Optimizing Inference for Probabilistic Logic Programs Exploiting Independence and Exclusiveness

Fabrizio Riguzzi

ENDIF – University of Ferrara, Italy
fabrizio.riguzzi@unife.it
Combining Logic and Probability

- Useful to model domains with complex and uncertain relationships among entities
- Many approaches proposed in the areas of Logic Programming, Uncertainty in AI, Machine Learning, Databases
- Logic Programming: Distribution Semantics [Sato, 1995]
- A probabilistic logic program defines a probability distribution over normal logic programs (called instances or possible worlds or simply worlds)
- The distribution is extended to a joint distribution over worlds and interpretations (or queries)
- The probability of a query is obtained from this distribution
Probabilistic Logic Programming (PLP) Languages under the Distribution Semantics

- Probabilistic Logic Programs [Dantsin, 1991]
- Probabilistic Horn Abduction [Poole, 1993], Independent Choice Logic (ICL) [Poole, 1997]
- PRISM [Sato, 1995]
- Logic Programs with Annotated Disjunctions (LPADs) [Vennekens et al., 2004]
- ProbLog [De Raedt et al., 2007]
- They differ in the way they define the distribution over logic programs
A Logic Program with Annotated Disjunctions consists of a finite set of annotated disjunctive clauses of the form

\[ h_{i1} : \Pi_{i1} ; \ldots ; h_{in} : \Pi_{in} \leftarrow b_{i1} , \ldots , b_{im} \]

- \( h_{i1} , \ldots , h_{in} \) are logical atoms and \( b_{i1} , \ldots , b_{im} \) are logical literals,
- \( \Pi_{i1} , \ldots , \Pi_{in} \) are real numbers in the interval \([0,1]\) such that \( \sum_{k=1}^{ni} \Pi_{ik} \leq 1 \).

Example

The following LPAD \( T \) encodes a very simple model of the development of an epidemic or a pandemic:

\[
C_1 = \text{epidemic} : 0.6; \text{pandemic} : 0.3 \leftarrow \text{flu}(X), \text{cold}.
C_2 = \text{cold} : 0.7.
C_3 = \text{flu}(david).
C_4 = \text{flu}(robert).
\]
Case of no function symbols: finite Herbrand universe, finite set of groundings of each disjunctive clause

- **Atomic choice**: selection of the $i$-th atom for grounding $C\theta$ of clause $C$

  Represented with the triple $(C, \theta, i)$

- **Composite choice** $\kappa$: consistent set of atomic choices

  $\kappa = \{(C_1, \{X/\text{david}\}, 1), (C_1, \{X/\text{david}\}, 2)\}$ not consistent

- The probability of composite choice $\kappa$ is

  $$P(\kappa) = \prod_{(C, \theta, i) \in \kappa} P_0(C, i)$$
Distribution Semantics

- **Selection** $\sigma$: a total composite choice (one atomic choice for every grounding of each clause)
- A selection $\sigma$ identifies a logic program $w_\sigma$ called **world**
- The probability of $w_\sigma$ is $P(w_\sigma) = P(\sigma) = \prod_{(C, \theta, i) \in \sigma} P_0(C, i)$
- Finite set of worlds: $W_T = \{w_1, \ldots, w_m\}$
- $P(w)$ distribution over worlds: $\sum_{w \in W_T} P(w) = 1$
Herbrand base $H_T = \{A_1, \ldots, A_n\}$

Query: $(A_j = true) = a_j$

$P(a_j \mid w) = 1$ if $A_j$ is true in $w$ and 0 otherwise

$P(a_j) = \sum_w P(a_j, w) = \sum_w P(a_j \mid w)P(w) = \sum_{w \models A_j} P(w)$
A composite choice $\kappa$ identifies the set of worlds
$$\omega_\kappa = \{ w_\sigma | \sigma \in S_T, \sigma \supseteq \kappa \}.$$  

A set of composite choices $K$ identifies the set of worlds
$$\omega_K = \bigcup_{\kappa \in K} \omega_\kappa.$$ 

Given a ground atom $Q$, a composite choice $\kappa$ is an explanation for $Q$ if $Q$ is true in every world of $\omega_\kappa$. 

A set of composite choices $K$ is covering with respect to $Q$ if every world $w_\sigma$ in which $Q$ is true is such that $w_\sigma \in \omega_K$. 

Two composite choices $\kappa_1$ and $\kappa_2$ are exclusive if their union is inconsistent. 

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 $\kappa_2$ are exclusive.
**Inference**

- 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.
- Each explanation is interpreted as the conjunction of all its atomic choices.
- The covering set of explanations that is found for a query is not necessarily mutually exclusive, so the probability of the query cannot be computed by a summation.
- The problem of computing the probability, known as disjoint sum, is \#P-complete [Valiant, 1979].
- The most effective way to date of solving the problem makes use of Decision Diagrams.
PITA [Riguzzi and Swift, 2010, Riguzzi and Swift, 2012] computes the probability of a query by first transforming the LPAD into a normal program.

PITA adds an extra argument to each subgoal to store a BDD.

