Optimizing Inference for Probabilistic Logic Programs Exploiting Independence and Exclusiveness

Fabrizio Riguzzi

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

Abstract. Probabilistic Logic Programming (PLP) is gaining popularity due to its many applications in particular in Machine Learning. An important problem in PLP is how to compute the probability of queries. PITA is an algorithm for solving such a problem that exploits tabling, answer subsumption and Binary Decision Diagrams (BDDs). PITA does not impose any restriction on the programs. Other algorithms, such as PRISM, achieve a higher speed by imposing two restrictions on the program, namely that subgoals are independent and that clause bodies are mutually exclusive. Another assumption that simplifies inference is that clause bodies are independent. In this paper we present the algorithms PITA(IND,IND) and PITA(OPT). PITA(IND,IND) assumes that subgoals and clause bodies are independent. PITA(OPT) instead first checks whether these assumptions hold for subprograms and subgoals: if they hold, PITA(OPT) uses a simplified calculation, otherwise it resorts to BDDs. Experiments on a number of benchmark datasets show that PITA(IND,IND) is the fastest on datasets respecting the assumptions while PITA(OPT) is a good option when nothing is known about a dataset.

1 Introduction

Probabilistic Logic Programming (PLP) is an emerging field devoted to the study of the representation of uncertain information in logic programming. Its use is increasing in particular in the Machine Learning field [5], where many domains present complex and uncertain relations among the entities.

A wide variety of semantics and languages have been proposed in PLP. Among these, the distribution semantics [20] is probably the most used. Many languages adopt this semantics, such as Probabilistic Logic Programs [3], Independent Choice Logic [11], PRISM [21], pD [7], Logic Programs with Annotated Disjunctions (LPADs) [25] and ProbLog [4]. All these languages have the same expressive power as a theory in one language can be translated into another. LPADs offer the most general syntax as the constructs of all the other languages can be directly encoded in LPADs.

An important problem in PLP is computing the probability of queries or the problem of inference. Solving this problem in a fast way is fundamental especially

for Machine Learning applications, where a high number of queries has be to answered. PRISM [21] is a system that performs inference but restricts the class of programs it can handle: subgoals in the body of clauses must be independent and bodies of clauses with the same head must be mutually exclusive. These restrictions allow PRISM to be very fast. Recently, other algorithms have been proposed that lift these restrictions, such as Ailog2 [12], ProbLog [9], cplint [13, 14], SLGAD [15] and PITA [16, 18].

In particular, PITA associates to each subgoal an extra argument used to store a Binary Decision Diagram (BDD) that encodes the explanations for the subgoal. PITA uses a Prolog library that exposes the functions of a highly efficient BDD package: the conjunction of BDDs is used for handling conjunctions of subgoals in the body while the disjunction of BDDs is used for combining explanations for the same subgoal coming from different clauses. PITA exploits tabling and answer subsumption to effectively combine answers for the same subgoal and to store them for a fast retrieval. PITA was recently shown [17] to be highly customizable for specific settings in order to increase its efficiency. When the modeling assumptions of PRISM hold (independence of subgoals and exclusiveness of clauses), PITA can be specialized to PITA(IND,EXC) obtaining a system that is comparable or superior to PRISM in speed.

In this paper, we consider another special case that can be treated by specializing PITA, one where the bodies of clause with the same head are mutually independent, a situation first considered in [8]. This requires a different modification of PITA and PITA(IND,IND), the resulting system, is much faster on programs where these assumption holds.

In order to generalize these results, we propose PITA(OPT), a version of PITA that performs conjunctions and disjunctions of explanations for subgoals by first checking whether the independence or exclusiveness assumptions hold. Then, depending on the test results, it uses a simplified calculation or, if no assumption holds, it falls back on using BDDs.

In order to investigate the performances of PITA(IND,IND) and PITA(OPT) in comparison with the other systems, including the specialized ones, we performed a number of experiments on real and artificial datasets. The results show that PITA(IND,IND) is the fastest when the corresponding assumptions hold. Moreover, PITA(OPT) is much faster than general purpose algorithms such as PITA and ProbLog in some cases, while being only slightly slower in the other cases, so that PITA(OPT) can be considered a valid alternative when nothing is known about the program.

After presenting the basic concepts of PLP, we illustrate PITA. Then we present the modeling assumptions that simplify probability computations and the system PITA(IND,IND). The description of PITA(OPT) follows, together with the experiments performed on a number of real and artificial datasets. We then conclude and present directions for future work.

2 Probabilistic Logic Programming

The distribution semantics [20] is one of the most widely used semantics for probabilistic logic programming. In the distribution semantics, a probabilistic logic program defines a probability distribution over a set of normal logic programs (called *worlds*). The distribution is extended to a joint distribution over worlds and a query and the probability of the query is obtained from this distribution by marginalization.

The languages based on the distribution semantics differ in the way they define the distribution over logic programs. Each language allows probabilistic choices among atoms in clauses. We consider here LPADs for their general syntax. LPADs are sets of disjunctive clauses in which each atom in the head is annotated with a probability.

Formally a *Logic Program with Annotated Disjunctions* [25] consists of a finite set of annotated disjunctive clauses. An annotated disjunctive clause C_i is of the form $h_{i1} : \Pi_{i1}; \ldots; h_{in_i} : \Pi_{in_i} \leftarrow b_{i1}, \ldots, b_{im_i}$. 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 $\sum_{k=1}^{n_i} \Pi_{ik} \leq 1$. 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.

