Probabilistic Logic Programs

Terrance Swift

Based on joint work with Fabrizio Riguzzi
Example (from [Riguzzi and Swift, 2012])

The LPAD Syntax:

\[
\text{strong_sneezing}(X) : 0.3 \lor \text{mild_sneezing}(X) : 0.5 \leftarrow \text{flu}(X).
\]
\[
\text{strong_sneezing}(X) : 0.2 \lor \text{mild_sneezing}(X) : 0.6 \leftarrow \text{hay_fever}(X).
\]
\[
\text{flu}(\text{david}).
\]
\[
\text{hay_fever}(\text{david}).
\]

The query moderate sneezing(david) is true in 5 of the 9 worlds of the program and its probability of being true is

\[
P_T(\text{moderate_sneezing}(\text{david})) = 0.5 \times 0.2 + 0.5 \times 0.6 + 0.5 \times 0.2 + 0.3 \times 0.6 + 0.2 \times 0.6 = 0.8
\]
Example (from [Kimmig et al., 2011])

\[
\text{path}(F,T) :\quad \text{e}(F,T).
\]

\[
\text{path}(F,T) :\quad \text{path}(F,M), \text{e}(M,T).
\]

\[
\text{e}(a,c) : 0.8 ; \text{null} : 0.2.
\]

\[
\text{e}(a,b) : 0.7 ; \text{null} : 0.3.
\]

\[
\text{e}(c,d) : 0.8 ; \text{null} : 0.2.
\]

\[
\text{e}(c,e) : 0.9 ; \text{null} : 0.1.
\]

\[
\text{e}(b,c) : 0.6 \quad \text{null} 0.4.
\]

\[
\text{e}(d,e) : 0.5 \quad \text{null} 0.5.
\]
Example

- Probability of $\text{path}(c, e)$ is probability of worlds with $e(c, e)$ plus probability of worlds with $e(c, d)$ and $e(d, e)$ but not $e(c, e)$.
  - We don’t want to count $e(c, e)$ twice.

- Probability is $0.9 + (0.1 \times 0.8 \times 0.5) = 0.94$
Recall the definition of an explanation

Given a query $Q$, a composite choice $\kappa$ is an explanation for $Q$ if

$$\forall w \in \omega_{\kappa} \; w \models Q$$

Want a finite set of finite explanations that is covering

The set containing $\{e(a, c), e(c, e)\}$, $\{e(c, e)\}$, $\{e(c, d), e(d, e)\}$ and $\{e(c, e), e(c, d), e(d, e)\}$ is such a covering set for the above example.

It is not mutually exclusive so the probability cannot be directly computed from this set.

The set containing $\{e(a, c), e(c, e)\}$, $\neg e(a, c), e(c, e)\}$, and $\{e(c, d), e(d, e)\}$ is mutually exclusive.

So is $\{\{e(c, e)\}, \{e(c, d), e(d, e)\}\}$
Computing the Full Distribution Semantics

Computing the probability of a query

- Each proof is associated with an explanation which can be seen as a conjunction.
- By determining all proofs of a query we determine all explanations of the query / all worlds where the query is true in DNF form.
- We still need to compute the probability of this DNF formula, which is NP-hard.
- Inclusion/exclusion principle does not scale: it constructs a polynomial exponential in the number of proofs.
- Computing the disjoint sum of the proofs is better but not good: it is $\#P$-complete [Valiant, 1979].
SAT is the core NP-complete proble: does a given boolean formula $\phi$ have solution – a truth assignment that makes $\phi$ true?

F-SAT is the core FNP-complete proble: produce a solution for $\phi$.

#SAT is the problem of determining the number of solutions for $\phi$.

#P is the problem of determining the number of solutions for any polynomially-balanced, polynomial-time (decidable) relation.

#SAT is #P-complete. A #P complete oracle can count the number of solutions to any problem in NP and P.

Determining whether a bipartite graph has a matching can be done in P, but finding the number of matchings is #P-complete [Valiant, 1979].

In fact

$$\#P = \#\text{PSPACE}$$

so its really bad.
Computing the Full Distribution Semantics

Computing the probability of a query

The state of the art is to use BDDs (first used in [Kimmig et al., 2011])

- Associate each disjunct with a boolean random variable that takes the values of the disjunct.
- Maintain these boolean random variables in a BDD
- BDD represents mutually exclusive explanations as paths from the root to 1 (true)
- By traversing all paths from start to 1 a mutually exclusive covering set of explanations is obtained; the probabilities of the explanations can be computed and summed.
Solid lines from a node \( n \) represent a transition taken if \( n \) is true; dashed lines represent the transition if \( n \) is false.

