V_r is

	$V_r = h$	$V_r = none$
...	0.0	1.0
$b_1 = true, \ldots, b_m = true, c_1 = false, \ldots, c_l = false$	α	$1 - \alpha$
...	0.0	1.0

Moreover, each variable a with $a \in H_B(P)$ has as parents all the variables V_r of rules r that have a in the head. The CPT for a is the following:

	$a = true$	$a = false$
all the parents equal to *none*	0.0	1.0
remaining rows	1.0	0.0

We will now show that every ground ProbLog program can be translated into Bayesian form in a way that preserves the semantics. This is done by collecting, for each atom $a \in H_B(P)$, all the rules R_a with a in the head. From R_a, the atoms D_a that appear in bodies of rules for a are collected. Then a rule c for each combination of signs of literals built on D_a is generated. The probability α_c of c is given by the law of the probability of an or:

$$\alpha_c = 1 - \prod_{r \in R_a | body(c) \models body(r)} (1 - \alpha_r)$$

Thus the bodies of the clauses for the same atom a constitute the causes of a in a noisy or model. For example the program

$0.3 : a \leftarrow b \quad 0.2 : a \leftarrow c$

can be transformed into

$0.0 : a \leftarrow \neg b, \neg c \quad 0.2 : a \leftarrow \neg b, c$

$0.3 : a \leftarrow b, \neg c \quad 0.44 : a \leftarrow b, c$

Theorem 2. *The transformation into Bayesian form preserves the semantics for ground acyclic ProbLog programs.*

Proof. We will prove the theorem by showing that the Bayesian networks encoded by the two programs are equal. Let P and P' be the programs before and after the transformation. Consider an atom a and let $D_a = \{b_1, \ldots, b_n\}$ be the set of atoms on which a depends. Suppose that P contains k rules for a

$R_a = \{r_1, \ldots, r_k\}$ and that rule r_i is annotated with probability α_i. Let d_a be a vector of values for D_a, let V_{R_a} be the vector of variables corresponding to the rules of R_a and let v_{R_a} be a vector of values for V_{R_a}. The probability for $a = true$ given d_a in the network obtained from P is given by

$$P(a = true|d_a) = 1 - P(a = false|d_a) = 1 - \sum_{v_{R_a}} P(a = false, v_{R_a}|d_a) =$$
$$= 1 - \sum_{v_{R_a}} P(a = false|v_{R_a}, d_a)P(v_{R_a}|d_a) =$$
$$= 1 - \sum_{v_{R_a}} P(a = false|v_{R_a})P(v_{R_a}|d_a)$$

Let V_{r_i} be the variable associated to rule r_i and let v_{r_i} be the value for V_{r_i} in v_{R_a}, thus

$$P(a = true|d_a) = 1 - \sum_{v_{R_a}} P(a = false|v_{R_a}) \prod_{i=1}^{k} P(v_{r_i}|d_a)$$

The only value v_{R_a} of V_{R_a} for which $P(a = false|v_{R_a}) \prod_{i=1}^{k} P(v_{r_i}|d_a)$ is different from 0 is the one where every v_{r_i} is equal to *none*. In fact, if $v_{r_i} = a$, then $P(a = false|v_{R_a})$ is 0. Suppose that, given the values of d_a, the set of rules $T_a \subseteq R_a$ has the body true. For the case in which every v_{r_i} is equal to *none*, $P(v_{r_i}|d_a) = 1 - \alpha_i$ if $r_i \in T_a$, $P(v_{r_i}|d_a) = 1$ if $r_i \notin T_a$ and $P(a = false|v_{R_a}) = 1$. Thus we have

$$P(a = true|d_a) = 1 - \prod_{r_i \in T_a} (1 - \alpha_i)$$

which is exactly the law for the probability of an or.