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

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

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. We are uncertain about whether the climate is cold but we know for sure that David and Robert have the flu.

We now present the distribution semantics for the case in which the program does not contain function symbols so that its Herbrand base is finite[1].

An *atomic choice* is a selection of the k-th atom for a grounding $C_i\theta_j$ of a probabilistic clause C_i and is represented by the triple (C_i, θ_j, k). An atomic choice represents an equation of the form $X_{ij} = k$ where X_{ij} is a random variable associated to $C_i\theta_j$. A set of atomic choices κ is *consistent* if $(C_i, \theta_j, k) \in \kappa, (C_i, \theta_j, m) \in \kappa \Rightarrow k = m$, i.e., only one head is selected for a ground clause.

A *composite choice* κ is a consistent set of atomic choices. The probability of a composite choice κ is $P(\kappa) = \prod_{(C_i,\theta_j,k)\in\kappa} \Pi_{ik}$. A *selection* σ is a total composite choice (one atomic choice for every grounding of each probabilistic clause). Let

[1] However, the distribution semantics for programs with function symbols has been defined as well [20, 12, 18].

us call S_T the set all selections. A selection σ identifies a logic program w_σ called a *world*. The probability of w_σ is $P(w_\sigma) = P(\sigma) = \prod_{(C_i,\theta_j,k)\in\sigma} \Pi_{ik}$. Since the program does not contain function symbols, the set of worlds is finite $W_T = \{w_1, \ldots, w_m\}$ and $P(w)$ is a distribution over worlds: $\sum_{w\in W_T} P(w) = 1$.

We can define the conditional probability of a query Q given a world w: $P(Q|w) = 1$ if Q is true in w and 0 otherwise. The probability of the query can then be obtained by marginalizing over the query

$$P(Q) = \sum_w P(Q, w) = \sum_w P(Q|w)P(w) = \sum_{w\models Q} P(w) \tag{1}$$

Example 2. For the LPAD T of Example 1, clause C_1 has two groundings, $C_1\theta_1$ with $\theta_1 = \{X/david\}$ and $C_1\theta_2$ with $\theta_2 = \{X/robert\}$, while clause C_2 has a single grounding $C_2\emptyset$. Since C_1 has three head atoms and C_2 two, T has $3 \times 3 \times 2$ worlds. The query *epidemic* is true in 5 of them and its probability is $P(epidemic) = 0.6{\cdot}0.6{\cdot}0.7{+}0.6{\cdot}0.3{\cdot}0.7{+}0.6{\cdot}0.1{\cdot}0.7{+}0.3{\cdot}0.6{\cdot}0.7{+}0.1{\cdot}0.6{\cdot}0.7 = 0.588$

Inference in probabilistic logic programming is performed by finding a covering set of explanations for queries.

A composite choice κ identifies a set of worlds ω_κ that contains all the worlds associated to a selection that is a superset of κ: i.e., $\omega_\kappa = \{w_\sigma | \sigma \in S_T, \sigma \supseteq \kappa\}$. We define the set of worlds identified by a set of composite choices K as $\omega_K = \bigcup_{\kappa\in K} \omega_\kappa$. Given a ground atom Q, a composite choice κ is an *explanation* for Q if Q is true in every world of ω_κ. In Example 1, the composite choice $\kappa_1 = \{(C_2, \emptyset, 1), (C_1, \{X/david\}, 1)\}$ is an explanation for *epidemics*. A set of composite choices K is *covering* with respect to Q if every world w_σ in which Q is true is such that $w_\sigma \in \omega_K$. For Example 1, a covering set of explanations for *epidemics* is $K = \{\kappa_1, \kappa_2\}$ where $\kappa_1 = \{(C_2, \emptyset, 1), (C_1, \{X/david\}, 1)\}$ and $\kappa_2 = \{(C_2, \emptyset, 1), (C_1, \{X/robert\}, 1)\}$. If we associate the variables X_{11} to $C_1\{X/david\}$, X_{12} to $C_1\{X/robert\}$ and X_{21} to $C_2\emptyset$, the query is true if the following formula is true:

$$f(\mathbf{X}) = X_{21} = 1 \wedge X_{11} = 1 \vee X_{21} = 1 \wedge X_{12} = 1. \tag{2}$$

Two composite choices κ_1 and κ_2 are *exclusive* if their union is inconsistent, i.e., if there exists a clause C_i and a substitution θ_j grounding C_i such that $(C_i, \theta_j, k) \in \kappa_1, (C_i, \theta_j, m) \in \kappa_2$ and $k \neq m$. A set K of composite choices is *mutually exclusive* if for all $\kappa_1 \in K, \kappa_2 \in K, \kappa_1 \neq \kappa_2 \Rightarrow \kappa_1$ and κ_2 are exclusive. As an illustration, the set of composite choices

$$\begin{aligned} K_2 = \{&\{(C_2, \emptyset, 1), (C_1, \{X/david\}, 1)\}, \\ &\{(C_2, \emptyset, 1), (C_1, \{X/david\}, 0), (C_1, \{X/robert\}, 1)\}\} \end{aligned}$$

is mutually exclusive for the theory of Example 1.