Here there are two paths to 1: \( ce \lor (\neg ce \land cd \land de) \) which correspond to \( 0.9 + 0.04 = 0.94 \).
Computing the probability of a query

- Both PITA and Problog use BDDs to compute the probability of sets of explanations, but do this somewhat differently
  - Problog stores the explanations in a trie, then copies them to a BDD
  - PITA stores them directly in a BDD using tabling
- Both systems use the Cudd package
- For many large queries, the vast majority of the time for both systems is spent in BDD manipulation.
Prior to evaluation, PITA uses a transformation: [Riguzzi and Swift, 2012] Conceptually, disunctive clauses are translated to normal clauses.

\[
\begin{align*}
\text{strong_sneezing}(X) & \leftarrow \text{flu}(X). \\
\text{mild_sneezing}(X) & \leftarrow \text{flu}(X). \\
\text{strong_sneezing}(X) & \leftarrow \text{hay_fever}(X). \\
\text{mild_sneezing}(X) & \leftarrow \text{hay_fever}(X). \\
\text{flu}(david). \\
\text{hay_fever}(david).
\end{align*}
\]

For each such clause, BDD API goals are added:

\[
\begin{align*}
\text{strong_sneezing}(X, \text{BDD}) & \leftarrow \text{one}(\text{BB}_0), \text{flu}(X, \text{B}_1), \text{and}(\text{BB}_0, \text{B}_1, \text{BB}_1), \\
& \text{get_var_n}(1, [X], [0.3, 0.5, 0.2], \text{Var}), \\
& \text{equality}(\text{Var}, 1, \text{B}), \text{and}(\text{BB}_1, \text{B}, \text{BDD}). \\
\text{mild_sneezing}(X, \text{BDD}) & \leftarrow \text{one}(\text{BB}_0), \text{flu}(X, \text{B}_1), \text{and}(\text{BB}_0, \text{B}_1, \text{BB}_1), \\
& \text{get_var_n}(1, [X], [0.3, 0.5, 0.2], \text{Var}), \\
& \text{equality}(\text{Var}, 2, \text{B}), \text{and}(\text{BB}_1, \text{B}, \text{BDD}).
\end{align*}
\]
These API goals do the following

- **init, end**: for allocation and deallocation of a BDD manager, a data structure used to keep track of the memory for storing BDD nodes;
- **zero(-B), one(-B), and(+B1, +B2, -B), or(+B1, +B2, -B), not(+B1, -B)**: Boolean operations between BDDs;

`get_var_n/4` dynamically associates a boolean variable with an atomic choice.

\[
\text{get\_var\_n}(R, S, \text{Probs, Var}) \leftarrow \\
(\text{var}(R, S, \text{Var}) \rightarrow \text{true}; \\
\text{length(Probs, L), add\_var(L, Probs, Var), assert(var(R, S, Var))}).
\]
PITA and Tabling

- Pointers to the Cudd BDD are tabled.
- Answer subsumption is used to combine the results of different derivations. Answer subsumption calls `or(+B1, +B2, -B)` which calls Cudd.
Two approximations

- **$K$-best** Keep the $K$ best explanations, not worrying about whether they are exclusive or not. At the end of the (sub-) evaluation, compute their probability. This puts a bound on the number of conjuncts in the DNF formula.

- **Epsilon** Keep combining probabilities until the $n^{th}$ probability minus the $(n - 1)^{th}$ probability is less than epsilon.

These approximations are available in Problog, and are under development in PITA (using completion conditions).
Two assumptions can drastically reduce the computational complexity:

- **Independence**: the probability of a conjunction $(A, B)$ is computed as the product of the probabilities of $A$ and $B$
- **Exclusion**: the probability of a disjunction $(A; B)$ is computed as the sum of the probabilities of $A$ and $B$

These assumptions are made by the PRISM system [Sato et al., 2010] (though not the PRISM language [Sato and Kameya, 1997]) and by PITA(IND/EXC) [Riguzzi and Swift, 2011]
The Independence Assumption

- The program

\[ p \leftarrow a, b. \quad a : 0.3 \lor b : 0.4. \]

does not satisfy the independence assumption because the conjunction \( a, b \) has probability 0, since \( a \) and \( b \) are never true in the same world (so \( p \) would always fail).

- If given to Prism or to PITA(IND/EXC) the probability returned would be 0.12 for reasons discussed below.