The extra arguments of subgoals are combined using a set of general library functions:

- \textit{init, end}: initialize and terminate the data structures.
- \textit{bdd\_zero}(-D), \textit{bdd\_one}(-D), \textit{bdd\_and}(+D1,+D2,-DO), \textit{bdd\_or}(+D1,+D2, -DO), \textit{bdd\_not}(+D1,-DO): Boolean operations between BDDs;
- \textit{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 \( Probs \);
- \textit{bdd\_equality}(+Var,+Value,-D): \( D \) is the BDD representing \( Var=\text{Value} \);
- \textit{ret\_prob}(+D,-P): returns the probability of the BDD \( D \).
The PITA transformation applies to atoms, literals and clauses.

- $PITA_h(h)$, is $h$ with the variable $D$ added as the last argument.
- $PITA_b(b_j)$, is $b_j$ with the variable $D_j$ added as the last argument.
- $b_j = \neg a_j$, $PITA_b(b_j) = (PITA'_b(a_j) \rightarrow \text{not}(DN_j, D_j); \text{one}(D_j))$, where $PITA'_b(a_j)$ is $a_j$ with the variable $DN_j$ added as the last argument.
The disjunctive clause \( C_r = h_1 : \Pi_1 \lor \ldots \lor h_n : \Pi_n \leftarrow b_1, \ldots, b_m \)
where the parameters sum to 1, is transformed into the set of clauses \( PITA(C_r) \):

\[
PITA(C_r, i) = PITA_h(h_i) \leftarrow one(DD_0),
\]

\[
PITA_b(b_1), \text{and}(DD_0, D_1, DD_1), \ldots,
\]

\[
PITA_b(b_m), \text{and}(DD_{m-1}, D_m, DD_m),
\]

\[
\text{get\_var\_n}(r, VC, [\Pi_1, \ldots, \Pi_n], Var),
\]

\[
\text{equality}(Var, i, \Pi_i, DD), \text{and}(DD_m, DD, D).
\]

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) ← bdd_one(D).`
- `not(A, B) ← bdd_not(A, B).`
- `and(A, B, C) ← bdd_and(A, B, C).`
- `equality(V, I, _P, D) ← bdd_equality(V, I, D).`
PITA uses tabling and 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.

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.

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 $or(d_1, d_2, d_3)$. 
PRISM makes the following modeling assumptions [Sato et al., 2010]:

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),

PRISM [Sato and Kameya, 1997] and PITA(IND,EXC) [Riguzzi and Swift, 2011] 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 $A \ast B$*.
- **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

\[
\text{or}(A, B, P) \leftarrow P \text{ is } A + B - A \ast B.
\]

We call PITA(IND,IND) the resulting system.
An example of a program satisfying assumptions 1 and 3 is the following:

- `path(Node, Node).`
- `path(Source, Target) : 0.3 ← edge(Source, Node), path(Node, Target).`
- `edge(0, 1) : 0.3.`

... depending on the graph
Graphs

Lanes (assumptions 1 and 3)

Branches (assumptions 1 and 3)

Parachutes (assumption 1)
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 are couples \((P, T)\) where \(P\) is a real number representing a probability and \(T\) is a term representing a set of explanation as a Boolean formula.

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

term((0, zero)).

term((1, one)).

or($(PA, TA), (PB, TB), (PC, or(TA, TB))) ← ind(TA, TB), !,

PC is $PA + PB − PA * PB$.

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

or($(._PA, TA), (._PB, TB), (PC, TC)) ← ev(TA, TA1), ev(TB, TB1),

bdd_or(TA1, TB1, TC), ret_prob(TC, PC).

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

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

and($(._PA, TA), (._PB, TB), (PC, TC)) ← ev(TA, TA1), ev(TB, TB1),

bdd_and(TA1, TB1, TC), ret_prob(TC, PC).

where ev/2 evaluates a term returning a BDD.
**PITA**(OPT)

\[
\text{exc}(A, B) \leftarrow \text{integer}(A), !, \text{ev}(B, BB), \text{bdd}_\text{and}(A, BB, C), \text{zero}(Z), Z = C.
\]

\[
\text{exc}(A, B) \leftarrow \text{integer}(B), !, \text{ev}(A, AB), \text{bdd}_\text{and}(AB, B, C), \text{zero}(Z), Z = C.
\]

\[
\text{exc}(\text{zero}, _) \leftarrow !.
\]

\[
\text{exc}(_, \text{zero}) \leftarrow !.
\]

\[
\text{exc}(c(V, N), c(V, N1)) \leftarrow !, N \setminus = N1.
\]

\[
\text{exc}(c(V, N), \text{or}(X, Y)) \leftarrow !, \text{exc}(c(V, N), X), \text{exc}(c(V, N), Y).
\]

\[
\text{exc}(c(V, N), \text{and}(X, Y)) \leftarrow !, (\text{exc}(c(V, N), X); \text{exc}(c(V, N), Y)).
\]

\[
\text{exc}(\text{or}(A, B), \text{or}(X, Y)) \leftarrow !, \text{exc}(A, X), \text{exc}(A, Y), \text{exc}(B, X), \text{exc}(B, Y).
\]