Given a covering set of explanations for a query, the query is true if the disjunction of the explanations in the covering set is true, where each explanation

is interpreted as the conjunction of all its atomic choices. In this way we obtain a formula in Disjunctive Normal Form (DNF).

The covering set of explanations that is found for a query is not necessarily mutually exclusive, so the probability of the query can not be computed by a summation as in Formula (1). The explanations have first to be made mutually exclusive so that a summation can be computed. This problem, known as disjoint sum, is #P-complete [24]. The most effective way to date of solving the problem makes use of Decision Diagram that are used to represent the DNF formula in a way that allows to compute the probability with a simple dynamic programming algorithm [4].

Since the random variables that are associated to atomic choices can assume multiple values, we need to use Multivalued Decision Diagrams (MDDs) [23]. An MDD represents a function $f(\mathbf{X})$ taking Boolean values on a set of multivalued variables $\mathbf{X}$ by means of a rooted graph that has one level for each variable. Each node n has one child for each possible value of the multivalued variable associated to n. The leaves store either 0 or 1. Given values for all the variables $\mathbf{X}$, an MDD can be used to 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. Since MDDs split paths on the basis of the values of a variable, the branches are mutually exclusive so a dynamic programming algorithm can be applied for computing the probability. Figure 1(a) shows the MDD corresponding to Formula (2).

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

A node n in a BDD has two children: the 1-child and the 0-child. 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. The following encoding, proposed in [19], gives good performances. 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 $k = n_i - 1$.

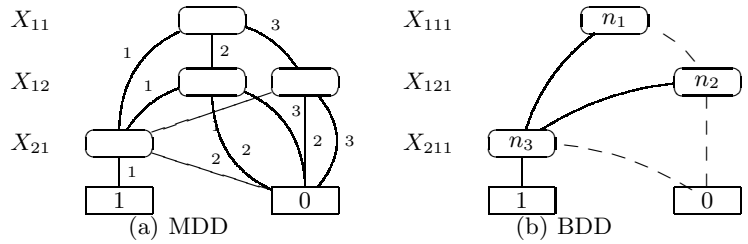


Fig. 1. Decision diagrams for Example 1

3 The PITA System

PITA computes the probability of a query from a probabilistic program in the form of an LPAD by first transforming the LPAD into a normal program containing calls for manipulating BDDs. The idea is to add an extra argument to each subgoal to store a BDD encoding the explanations for the answers of the subgoal. The extra arguments of subgoals are combined using a set of general library functions:

- *init, end*: initialize and terminate the data structures for manipulating BDDs;
- *bdd_zero(-D), bdd_one(-D), bdd_and(+D1,+D2,-DO), bdd_or(+D1,+D2, -DO), bdd_not(+D1,-DO)*: Boolean operations between BDDs;
- *get_var_n(+R,+S,+Probs,-Var)*: returns the multi-valued random variable associated to rule R with grounding substitution S and list of probabilities *Probs*;
- *bdd_equality(+Var,+Value,-D)*: D is the BDD representing *Var=Value*, i.e. that the multivalued random variable *Var* is assigned *Value*;
- *ret_prob(+D,-P)*: returns the probability of the BDD D.

These functions are implemented in C as an interface to the CUDD library for manipulating BDDs. A BDD is represented in Prolog as an integer that is a pointer in memory to the root node of the BDD

The PITA transformation applies to atoms, literals and clauses. The transformation for a head atom h, $PITA_h(h)$, is h with the variable D added as the last argument. Similarly, the transformation for a positive body literal b_j, $PITA_b(b_j)$, is b_j with the variable D_j added as the last argument. The transformation for a negative body literal $b_j = \neg a_j$, $PITA_b(b_j)$, is the Prolog conditional $(PITA'_b(a_j) \rightarrow not(DN_j, D_j); one(D_j))$, where $PITA'_b(a_j)$ is a_j with the variable DN_j added as the last argument. In other words, the input data structure, DN_j, is negated if it exists; otherwise the data structure for the constant function 1 is returned.

The disjunctive clause $C_r = h_1 : \Pi_1 \vee \ldots \vee h_n : \Pi_n \leftarrow b_1, \ldots, b_m.$ where the parameters sum to 1, is transformed into the set of clauses $PITA(C_r)$:

$PITA(C_r, i) = PITA_h(h_i) \leftarrow one(DD_0),$
$PITA_b(b_1), and(DD_0, D_1, DD_1), \ldots,$
$PITA_b(b_m), and(DD_{m-1}, D_m, DD_m),$
$get\_var\_n(r, VC, [\Pi_1, \ldots, \Pi_n], Var),$
$equality(Var, i, \Pi_i, DD), and(DD_m, DD, D).$

for $i = 1, \ldots, n$, where VC is a list containing each variable appearing in C_r.

The predicates $one/1$, $not/2$, $and/3$ and $equality/4$ are defined by

$one(D) \leftarrow bdd\_one(D).$
$not(A, B) \leftarrow bdd\_not(A, B).$
$and(A, B, C) \leftarrow bdd\_and(A, B, C).$
$equality(V, I, \_P, D) \leftarrow bdd\_equality(V, I, D).$

PITA uses tabling and a feature called *answer subsumption* available in XSB Prolog that, when a new answer for a tabled subgoal is found, combines old answers with the new one according to a partial order or lattice. For example, if the lattice is on the second argument of a binary predicate p, answer subsumption may be specified by means of the declaration *table p(_,or/3- zero/1)* where $zero/1$ is the bottom element of the lattice and $or/3$ is the join operation of the lattice. Thus if a table has an answer $p(a, d_1)$ and a new answer $p(a, d_2)$ is derived, the answer $p(a, d_1)$ is replaced by $p(a, d_3)$, where d_3 is obtained by calling $or(d_1, d_2, d_3)$.