- Similarly

\[ q \leftarrow a, b. \quad a \leftarrow c. \quad b \leftarrow c. \quad c : 0.2. \]

does not satisfy the independence assumption because \( a \) and \( b \) both depend on \( c \). In the distribution semantics the probability is 0.2, but if an independence assumption is made the probability is 0.04.
The Exclusion Assumption

The program

\[ q \leftarrow a. \quad q \leftarrow a, b. \quad a : 0.2. \quad b : 0.4. \]

violates the exclusiveness assumption as the two clauses for the ground atom \( q \) have non-exclusive bodies.
Consider some sample programs:

- *Pea plants*, *Coin* and *Blood type* satisfy both independence and exclusion
- Hidden Markov models may satisfy both assumptions
- *Russian roulette* satisfies independence
- *Dice* satisfies exclusion
- *Path*, UWCSE (a program to allocate teaching and advising loads) do not satisfy either assumption
Examples that satisfy independence and exclusion

Mendeleean inheritance rules for pea plants

\[
\begin{align*}
color(X, \text{white}) : & \neg cg(X, 1, w), cg(X, 2, w). \\
color(X, \text{purple}) : & \neg cg(X, _A, p).
\end{align*}
\]

\[
\begin{align*}
cg(X, 1, A) & : 0.5 ; cg(X, 1, B) : 0.5 : - \\
& \quad \text{mother}(Y, X), cg(Y, 1, A), cg(Y, 2, B).
\end{align*}
\]

\[
\begin{align*}
cg(X, 2, A) & : 0.5 ; cg(X, 2, B) : 0.5 : - \\
& \quad \text{father}(Y, X), cg(Y, 1, A), cg(Y, 2, B).
\end{align*}
\]

The \textit{coin} program

\[
\begin{align*}
\text{heads}(Coin) & : 1/2 ; \text{tails}(Coin) : 1/2 : - \\
& \quad \text{toss}(Coin), \neg \text{biased}(Coin).
\end{align*}
\]

\[
\begin{align*}
\text{heads}(Coin) & : 0.6 ; \text{tails}(Coin) : 0.4 : - \\
& \quad \text{toss}(Coin), \text{biased}(Coin).
\end{align*}
\]

\[
\begin{align*}
\text{fair}(Coin) & : 0.9 ; \text{biased}(Coin) : 0.1. \\
\text{toss}(coin).
\end{align*}
\]
Examples that satisfy independence and exclusion

A Hidden Markov Model

\[
\begin{align*}
X(t-1) & \longrightarrow X(t) \longrightarrow X(t+1) \\
Y(t-1) & \longrightarrow Y(t) \longrightarrow Y(t+1)
\end{align*}
\]

\[
\text{hmm}(S, O) :- \text{hmm}(q1, [], S, O). \\
\text{hmm}(\text{end}, S, S, []). \\
\text{hmm}(Q, S0, S, [L|O]) :- \\
\quad Q \neq \text{end}, \ \text{next_state}(Q, Q1, S0), \\
\quad \text{letter}(Q, L, S0), \ \text{hmm}(Q1, [Q|S0], S, O). \\
\text{next_state}(q1, q1, _S) : 1/3; \text{next_state}(q1, q2_, _S) : 1/3; \\
\quad \text{next_state}(q1, \text{end}, _S) : 1/3. \\
\text{next_state}(q2, q1, _S) : 1/3; \text{next_state}(q2, q2_, _S) : 1/3; \\
\quad \text{next_state}(q2, \text{end}, _S) : 1/3. \\
\text{letter}(q1, a, _S) : 0.25; \text{letter}(q1, c, _S) : 0.25; \\
\quad \text{letter}(q1, g, _S) : 0.25; \text{letter}(q1, t, _S) : 0.25. \\
\text{letter}(q2, a, _S) : 0.25; \text{letter}(q2, c, _S) : 0.25; \\
\quad \text{letter}(q2, g, _S) : 0.25; \text{letter}(q2, t, _S) : 0.25.
\]
Examples that satisfy independence and exclusion