\[
\text{exc}(\text{or}(A, B), \text{and}(X, Y)) \leftarrow !, (\text{exc}(A, X); \text{exc}(A, Y)), (\text{exc}(B, X); \text{exc}(B, Y)).
\]

\[
\text{exc}(\text{and}(A, B), \text{and}(X, Y)) \leftarrow !, \text{exc}(A, X); \text{exc}(A, Y); \text{exc}(B, X); \text{exc}(B, Y).
\]

\[
\text{exc}(\text{and}(A, B), \text{or}(X, Y)) \leftarrow !, (\text{exc}(A, X); \text{exc}(B, X)), (\text{exc}(A, Y); \text{exc}(B, Y)).
\]

\[
\text{exc}(\text{not}(A), A) \leftarrow !.
\]

\[
\text{exc}(\text{not}(A), \text{and}(X, Y)) \leftarrow !, \text{exc}(\text{not}(A), X); \text{exc}(\text{not}(A), Y).
\]

\[
\text{exc}(\text{not}(A), \text{or}(X, Y)) \leftarrow !, \text{exc}(\text{not}(A), X), \text{exc}(\text{not}(A), Y).
\]

\[
\text{exc}(A, \text{or}(X, Y)) \leftarrow !, \text{exc}(A, X), \text{exc}(A, Y).
\]

\[
\text{exc}(A, \text{and}(X, Y)) \leftarrow \text{exc}(A, X); \text{exc}(A, Y).
\]
**PITA(OPT)**

\[
\begin{align*}
\text{ind}(\text{one}, \_ ) & \leftarrow !. \\
\text{ind}(\text{zero}, \_ ) & \leftarrow !. \\
\text{ind}(\_, \text{one} ) & \leftarrow !. \\
\text{ind}(\_, \text{zero} ) & \leftarrow !. \\
\text{ind}(A, B) & \leftarrow \text{integer}(A), !, \text{ev}(B, BB), \text{bdd\_ind}(A, BB, I), I = 1. \\
\text{ind}(A, B) & \leftarrow \text{integer}(B), !, \text{ev}(A, AB), \text{bdd\_ind}(AB, B, I), I = 1. \\
\text{ind}(c(V, \_, N), B) & \leftarrow !, \text{absent}(V, B). \\
\text{ind}(\text{or}(X, Y), B) & \leftarrow !, \text{ind}(X, B), \text{ind}(Y, B). \\
\text{ind}(\text{and}(X, Y), B) & \leftarrow !, \text{ind}(X, B), \text{ind}(Y, B). \\
\text{ind}(\text{not}(A), B) & \leftarrow \text{ind}(A, B). \\
\text{absent}(V, c(V1, \_, N1)) & \leftarrow !, V \setminus = V1. \\
\text{absent}(V, \text{or}(X, Y)) & \leftarrow !, \text{absent}(V, X), \text{absent}(V, Y). \\
\text{absent}(V, \text{and}(X, Y)) & \leftarrow !, \text{absent}(V, X), \text{absent}(V, Y). \\
\text{absent}(V, \text{not}(A)) & \leftarrow \text{absent}(V, A).
\end{align*}
\]

\text{bdd\_ind}(B1, B2, I) is implemented in C and 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.
\[
ev(B, B) \leftarrow \text{integer}(B), !. \\
ev(\text{zero}, B) \leftarrow !, \text{bdd_zero}(B). \\
ev(\text{one}, B) \leftarrow !, \text{bdd_one}(B). \\
ev(c(V, N), B) \leftarrow !, \text{bdd}_{-}\text{equality}(V, N, B). \\
ev(\text{and}(A, B), C) \leftarrow !, \ev(A, BA), \ev(B, BB), \text{bdd}_{-}\text{and}(BA, BB, C). \\
ev(\text{or}(A, B), C) \leftarrow !, \ev(A, BA), \ev(B, BB), \text{bdd}_{-}\text{or}(BA, BB, C). \\
ev(\text{not}(A), C) \leftarrow \ev(A), \text{bdd}_{-}\text{not}(A, C).
\]
Experiments on graphs

- We compare PITA, PITA(IND,EXC), PITA(IND,IND), PITA(OPT), PRISM and ProbLog on a number of datasets.
- We first consider the above graphs and path program.
Experiments on the datasets of [Meert et al., 2009]

**Bloodtype (assumptions 1 and 2)**

**Growing head (assumption 1)**
Experiments on the datasets of [Meert et al., 2009]

Growing negated body (assumption 2)

UWCSE (assumption 2)
Conclusions

- 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
  - it gives very good results in datasets where these assumptions hold for the whole program or for parts of it,
  - it pays a limited penalty on datasets where these assumptions are completely false.

**Future works:** investigate the optimization techniques of [Van den Broeck et al., 2011, Fierens et al., 2011]

Thank you!

Questions?


References II

CP-Logic theory inference with contextual variable elimination and comparison to BDD based inference methods.

Logic programming, abduction and probability - a top-down anytime algorithm for estimating prior and posterior probabilities.

Poole, D. (1997).
The Independent Choice Logic for modelling multiple agents under uncertainty.