In PITA various predicates of the transformed program should be declared as tabled. For a predicate p/n, the declaration is *table p(_1,...,_n,or/3-zero/1)*, which indicates that answer subsumption is used to form the disjunction of BDDs, with:

$zero(D) \leftarrow bdd\_zero(D).$
$or(A, B, C) \leftarrow bdd\_or(A, B, C).$

At a minimum, the predicate of the goal and all the predicates appearing in negative literals should be tabled with answer subsumption. However, it is usually better to table every predicate whose answers have multiple explanations and are going to be reused often.

4 Modeling Assumptions

PRISM makes the following modeling assumptions [22]:

1. the probability of a conjunction (A, B) is computed as the product of the probabilities of A and B (*and independence assumption*),
2. the probability of a disjunction $(A; B)$ is computed as the sum of the probabilities of A and B (*or exclusiveness assumption*),

These assumptions can be stated more formally by referring to explanations. Given an explanation κ, let $RV(\kappa) = \{C_i\theta_j | (C_i, \theta_j, k) \in \kappa\}$. Given a set of explanations K, let $RV(K) = \bigcup_{\kappa \in K} RV(\kappa)$. Two sets of explanations, K_1 and K_2, are *independent* if $RV(K_1) \cap RV(K_2) = \emptyset$ and *exclusive* if, $\forall \kappa_1 \in K_1, \kappa_2 \in K_2$, κ_1 and κ_2 are exclusive.

Assumption 1 means that, when deriving a covering set of explanations for a goal, the covering sets of explanations K_i and K_j for two ground subgoals in the body of a clause are independent.

Assumption 2 means that, when deriving a covering set of explanations for a goal, two covering sets of explanations K_i and K_j obtained for a ground subgoal h from two different clauses are exclusive. This implies that the atom h is derived using clauses that have mutually exclusive bodies, i.e., that their bodies are not both true in a world.

PRISM [21] and PITA(IND,EXC) [17] exploit these assumptions to speed up the computation. PITA(IND,EXC) differs from PITA in the definition of the $one/1$, $zero/1$, $not/2$, $and/3$, $or/3$ and $equality/4$ predicates that now work on probabilities P rather than on BDDs. Their definitions are

$zero(0)$.
$one(1)$.
$not(A, B) \leftarrow B\ is\ 1 - A$.
$and(A, B, C) \leftarrow C\ is\ A * B$.
$or(A, B, C) \leftarrow C\ is\ A + B$.
$equality(V, \_N, P, P)$.

The or exclusiveness assumption can be replaced by

3. the probability of a disjunction $(A; B)$ is computed as if A and B were independent (*or independence assumption*).

This means that, when deriving a covering set of explanations for a goal, two covering sets of explanations K_i and K_j obtained for a ground subgoal h from two different clauses are independent. PITA(IND,EXC) can exploit this assumption by modifying the $or/3$ predicate in this way

$or(A, B, P) \leftarrow P\ is\ A + B - A * B$.

We call PITA(IND,IND) the resulting system.

The exclusiveness assumption for conjunctions of literals means that the conjunction is true in 0 worlds and thus has always a probability of 0 so it does not make sense to consider a PITA(EXC,_) system.

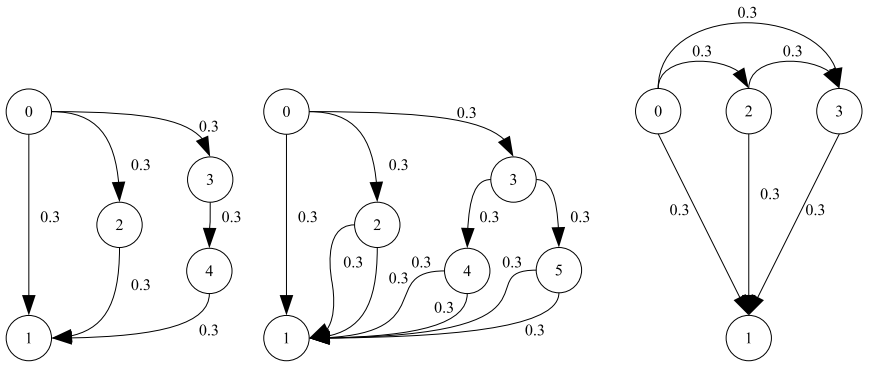
An example of a program satisfying assumptions 1 and 3 is the following

$path(Node, Node)$.
$path(Source, Target) : 0.3 \leftarrow edge(Source, Node), path(Node, Target)$.
$edge(0, 1) : 0.3$.
...

depending on the structure of the graph. For example, the graphs in Figures 2(a) and 2(b) respect these assumptions for the query $path(0, 1)$. Similar graphs of increasing sizes can be obtained with the procedures presented in [1]. We call the first graph type a "lanes" graph and the second a "branches" graph. The graphs of the type of the one in Figure 2(c), called "parachutes" graphs, instead, satisfy only Assumption 1 for the query $path(0, 1)$.

