Statistical Relational AI

Andrea Passerini
andrea.passerini@unitn.it

Advanced Topics in Machine Learning and Optimization

Combining logic with probability

Motivation

- First-order logic is a powerful language to represent complex relational information
- Probability is the standard way to represent uncertainty in knowledge
- Combining the two would allow to model complex probabilistic relationships in the domain of interest

Combining logic with probability

Problem of uncertainty

- In most real world scenarios, logic formulas are *typically* but not *always* true
- For instance:
  - “Every bird flies” : what about an ostrich (or Charlie Parker) ?
  - “Every predator of a bird is a bird”: what about lions with ostriches (or heroin with Parker) ?
  - “Every prey has a predator”: predators can be extinct
- A world failing to satisfy even a single formula would not be possible
- there could be no possible world satisfying all formulas

Combining logic with probability

Handling uncertainty

- We can relax the hard constraint assumption on satisfying all formulas
- A possible world not satisfying a certain formula will simply be less likely
- The more formula a possible world satisfies, the more likely it is
- Each formula can have a weight indicating how strong a constraint it should be for possible worlds
- Higher weight indicates higher probability of a world satisfying the formula wrt one not satisfying it

Example: probabilistic relational robotics

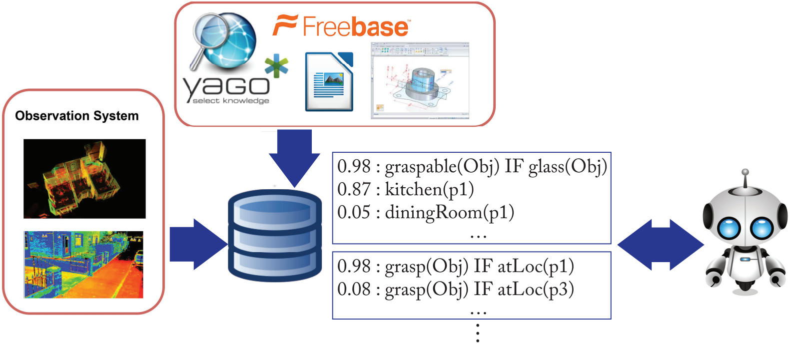


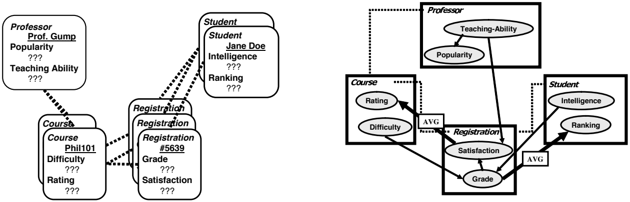
Image from De Raedt et al., 2016

Combining logic with probability

logic graphical models

- Graphical models are a mean to represent joint probabilities highlighting the relational structure among variables
- A compressed representation of such models can be obtained using templates, cliques in the graphs sharing common parameters (e.g. as in HMM for BN or CRF for MN)
- Logic can be seen as a language to build templates for graphical models
- Logic based versions of HMM, BN and MN have been defined

Relational Probabilistic Models (RPM)



- Extend probabilistic models to include relations
- Exploit *exchangeability*: all individuals with the same properties are treated the same
- Template-based models: nodes contain random variables, model can be grounded over instantiation of variables (individuals)
- Alphabetic soup of relational probabilistic models

Image from Getoor et al., 2007

Representative Relational Probabilistic Models (RPM)

Markov Logic

- First-order logic version of Markov Networks
- Undirected RPM
- Extend probabilistic graphical model with logic

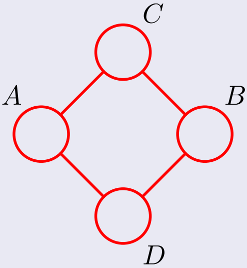
ProbLog

- Probabilistic version of the Prolog language
- Directed RPM
- Extend logic programming language with probabilities

Markov Networks (MN) or Markov random fields

Undirected graphical models

- Graphically model the joint probability of a set of variables encoding independency assumptions (as BN)
- Do not assign a directionality to pairwise interactions

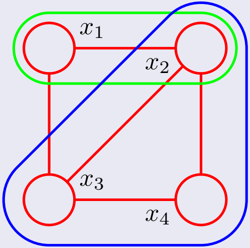


- Do not assume that local interactions are proper conditional probabilities (more freedom in parametrizing them)

Markov Networks (MN)

Factorization properties

- We need to decompose the joint probability in *factors* (like $P(x_i|pa_i)$ in BN) in a way consistent with the independency assumptions.
- Two nodes x_i and x_j not connected by a link are independent given the rest of the network (any path between them contains at least another node) $\Rightarrow$ they should stay in separate factors
- Factors are associated with *cliques* in the network (i.e. fully connected subnetworks)
- A possible option is to associate factors only with *maximal* cliques



Markov Networks (MN)