4 Learning Ground Acyclic ProbLog Programs

We consider a learning problem of the following form [8]:
Given:

- a set E of examples that are couples $(I, \pi(I))$ where I is an interpretation and $\pi(I)$ is its associated probability, such that $\sum_{(I,\pi(I))\in E} \pi(I) = 1$
- a space of possible ground ProbLog programs S (described by a language bias LB)

Find: a ground ProbLog program $P \in S$ such that $\forall (I, \pi(I)) \in E \;\; \pi^P_{PB}(I) = \pi(I)$

Instead of a set of couples $(I, \pi(I))$, 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' by relative frequency.

The approach we propose for learning a ground acyclic ProbLog program consists in transforming the input data into a table, learning a Bayesian network from the table, converting the learned network into a ground ProbLog in Bayesian form and then identifying the noisy or relations in order to obtain a general ground acyclic ProbLog program.

The translation of the input interpretations into a table is done by considering each atom appearing in them as a binary random variable. Each interpretation I is then transformed into a binary vector B_I where the variable corresponding to atom a assumes value 1 if $a \in I$ and value 0 otherwise.

We obtain the table to be given as input to the Bayesian network learning algorithm by fixing the number of rows N of the table and replicating the binary vector B_I a number of times equal to $N \times \pi(I)$.

The translation of the learned Bayesian network into a general ground ProbLog program is performed by the algorithm Identification that analyzes the CPT of each atom and tries to identify the noisy or relations in order to apply the inverse of the transformation into Bayesian form. If it fails in finding such a relation, it returns the CPT converted into ProbLog rules.

Identification, shown in Figure 1, performs exhaustive search in the space of possible bodies of rules. It takes as input, besides the learned Bayesian network, also the parameters $MaxBodySize$, $MaxRules$ and ϵ that define, respectively, the maximum number of literals in the body of rules, the maximum number of rules for an atom and the error allowed. The first two parameters put a limit on the search space in order to contain the computational cost.

Identification analyzes the CPT of each atom in turn. For an atom a the algorithm first builds all possible sets of parents of a from cardinality 1 to cardinality $MaxBodySize$. Then, with function Select (show in Figure 2), it considers all possible combinations of signs for the atoms in each set, thus generating the possible bodies PB. In this phase, the possible bodies that appear in a row of the CPT where the probability of a is close to 0 (smaller than ϵ) are eliminated, because they cannot be possible causes.

Then the combinations of possible bodies are explored with function Explore (shown in Figure 3) by performing a depth first search in the space of subsets of PB. Explore is called with a current set of bodies and the set of bodies not yet added to the current set: it first check if the current set of bodies contains all possible parents of a and if the set of bodies respects the noisy or relation. If so, it returns the current set of bodies. Otherwise, if the current set of bodies has not yet reached cardinality $MaxRules$, it performs a cycle in which, at each iteration, it adds a possible body and calls itself recursively. If no set of bodies respecting the noisy or relation can be found, the empty set is returned and Identification translates the CPT directly into rules.

The test that a set of possible bodies respects the noisy or relation is performed by function Check (shown in Figure 4). The function considers only the rows with $P(a|row) > \epsilon$, because the bodies true in the other rows have already

been removed. The function first identifies the probability of each body considered as a single cause, looking for those rows where a single body is true. The probability of a body is given by the average of the probabilities of such rows. Then it checks that for all the rows the or law is respected with an error smaller than ϵ.

Fig. 1. Algorithm Identification

algorithm Identification(
  inputs : B : Bayesian network,
    ϵ: maximum error,
    $MaxBodySize$: integer,
    $MaxRules$: integer,
  returns : P : ProbLog program)

let B be a set of triples $(Variable, Parents, CPT)$ one for each variable
$P := \emptyset$
for every triple $(Variable, Parents, CPT)$ in B
  $F := \{r | r$ is a row of CPT such that $P(Variable = true|r) < \epsilon\}$
  $G :=$ set of all the possible subsets of $Parents$ from dimension 1 to dimension $MaxBodySize$
  $PB :=$Select(G, F)
  $Bodies :=$Explore$(PB, CPT, \emptyset, MaxRules, \epsilon)$
  if $Bodies \neq \emptyset$ then
    convert $Bodies$ into a set of rules R
  else
    convert CPT into a set of rules R
  $P := P \cup R$
return P