All three types of graphs respect Assumption 1 because, when deriving the goal $path(0, 1)$, paths are built incrementally starting from node 0 and adding one edge at a time with the second clause of the definition of $path/2$. Since the edge that is added does not appear in the following path, the assumption is respected.

Lanes and branches graphs respect Assumption 3 because, when deriving the goal $path(0, 1)$, ground instantiations of the second path clause have $path(i, 1)$ in the head and originate atomic choices of the form $(C_2, \{Source/i, Target/1, Node/j\}, 1)$. Explanations for $path(i, 1)$ contain also atomic choices $(E_{i,j}, \emptyset, 1)$ for every fact $edge(i, j) : 0.3$ in the path. In the lanes graph each node except 0 and 1 lies on a single path, so the explanations for $path(i, 1)$ do not share random variables. In the branches graphs, each explanation for $path(i, 1)$ depends on a different set of edges. In the parachutes graph instead this is not true: for example, the path from 3 to 1 shares the edge from 3 to 1 with the path 2, 3, 1.



(a) Lanes (b) Branches (c) Parachutes

Fig. 2. Graphs

5 PITA(OPT)

PITA(OPT) differs from PITA because, before applying BDD logical operators between sets of explanations, it checks for the truth of the assumptions. If they hold, then simplified probability computations are used.

The data structures used to store probabilistic information in PITA(OPT) are couples (P, T) where P is a real number representing a probability and T is a term formed with the functor symbols $zero/0, one/0, c/2, or/2, and/2, not/1$ and the integers. If T is an integer it represents a pointer to the root node of a BDD. If T is not an integer, it represents a Boolean expression of which the terms of the form $zero$, one, $c(var, val)$ or the integers represent the base case: $c(var, val)$ indicates the equation $var = val$ while an integer indicates a BDD. In this way we are able to represent Boolean formulas by means of a BDD, by means of a Prolog term or a combination thereof. The last case happens when an expression has been only partially converted to a BDD.

For example, *or(and(0x94ba008,c(1,1)), not(c(2,3)))* represents the expression: $(B \wedge X_1 = 1) \vee \neg(X_2 = 3)$ where B is the Boolean function represented by the BDD whose root node address in memory is *0x94ba008*.

PITA(OPT) differs from PITA also in the definition of $zero/1$, $one/1$, $not/2$, $and/3$, $or/3$ and $equality/4$ that now work on couples (P, T) rather than on BDDs. $eqaulity/4$ is defined as

$equality(V, N, P, (P, c(V, N)))$.

The $one/1$ and $zero/1$ predicates are defined as

$zero((0, zero))$.

$one((1, one))$.

The $or/3$ and $and/3$ predicates first check whether the independence or the exclusiveness assumption holds. If so, they update the value of the probability using the appropriate formula and return a compound term. If not, then they "evaluate" the terms, turning them into BDDs, applying the corresponding operation and returning the resulting BDD together with the probability it represents:

$or((PA, TA), (PB, TB), (PC, or(TA, TB))) \leftarrow ind(TA, TB), !,$
$PC\ is\ PA + PB - PA * PB.$

$or((PA, TA), (PB, TB), (PC, or(TA, TB))) \leftarrow exc(TA, TB), !, PC\ is\ PA + PB.$

$or((\_PA, TA), (\_PB, TB), (PC, TC)) \leftarrow ev(TA, TA1), ev(TB, TB1),$
$bdd\_or(TA1, TB1, TC), ret\_prob(TC, PC).$

$and((PA, TA), (PB, TB), (PC, and(TA, TB))) \leftarrow ind(TA, TB), !, PC\ is\ PA * PB.$

$and((\_PA, TA), (\_PB, TB), \_) \leftarrow exc(TA, TB), !, fail.$

$and((\_PA, TA), (\_PB, TB), (PC, TC)) \leftarrow ev(TA, TA1), ev(TB, TB1),$
$bdd\_and(TA1, TB1, TC), ret\_prob(TC, PC).$

where $ev/2$ evaluates a term returning a BDD. The $not/2$ predicate is very simple: it complements the probability and returns a new term:

$not((P, B), (P1, not(B))) \leftarrow !, P1\ is\ 1 - P.$

When checking for exclusiveness between two terms, if one of them is an integer, then the other is evaluated to obtain a BDD and the conjunction of the two BDDs is computed. If the result is equal to the 0 function, this means that the terms are exclusive. Otherwise the predicate $exc/2$ recurses through the structure of the two terms. The following code defines $exc/2$:

$exc(A, B) \leftarrow integer(A), !, ev(B, BB), bdd\_and(A, BB, C), zero(Z), Z = C.$
$exc(A, B) \leftarrow integer(B), !, ev(A, AB), bdd\_and(AB, B, C), zero(Z), Z = C.$
$exc(zero, \_) \leftarrow !.$
$exc(\_, zero) \leftarrow !.$
$exc(c(V, N), c(V, N1)) \leftarrow !, N \backslash = N1.$
$exc(c(V, N), or(X, Y)) \leftarrow !, exc(c(V, N), X), exc(c(V, N), Y).$
$exc(c(V, N), and(X, Y)) \leftarrow !, (exc(c(V, N), X); exc(c(V, N), Y)).$
$exc(or(A, B), or(X, Y)) \leftarrow !, exc(A, X), exc(A, Y), exc(B, X), exc(B, Y).$
$exc(or(A, B), and(X, Y)) \leftarrow !, (exc(A, X); exc(A, Y)), (exc(B, X); exc(B, Y)).$
$exc(and(A, B), and(X, Y)) \leftarrow !, exc(A, X); exc(A, Y); exc(B, X); exc(B, Y).$
$exc(and(A, B), or(X, Y)) \leftarrow !, (exc(A, X); exc(B, X)), (exc(A, Y); exc(B, Y)).$
$exc(not(A), A) \leftarrow !.$
$exc(not(A), and(X, Y)) \leftarrow !, exc(not(A), X); exc(not(A), Y).$
$exc(not(A), or(X, Y)) \leftarrow !, exc(not(A), X), exc(not(A), Y).$
$exc(A, or(X, Y)) \leftarrow !, exc(A, X), exc(A, Y).$
$exc(A, and(X, Y)) \leftarrow exc(A, X); exc(A, Y).$

In the test of independence between two terms, if one of them is a BDD, then the library function $bdd\_ind(B1, B2, I)$ is called. Such a function is implemented in C and uses the CUDD function `Cudd_SupportIndex` that returns an array indicating which variables appear in a BDD (the support variables). $bdd\_ind(B1, B2, I)$ checks whether there is an intersection between the set of support variables of $B1$ and $B2$ and returns $I = 1$ if the intersection is empty. If none of the two terms are BDDs, then $ind/2$ visits the structure of the first term until it reaches either a BDD or an atomic choice. In the latter case it checks for the absence of the variable in the second term with the predicate $absent/2$. $ind/2$ is defined as:

$ind(one, \_) \leftarrow !.$
$ind(zero, \_) \leftarrow !.$
$ind(\_, one) \leftarrow !.$
$ind(\_, zero) \leftarrow !.$
$ind(A, B) \leftarrow integer(A), !, ev(B, BB), bdd\_ind(A, BB, I), I = 1.$
$ind(A, B) \leftarrow integer(B), !, ev(A, AB), bdd\_ind(AB, B, I), I = 1.$
$ind(c(V, \_N), B) \leftarrow !, absent(V, B).$
$ind(or(X, Y), B) \leftarrow !, ind(X, B), ind(Y, B).$
$ind(and(X, Y), B) \leftarrow !, ind(X, B), ind(Y, B).$
$ind(not(A), B) \leftarrow ind(A, B).$
$absent(V, c(V1, \_N1)) \leftarrow !, V \backslash = V1.$
$absent(V, or(X, Y)) \leftarrow !, absent(V, X), absent(V, Y).$
$absent(V, and(X, Y)) \leftarrow !, absent(V, X), absent(V, Y).$
$absent(V, not(A)) \leftarrow absent(V, A).$

The predicates $exc/3$ and $ind/3$ define sufficient conditions for exclusion and independence respectively.

The evaluation of a term, i.e., its transformation into a BDD, is defined as

$ev(B, B) \leftarrow integer(B), !.$
$ev(zero, B) \leftarrow !, bdd\_zero(B).$
$ev(one, B) \leftarrow !, bdd\_one(B).$
$ev(c(V, N), B) \leftarrow !, bdd\_equality(V, N, B).$
$ev(and(A, B), C) \leftarrow !, ev(A, BA), ev(B, BB), bdd\_and(BA, BB, C).$
$ev(or(A, B), C) \leftarrow !, ev(A, BA), ev(B, BB), bdd\_or(BA, BB, C).$
$ev(not(A), C) \leftarrow ev(A), bdd\_not(A, C).$

6 Experiments

In this section we compare PITA, PITA(IND,EXC), PITA(IND,IND), PITA(OPT), PRISM and ProbLog on a number of datasets. We first consider the graphs of figures 2 and the path program shown in Section 4. The execution times of PITA(OPT), PITA(IND,IND), PITA and ProbLog for graphs of increasing sizes are shown in Figure 3 for lanes, Figure 4 for branches and Figure 5 for parachutes graphs. Figure 3 clearly show the advantage of the (IND,IND) modeling assumptions that allow PITA(IND,IND) to achieve high speed and scalability. PITA(OPT) has lower performances but is still much better than PITA and ProbLog. Figure 4 again shows the good performances of PITA(IND,IND). Here PITA(OPT) is faster and more scalable than PITA and ProbLog as well.