Joint probability (1)

$$p(\boldsymbol{x}) = \frac{1}{Z} \prod_C \Psi_C(\boldsymbol{x}_C)$$

- $\Psi_C(\boldsymbol{x}_C) : Val(\boldsymbol{x}_c) \rightarrow \mathbb{R}^+$ is a positive function of the value of the variables in clique $\boldsymbol{x}_C$ (called clique *potential*)
- The joint probability is the (normalized) product of clique potentials for all (maximal) cliques in the network
- The *partition function* Z is a normalization constant assuring that the result is a probability (i.e. $\sum_{\boldsymbol{x}} p(\boldsymbol{x}) = 1$):

$$Z = \sum_{\boldsymbol{x}} \prod_C \Psi_C(\boldsymbol{x}_C)$$

Joint probability (2)

- In order to guarantee that potential functions are positive, they are typically represented as exponentials:

$$\Psi_C(\boldsymbol{x}_C) = \exp\left(-E(\boldsymbol{x}_c)\right)$$

- $E(\boldsymbol{x}_c)$ is called *energy* function: low energy means highly probable configuration
- The joint probability becames a sum of exponentials:

$$p(\boldsymbol{x}) = \frac{1}{Z} \exp\left(-\sum_C E(\boldsymbol{x}_c)\right)$$

Markov Networks (MN)

Comparison to BN

- Advantage:
  - More freedom in defining the clique potentials as they don't need to represent a probability distribution
- Problem:
  - The partition function Z has to be computed summing over all possible values of the variables
- Solutions:
  - Intractable in general
  - Efficient algorithms exists for certain types of models (e.g. linear chains)
  - Otherwise Z is approximated

Maximum likelihood estimation

- For general MN, the likelihood function cannot be decomposed into independent pieces for each potential (as happens for BN) because of the global normalization (Z)

$$\mathcal{L}(E, \mathcal{D}) = \prod_{i=1}^{N} \frac{1}{Z} \exp\left(-\sum_{C} E_c(\boldsymbol{x}_c(i))\right)$$

- However the likelihood is concave in E_C, the energy functionals whose parameters have to be estimated
- The problem is an unconstrained concave maximization problem solved by gradient ascent (or second order methods)

Learning in Markov Networks

Maximum likelihood estimation

- For each configuration $\mathbf{u}_c \in Val(\mathbf{x}_c)$ we have a parameter $E_{c,\mathbf{u}_c} \in \mathbb{R}$ (ignoring parameter tying)
- The partial derivative of the log-likelihood wrt $E_{c,\mathbf{u}_c}$ is:

$$\frac{\partial \log \mathcal{L}(E, \mathcal{D})}{\partial E_{c,\mathbf{u}_c}} = \sum_{i=1}^{N} [\delta(\mathbf{x}_c(i), \mathbf{u}_c) - p(\boldsymbol{X}_c = \mathbf{u}_c | E)]$$
$$= N_{\mathbf{u}_c} - N\, p(\boldsymbol{X}_c = \mathbf{u}_c | E)$$

- The derivative is zero when the counts of the data correspond to the expected counts predicted by the model
- In order to compute $p(\boldsymbol{X}_c = \mathbf{u}_c | E)$, inference has to be performed on the Markov network, making learning quite expensive

Markov Logic networks

Definition

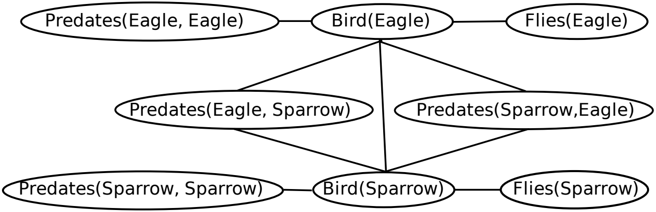
- A Markov Logic Network (MLN) L is a set of pairs (F_i, w_i) where:
  - F_i is a formula in first-order logic
  - w_i is a real number (the weight of the formula)
- Applied to a finite set of constants $C = \{c_1, \dots, c_{|C|}\}$ it defines a Markov network $M_{L,C}$:
  - $M_{L,C}$ has one binary node for each possible grounding of each atom in L. The value of the node is 1 if the ground atom is true, 0 otherwise.
  - $M_{L,C}$ has one feature for each possible grounding of each formula F_i in L. The value of the feature is 1 if the ground formula is true, 0 otherwise. The weight of the feature is the weight w_i of the corresponding formula

Markov Logic networks

Intuition

- A MLN is a *template* for Markov Networks, based on logical descriptions
- Single atoms in the template will generate nodes in the network
- Formulas in the template will be generate cliques in the network
- There is an edge between two nodes iff the corresponding ground atoms appear together in at least one grounding of a formula in L

Markov Logic networks: example



Ground network

- A MLN with two (weighted) formulas:

$$w_1 \quad \forall X \,(\texttt{bird(X)} \Rightarrow \texttt{flies(X)})$$
$$w_2 \quad \forall X, Y \,(\texttt{predates(X,Y)} \land \texttt{bird(Y)} \Rightarrow \texttt{bird(X)})$$

- applied to a set of two constants $\{\texttt{sparrow}, \texttt{eagle}\}$
- generates the Markov Network shown in figure