The Blood Type Program

```prolog
bloodtype(P,a):0.90 ; bloodtype(P,b):0.03 ; bloodtype(P,ab):0.03 ; bloodtype(P,null):0.04 :-
pchrom(P,a ),mchrom(P,a ).
bloodtype(P,a):0.03 ; bloodtype(P,b):0.03 ; bloodtype(P,ab):0.90 ; bloodtype(P,null):0.04 :-
pchrom(P,b ),mchrom(P,a ).
bloodtype(P,a):0.90 ; bloodtype(P,b):0.04 ; bloodtype(P,ab):0.03 ; bloodtype(P,null):0.03 :-
pchrom(P,null),mchrom(P,a ).
bloodtype(P,a):0.03 ; bloodtype(P,b):0.03 ; bloodtype(P,ab):0.90 ; bloodtype(P,null):0.04 :-
pchrom(P,a ),mchrom(P,b ).
bloodtype(P,a):0.04 ; bloodtype(P,b):0.90 ; bloodtype(P,ab):0.03 ; bloodtype(P,null):0.03 :-
pchrom(P,b ),mchrom(P,b ).
bloodtype(P,a):0.03 ; bloodtype(P,b):0.09 ; bloodtype(P,ab):0.04 ; bloodtype(P,null):0.03 :-
pchrom(P,null),mchrom(P,b ).
bloodtype(P,a):0.90 ; bloodtype(P,b):0.03 ; bloodtype(P,ab):0.03 ; bloodtype(P,null):0.04 :-
pchrom(P,a ),mchrom(P,null).
bloodtype(P,a):0.03 ; bloodtype(P,b):0.90 ; bloodtype(P,ab):0.04 ; bloodtype(P,null):0.03 :-
pchrom(P,b ),mchrom(P,null).
bloodtype(P,a):0.03 ; bloodtype(P,b):0.04 ; bloodtype(P,ab):0.03 ; bloodtype(P,null):0.90 :-
pchrom(P,null),mchrom(P,null).
```

Terrance Swift ()
Probabilistic Logic Programs Complexity, Imp
An example that satisfies independence but not exclusion

Russian roulette with two guns

death:1/6 :- pull_trigger(left_gun).
death:1/6 :- pull_trigger(right_gun).
pull_trigger(left_gun).
pull_trigger(right_gun).
Assumptions of Exclusion and Independence

An example that satisfies exclusion but not independence

Game of dice (keep rolling until you get a 3)

\[
\begin{align*}
on(0,1) & : 1/3 ; \ on(0,2) : 1/3 ; \ on(0,3) : 1/3. \\
on(T,1) & : 1/3 ; \ on(T,2) : 1/3 ; \ on(T,3) : 1/3 :- \\
& \quad \text{T1 is T-1, T1} \geq 0, \ on(T1,F), \ \neg \ on(T1,3). 
\end{align*}
\]
An examples that satisfy neither independence nor exclusion

Probability of paths

\[\text{path}(X, X),\]
\[\text{path}(X, Y) : \neg \text{path}(X, Z), \text{edge}(Z, Y),\]
\[\text{edge}(a, b) : 0.3,\]
\[\text{edge}(b, c) : 0.2,\]
\[\text{edge}(a, c) : 0.6.\]
Assumptions of Exclusion and Independence

A source-transformation

Removal of non-discriminating arguments [Christiansen and Gallagher, 2009] can be particularly for some PLPs. As an example

\[
\text{hmm}(S,O) :- \text{hmm}(q1, [], S, O).
\]
\[
\text{hmm}(\text{end}, S, S, []). 
\]
\[
\text{hmm}(Q, S0, S, [L|O]) :-
\]
\[
Q \neq \text{end}, \quad \text{next\_state}(Q, Q1, S0),
\]
\[
\text{letter}(Q, L, S0), \quad \text{hmm}(Q1, [Q|S0], S, O).
\]

becomes

\[
\text{hmm}(O) :- \text{hmm}(q1, O).
\]
\[
\text{hmm}(\text{end}, []). 
\]
\[
\text{hmm}(Q, [L|O]) :-
\]
\[
Q \neq \text{end}, \quad \text{next\_state}(Q, Q1, S0),
\]
\[
\text{letter}(Q, L, S0), \text{hmm}(Q1, O).
\]

This reduces the complexity of the tabled program and a speedup of several orders of magnitude [Riguzzi and Swift, 2011].
PITA implementation

- PITA(Ind/Exc) uses the same transformation as in the general case:

\[
\begin{align*}
\text{strong_sneeze}(X, \text{BDD}) & \leftarrow \text{one}(\text{BB}_0), \text{flu}(X, \text{B}_1), \text{and}(\text{BB}_0, \text{B}_1, \text{BB}_1), \\
\text{mild_sneeze}(X, \text{BDD}) & \leftarrow \text{one}(\text{BB}_0), \text{flu}(X, \text{B}_1), \text{and}(\text{BB}_0, \text{B}_1, \text{BB}_1), \\
& \text{get_var}_n(1, [X], [0.3, 0.5, 0.2], \text{Var}), \\
& \text{equality}(\text{Var}, 1, \text{B}), \text{and}(\text{BB}_1, \text{B}, \text{BDD}).
\end{align*}
\]

- Answer subsumption is still used.
- But now \texttt{and/3}, \texttt{or/3} etc are implemented by multiplication, addition and other simple operations: no interfaces to BDDs.
Further Approaches

PITA(IND,IND)

The Exclusion assumption is a strong assumption that could be replaced by:

- **Clause Independence**: The probability of a disjunction \((A; B)\) is computed as if \(A\) and \(B\) were independent.