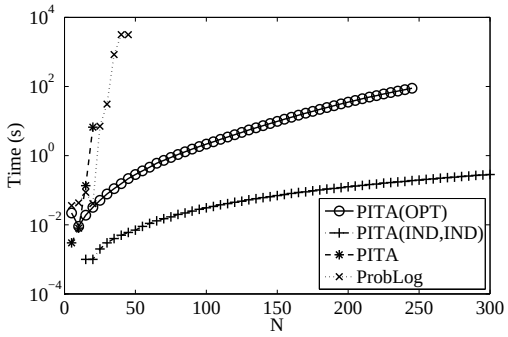


Fig. 3. Execution times on the lanes graphs

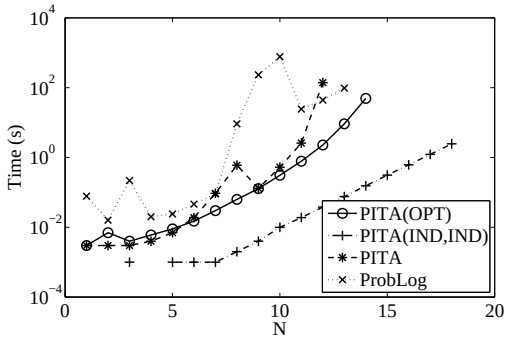


Fig. 4. Execution times on the branches graphs

The parachute graphs do not respect the (IND,IND) modeling assumption so PITA(IND,IND) has not been applied to this dataset. Figure 5 compares PITA, PITA(OPT) and ProbLog and shows that PITA(OPT) is the fastest and most scalable.

The blood type dataset [10] encodes the genetic inheritance of blood type in families of increasing size. In this dataset, the (IND,EXC) assumption holds so PRISM can also be used. Figure 6(a) shows the execution times of the algorithms. As can be seen, PITA(IND,EXC) is much faster and more scalable than PRISM that exploits the same assumptions and PITA is slightly faster than PITA(OPT).

The growing head dataset [10] contains propositional programs with heads of increasing size. In this dataset, neither the (IND,EXC) nor the (IND,IND) assumptions hold so we compare PITA(OPT) with PITA and ProbLog. Figure 6(b) shows that PITA(OPT) is faster than PITA for sizes larger than 12 and is able to solve 5 more programs.

The growing negated body dataset [10] contains propositional programs with bodies of increasing size. Also in this dataset neither the (IND,EXC) nor the (IND,IND) assumptions hold so we compare PITA(OPT) with PITA and ProbLog.

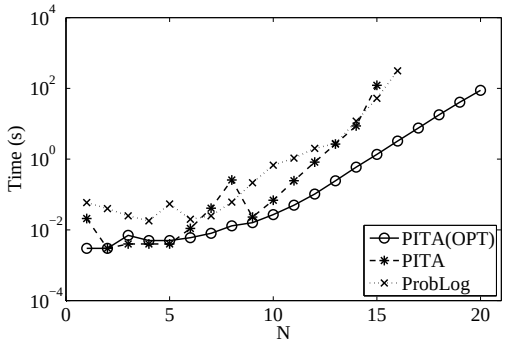
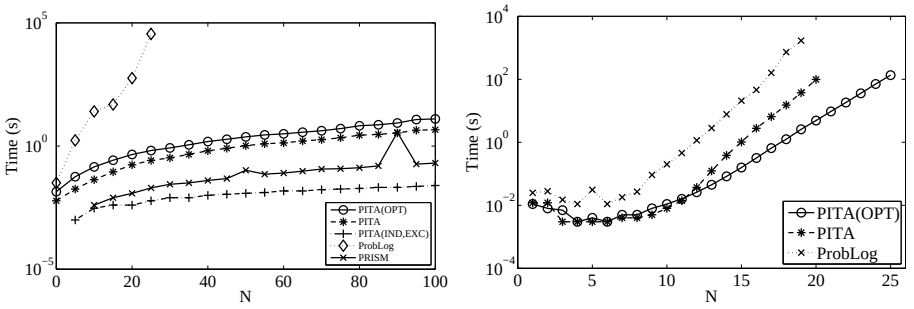


Fig. 5. Execution times on the parachutes graphs



(a) Execution times on the blood type dataset

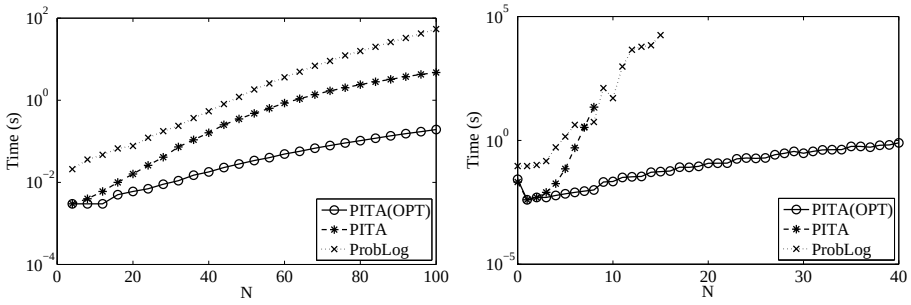
(b) Execution times on the growing head dataset

Fig. 6. Experiments on the blood type and growing head datasets

Figure 7(a) shows an even larger improvement of PITA(OPT) with respect to PITA and ProbLog.

The UWCSE dataset [10] encodes a university domain. Also in this dataset neither the (IND,EXC) nor the (IND,IND) assumptions hold so we compare PITA(OPT) with PITA and ProbLog. Figure 7(a) shows again a large improvement of PITA(OPT) with respect to PITA and ProbLog.

These experiments show that, if we know that the program respects either the (IND,EXC) or the (IND,IND) assumptions, using the corresponding algorithm gives the best results. If nothing is known about the program, PITA(OPT) is a good option since it gives very good results in datasets where these assumptions hold for the whole program or for parts of it, while paying a limited penalty on datasets where these assumptions are completely false.