Let us show the behavior of the algorithm with an example. Consider an atom a that has b, c and d as parents and that has the conditional probability table shown below:

row	b	c	d	a	$\neg a$
1	*false*	*false*	*false*	0.00	1.00
2	*false*	*false*	*true*	0.30	0.70
3	*false*	*true*	*false*	0.00	1.00
4	*false*	*true*	*true*	0.30	0.70
5	*true*	*false*	*false*	0.00	1.00
6	*true*	*false*	*true*	0.30	0.70
7	*true*	*true*	*false*	0.40	0.60
8	*true*	*true*	*true*	0.58	0.42

Suppose that $MaxBodySize$ is 2, $MaxRules$ is 3 and that ϵ is 0.01. The set F of rows with $P(a|row) = 0$ is $F = \{\{\neg b, \neg c, \neg d\}, \{\neg b, c, \neg d\}, \{b, \neg c, \neg d\}\}$

Fig. 2. Function Select

```
function Select(
    inputs : G : set of sets of parents,
        F: rows with probability of the child variable < ε,
    returns : PB : set of possible bodies)

PB := ∅
for every parent set Par from G
    let B be the set of all possible assignment of signs to variables of Par
    for every b ∈ B
        if b ∉ F then
            PB := PB ∪ {b}
return PB
```

Fig. 3. Function Explore

```
function Explore(
    inputs : PB : possible bodies,
        CPT: conditional probability table
        Bodies: current set of bodies,
        MaxRules: maximum number of rules,
        ε: maximum error,
    returns : B : valid set of bodies)

if Bodies contains all possible parents and Check(Bodies, CPT, ε) then
    return Bodies
else
    if |Bodies| = MaxRules then
        return ∅
    else
        let PB be {b_1, b_2, ..., b_n}
        for i := 1 to n
            Bodies' :=Explore({b_{i+1}, ..., b_n}, CPT, Bodies ∪ {b_i}, MaxRules)
            if Bodies' ≠ ∅ then
                return Bodies'
        return ∅
```

Fig. 4. Function Check

function Check(
inputs : B : a set of bodies,
CPT: conditional probability table,
ϵ: maximum error,
returns : *Satisfy* : a Boolean value)

remove from CPT all the rows r with $P(Variable = true|r) < \epsilon$
for every body b in B
let R_b be the set of rows of CPT that contains only b true
let $p_b = \sum_{r \in R_b} \frac{P(Variable=true|r)}{|R_b|}$
for each row r of CPT
let B' be the set of bodies of B that are true in r
$TP := 1 - \prod_{b \in B'}(1 - p_b)$
if $|TP - P(Variable|r)| > \epsilon$ then
return *false*
return *true*

The set G of possible subsets of the set of $Parents$ with maximum size 2 is

$$G = \{\{b\}, \{c\}, \{d\}, \{b, c\}, \{b, d\}, \{c, d\}\}$$

The function Select is called and the set PB of possible bodies is returned

$$PB = \{\{d\}, \{b, c\}, \{b, d\}, \{\neg b, d\}, \{c, d\}, \{\neg c, d\}\}$$

Then Explore is called with $Bodies = \emptyset$. $Bodies$ does not contain all possible parents of a so Explore is called again with $Bodies = \{\{d\}\}$. $Bodies$ still does not contain all possible parents of a so Explore is called again with $Bodies = \{\{d\}, \{b, c\}\}$. Since all the parents are now present in $Bodies$, Check is called.

Rows 1, 3 and 5 are removed form CPT. The probabilities p_d and $p_{b,c}$ are computed: p_d is obtained from rows 2, 4 and 6 and has value 0.3 while $p_{b,c}$ is obtained form row 7 and has value 0.4.