- Clause Independence allows one to combine the probability of two explanations \(A\) and \(B\) as

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

- This has been implemented as PITA(Ind,Ind) [Riguzzi, 2012].
These graphs satisfy conjunct independence since they have no cycles – the same edge fact cannot be used more than once to prove a path.

Lanes
Further Approaches

PITA(OPT)

- But maybe you don’t know whether clause independence or exclusion holds, or maybe it holds only for some predicates.
- The approach of PITA(OPT) is to dynamically check for independence and exclusion and only use BDDs if these assumptions don’t hold.
- Explanations are kept in a mixture of Prolog terms and BDDs.
Further Approaches

PITA(OPT)

\[ \text{or}(((PA, TA), (PB, TB), (PC, \text{or}(TA, TB)))) \leftarrow \text{ind}(TA, TB),!,\]
\[ PC \text{ is } PA + PB - PA \ast PB. \]
\[ \text{exc}(TA, TB),!, PC \text{ is } PA + PB. \]
\[ \text{ev}(TA, TA1), \text{ev}(TB, TB1),\]
\[ \text{bdd}_\text{or}(TA1, TB1, TC), \text{ret}_\text{prob}(TC, PC). \]

\[ \text{and}(((PA, TA), (PB, TB), (PC, \text{and}(TA, TB)))) \leftarrow \text{ind}(TA, TB),!, PC \text{ is } PA \ast PB. \]
\[ \text{exc}(TA, TB),!, \text{fail}. \]
\[ \text{ev}(TA, TA1), \text{ev}(TB, TB1),\]
\[ \text{bdd}_\text{and}(TA1, TB1, TC), \text{ret}_\text{prob}(TC, PC). \]

- \text{ev}(\text{Term}, \text{BDD}) \text{ produces a BDD corresponding to a boolean term.} \\
- \text{exc/2, ind/2 check the independence of Prolog terms, and call BDD independence and exclusion routines.}
All version of PITA use the same transformation: they differ in predicates such as `and/3`, `or/3` etc.

Full PITA must build and maintain BDDs for all explanations.

PITA(OPT) keeps explanations as Prolog terms, and interns as BDDs only as needed.

Note that PITA(IND,IND) and PITA(IND,EXC) do not need to build up explanations: they just keep track of probabilities (as floats).

The findings of [Riguzzi, 2012] are:

- If it is known whether the independence or exclusion assumptions hold, PITA(IND,EXC) or PITA(IND,IND) is best.
- PITA(OPT) is always at least as scalable as full PITA or Problog, but is slightly less scalable than PITA(IND,EXC) or Prism.
PITA(Poss)

- PITA(Poss) uses the same transformation as in the general case:

\[
\text{strong_sneezing}(X, \text{BDD}) \leftarrow \text{one}(\text{BB}_0), \text{flu}(X, B_1), \text{and}(\text{BB}_0, B_1, \text{BB}_1), \\
\text{get}_\text{var}_n(1, [X], [0.3, 0.5, 0.2], \text{Var}), \\
\text{equality}(\text{Var}, 1, B), \text{and}(\text{BB}_1, B, \text{BDD}).
\]

\[
\text{mild_sneezing}(X, \text{BDD}) \leftarrow \text{one}(\text{BB}_0), \text{flu}(X, B_1), \text{and}(\text{BB}_0, B_1, \text{BB}_1), \\
\text{get}_\text{var}_n(1, [X], [0.3, 0.5, 0.2], \text{Var}), \\
\text{equality}(\text{Var}, 2, B), \text{and}(\text{BB}_1, B, \text{BDD}).
\]

- Answer subsumption is still used.

- But now \textit{and/3}, \textit{or/3} etc are implemented by \text{min}, \text{max} and other operations.

- From a tabling point of view, this implementation is very similar to that of PITA(IND/EXC).
Implementation Status

- Implementation of full distribution semantics is currently available only on Linux (uncertain of Cudd portability)
- Answer subsumption works\(^1\) with
  - stratified programs
  - call abstraction
  - attributed variables (delay & constraints)
  - multi-threading
  - completion condition (under development)
- It does not currently work with
  - full WFS with conditional answers
  - incremental tabling
  - call subsumption

... however incremental tabling itself does not work with uncertain truth values in WFS... (enhancement request 451)

\(^1\)EPA-estimated compatibility, actual mileage may vary.
References I


In *International Joint Conference on Artificial Intelligence,* pages 1330–1339.