(a) Execution times on the growing negated body dataset

(b) Execution times on the UWCSE dataset

Fig. 7. Experiments on the growing negated body and UWCSE datasets

7 Conclusions

We have discussed how assumptions on the program can much simplify the computation of the probability of queries. When subgoals in the body of clauses are independent and the bodies of clauses for the same atom are independent as well, PITA(IND,IND) achieves a high speed, being faster than general purpose inference algorithm. When we don't know whether these assumptions hold or not, PITA(OPT) can be used that applies simplified probability computations when the corresponding assumptions hold on the program or on parts of it.

In the future we plan to investigate other optimization techniques such as those presented in [2, 6].

References

1. Bragaglia, S., Riguzzi, F.: Approximate inference for logic programs with annotated disjunctions. In: International Conference on Inductive Logic Programming. LNAI, vol. 6489, pp. 30–37. Springer (2011)
2. Van den Broeck, G., Taghipour, N., Meert, W., Davis, J., De Raedt, L.: Lifted probabilistic inference by first-order knowledge compilation. In: International Joint Conference on Artificial Intelligence. pp. 2178–2185. AAAI Press/IJCAI (2011)
3. Dantsin, E.: Probabilistic logic programs and their semantics. In: Russian Conference on Logic Programming. LNCS, vol. 592, pp. 152–164. Springer (1991)
4. De Raedt, L., Kimmig, A., Toivonen, H.: ProbLog: A probabilistic Prolog and its application in link discovery. In: International Joint Conference on Artificial Intelligence. pp. 2462–2467 (2007)
5. De Raedt, L., Frasconi, P., Kersting, K., Muggleton, S. (eds.): Probabilistic Inductive Logic Programming - Theory and Applications, LNCS, vol. 4911. Springer (2008)
6. Fierens, D., Van den Broeck, G., Thon, I., Gutmann, B., De Raedt, L.: Inference in probabilistic logic programs using weighted CNF's. In: Conference on Uncertainty in Artificial Intelligence. pp. 211–220. AUAI Press (2011)

7. Fuhr, N.: Probabilistic datalog: Implementing logical information retrieval for advanced applications. J. Am. Soc. Inf. Sci. 51(2), 95–110 (2000)
8. Hommersom, A., Lucas, P.: Generalising the interaction rules in probabilistic logic. In: International Joint Conference on Artificial Intelligence (2011)
9. Kimmig, A., Demoen, B., Raedt, L.D., Costa, V.S., Rocha, R.: On the implementation of the probabilistic logic programming language problog. Theor. Prac. Log. Prog. 11(2-3), 235–262 (2011)
10. Meert, W., Struyf, J., Blockeel, H.: CP-Logic theory inference with contextual variable elimination and comparison to BDD based inference methods. In: International Conference on Inductive Logic Programming. LNCS, vol. 5989, pp. 96–109. Springer (2009)
11. Poole, D.: The independent choice logic for modelling multiple agents under uncertainty. Artif. Intell. 94(1-2), 7–56 (1997)
12. Poole, D.: Abducing through negation as failure: stable models within the independent choice logic. J. Log. Program. 44(1-3), 5–35 (2000)
13. Riguzzi, F.: A top down interpreter for LPAD and CP-logic. In: Congress of the Italian Association for Artificial Intelligence. LNAI, vol. 4733, pp. 109–120. Springer (2007)
14. Riguzzi, F.: Extended semantics and inference for the Independent Choice Logic. Logic J. IGPL 17(6), 589–629 (2009)
15. Riguzzi, F.: SLGAD resolution for inference on Logic Programs with Annotated Disjunctions. Fundam. Inform. 102(3-4), 429–466 (2010)
16. Riguzzi, F., Swift, T.: Tabling and Answer Subsumption for Reasoning on Logic Programs with Annotated Disjunctions. In: Technical Communications of the International Conference on Logic Programming. LIPIcs, vol. 7, pp. 162–171. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik (2010)
17. Riguzzi, F., Swift, T.: The PITA system: Tabling and answer subsumption for reasoning under uncertainty. Theor. Prac. Log. Prog., International Conference on Logic Programming Special Issue 11(4–5), 433–449 (2011)
18. Riguzzi, F., Swift, T.: Well-definedness and efficient inference for probabilistic logic programming under the distribution semantics. Theor. Prac. Log. Prog., Convegno Italiano di Logica Computazionale Special Issue (2012), to appear
19. Sang, T., Beame, P., Kautz, H.A.: Solving bayesian networks by weighted model counting. In: National Conference on Artificial Intelligence. pp. 475–482. AAAI Press / The MIT Press (2005)
20. Sato, T.: A statistical learning method for logic programs with distribution semantics. In: International Conference on Logic Programming. pp. 715–729. MIT Press (1995)
21. Sato, T., Kameya, Y.: Prism: A language for symbolic-statistical modeling. In: International Joint Conference on Artificial Intelligence. pp. 1330–1339 (1997)
22. Sato, T., Zhou, N.F., Kameya, Y., Izumi, Y.: PRISM User's Manual (Version 2.0) (2010)
23. Thayse, A., Davio, M., Deschamps, J.P.: Optimization of multivalued decision algorithms. In: International Symposium on Multiple-Valued Logic. pp. 171–178 (1978)
24. Valiant, L.G.: The complexity of enumeration and reliability problems. SIAM J. Comp. 8(3), 410–421 (1979)
25. 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)