Then the probabilities of a in rows 2, 4, 6, 7 and 8 are checked. In row 2, 4 and 6 only d is true, so it is checked that $|p_d - P(a|row)| < \epsilon$. This is true so Check continues. Row 7 has only b, c true and the test of $|p_{b,c} - P(a|row_7)| < \epsilon$ succeeds. In row 8, both d and b, c are true, so it is checked that $|TP - P(a|row_8)| < \epsilon$. TP is $1 - (1 - p_d)(1 - p_{b,c}) = 1 - (1 - 0.3)(1 - 0.4) = 1 - 0.42 = 0.58$ so the test succeeds and Check returns true.

Therefore b, c and d are recognized as valid bodies and the rules

$0.4 : a \leftarrow b, c \quad 0.3 : a \leftarrow d$

are returned.

With this approach, we may have problem when learning ProbLog programs that are not layered, i.e. when its Herbrand base cannot be divided into subsets (layers) such that each atom directly depends only on atoms from the previous

layer. In fact, in that case the probabilities in the CPT may not be estimated correctly.

For example, consider a dataset generated from the program

$0.6 : c \leftarrow a \quad 0.7 : c \leftarrow b$

$0.2 : d \leftarrow a, b \quad 0.9 : d \leftarrow c$

The case in which a and b are false and c is true never appears in the data because if a and b are false then so is c. Therefore the probabilities in the corresponding row of the CPT for d cannot be estimated. A typical approach used by Bayesian network learning algorithms is to assign probability 0.5 to d true and to d false. In this case the CPT of d does not respect the noisy or law that would assigns probability 0.9 to d true in that row and therefore the rules for d cannot be identified.

This problem does not appear if the program from which the data is generated is layered. For the program above the Herbrand base cannot be divided in layers because d depends on c and on a and b that belong to the layer preceding c.

5 Experiments

A series of experiments were performed for investigating the time complexity of the algorithm. A number of programs consisting of the definition of a single atom a have been generated: each program consists of two clauses, the programs differ for the number of atoms on which a depends that ranges from 4 to 10. The parents of a are distributed among the two clauses in order to have two bodies whose lengths differ by at most 1. Programs of this form were considered because they represent the worst case, since the bodies in PB are ordered from the shortest to the longest. From the programs, the CPT for a is produced with the transformation procedure presented in section 3.

Algorithm Identification is applied with the following parameters: $MaxRules$ is set to 3 and $MaxBodySize$ is set to values ranging from 3 to 10. The execution times in milliseconds are shown in Figure 5. The experiments were performed with Sicstus Prolog 3.12.5 on a Windows machine with a 2.00 GHz Pentium M and 1 Gb of RAM.

The missing points correspond to combinations for which the algorithm terminates because Sicstus gave an insufficient memory error: Sicstus 3.12.5 has a limitation of 256 Mb for the stack on 32 bit machines. For 10 parents the algorithm has successfully terminated only for $MaxBodySize = 3$ in 24.5 seconds while for higher values of $MaxBodySize$ has given an insufficient memory error,

For the points where $MaxBodySize < \lceil Parents/2 \rceil$ the algorithm returned a program in Bayesian form because the solution was outside the search space.

A number of experiments were conducted to test the feasibility of the whole approach: the aim was to learn back a ground ProbLog program. A few ProbLog programs were written, all the possible interpretations for them were generated and assigned a probability according to the semantics. The sets of annotated interpretations were then translated into tables and given as input to a Bayesian

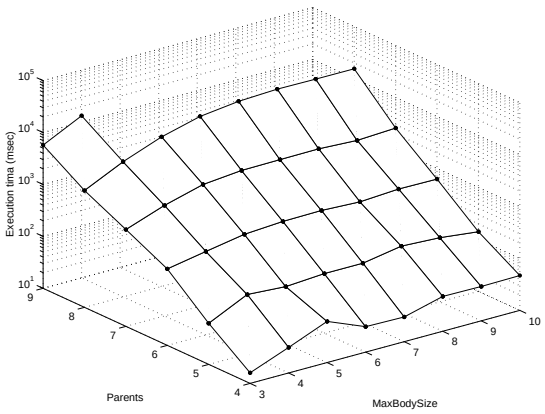

Fig. 5. Execution times

learning algorithm from the suite WEKA. Then the algorithm Identification was applied to the learned networks.

For example, the approach was tested on a program containing 9 atoms and 14 rules. From it, a table containing approximately 30,000 rows was generated and given as input to the implementation of the K2 algorithm available in WEKA. K2 exploits the ordering of the variables so the correct order was supplied to it. The other parameters were left at default values except for initAsNaiveBayes, set to false, and for the maximum number of parents of a node, set to 8.

K2 learned a Bayesian network in 0.3 seconds. Then Identification was applied with $\epsilon = 0.05$, $MaxBodySize = 3$ and $MaxRules = 3$. In 0.42 seconds Identification returned the original ProbLog program. None of the other Bayesian network learning algorithms available in WEKA were able to correctly discover the dependencies encoded by the original program.

Experiments with programs of similar complexity were performed using K2 and in all cases the original programs were returned.

6 Related Works

In [9] the authors propose an approach for revising ProbLog programs. The learning problem they consider consists in finding a subsets of clauses from a given program that maximizes the likelihood of a set of examples in the form of ground goals. Moreover, they set an upper limit to the cardinality of the program to be returned. Thus the approach in [9] is complementary to the one given here, where a set of interpretations is considered as input.

Another related work is [8] where the author proposes the algorithm ALLPAD for learning ground LPADs from interpretations. However, ALLPAD can only

learn LPADs with mutually exclusive bodies and thus it cannot learn ProbLog programs encoding a noisy or.

7 Conclusions

We have presented an approach for learning ground acyclic ProbLog programs from interpretations. The approach consists in translating the input interpretations into tabular form, applying a Bayesian network learning algorithm and then trying to identify noisy or relations in the learned network.

The identification algorithm has been experimentally tested and it was found feasible for a number of parents up to 8. Experiments of the overall approach showed that it was possible to perfectly recover ground programs of around 10 atoms and 14 rules.

8 Acknowledgements

This work has been partially supported by the PRIN 2005 project "Specification and verification of agent interaction protocols".

References

1. De Raedt, L., Kimmig, A., Toivonen, H.: Problog: A probabilistic prolog and its application in link discovery. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence. (2007) 2462–2467
2. Vennekens, J., Verbaeten, S., Bruynooghe, M.: Logic programs with annotated disjunctions. In: Proc. of the 20th Int. Conf. on Logic Programming. (2004)
3. Apt, K.R., Bezem, M.: Acyclic programs. New Generation Comput. **9**(3/4) (1991) 335–364
4. Clark, K.L.: Negation as failure. In: Logic and Databases. Plenum Press (1978)
5. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In Kowalski, R., Bowen, K.A., eds.: Proceedings of the 5th Int. Conf. on Logic Programming, MIT Press (1988) 1070–1080
6. Van Gelder, A., Ross, K.A., Schlipf, J.S.: The well-founded semantics for general logic programs. Journal of the ACM **38**(3) (1991) 620–650
7. Vennekens, J., Denecker, M., Bruynooghe, M.: Representing causal information about a probabilistic process. In: Proc. of the 10th Eur. Conf. on Logics in Artificial Intelligence. LNAI, Springer (September 2006)
8. Riguzzi, F.: ALLPAD: Approximate learning of logic programs with annotated disjunctions. In: Proceedings of the 16th International Conference on Inductive Logic Programming. Number 4455 in LNAI, Springer (2007)
9. De Raedt, L., Kersting, K., Kimmig, A., Revoredo, K., Toivonen, H.: Revising probabilistic prolog programs. In: Proceedings of the 16th International Conference on Inductive Logic Programming. Number 4455 in LNAI, Springer (2007)