Markov Logic networks

Joint probability

- A ground MLN specifies a joint probability distribution over possible worlds (i.e. truth value assignments to all ground atoms)
- The probability of a possible world x is:

$$p(x) = \frac{1}{Z} \exp\left(\sum_{i=1}^{F} w_i n_i(x)\right)$$

  where:

  - the sum ranges over formulas in the MLN (i.e. clique templates in the Markov Network)
  - $n_i(x)$ is the number of true groundings of formula F_i in x
  - The partition function Z sums over all possible worlds (i.e. all possible combination of truth assignments to ground atoms)

Markov Logic networks

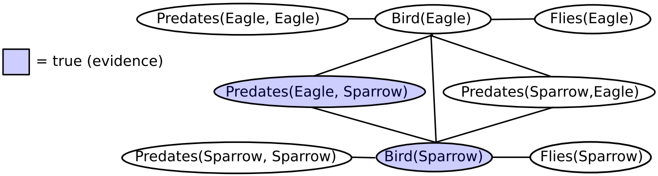
Adding evidence

- Evidence is usually available for a subset of the ground atoms (as their truth value assignment)
- The MLN can be used to compute the conditional probability of a possible world x (consistent with the evidence) given evidence e:

$$p(x|e) = \frac{1}{Z(e)} \exp\left(\sum_{i=1}^{F} w_i n_i(x)\right)$$

- where the partition function $Z(e)$ sums over all possible worlds consistent with the evidence.

Example: evidence

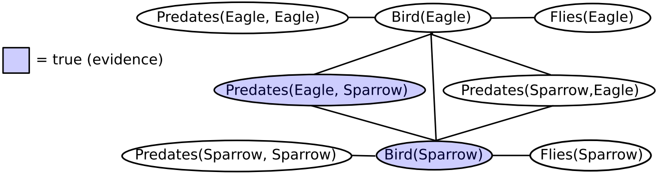


Including evidence

- Suppose that we have (true) evidence *e* given by these two facts:

```
bird(sparrow)
predates(eagle,sparrow)
```

Example: assignment 1



Computing probability

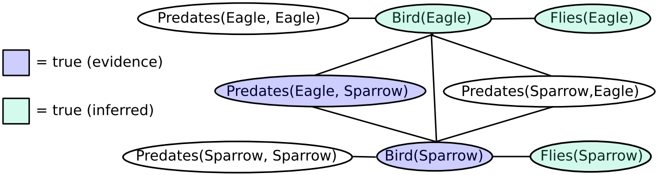
$$p(x) = \frac{1}{Z} \exp(w_1 + 3w_2)$$

- The probability of a world with only evidence atoms set as true violates two ground formulas:

  `bird(sparrow)` $\Rightarrow$ `flies(sparrow)`

  `predates(eagle,sparrow)` $\wedge$ `bird(sparrow)` $\Rightarrow$ `bird(eagle)`

Example: assignment 2

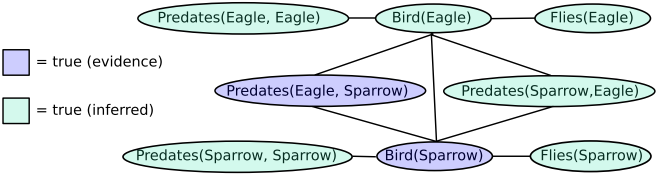


Computing probability

$$p(x) = \frac{1}{Z}\exp(2w_1 + 4w_2)$$

- This possible world is the most likely among all possible worlds as it satisfies all constraints.

Example: assignment 3



Computing probability

$$p(x) = \frac{1}{Z}\exp(2w_1 + 4w_2)$$

- This possible world has also highest probability.
- The problem is that we did not encode constraints saying that:
  - A bird is not likely to be predator of itself
  - A prey is not likely to be predator of its predator

Hard constraints

Impossible worlds

- It is always possible to make certain worlds impossible by adding constraints with infinite weight
- Infinite weight constraints behave like pure logic formulas: any possible world has to satisfy them, otherwise it receives zero probability

Example

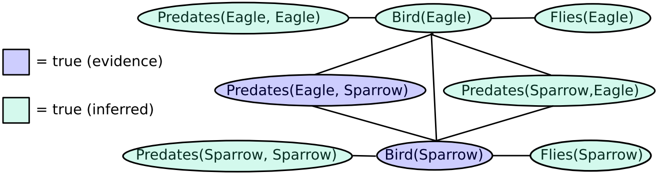
- Let's add the infinite weight constraint:

  "Nobody can be a self-predator"

  $w_3 \quad \forall X \ \neg\texttt{predates(X,X)}$

  to the previous example

Hard constraint: assignment 3

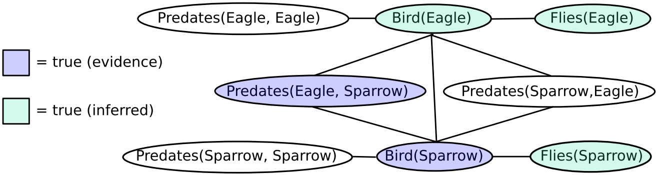


Computing joint probability

$$p(x) = \frac{1}{Z} \exp(2w_1 + 4w_2) = 0$$

- The numerator does not contain w_3, as the no-self-predator constraint is never satified
- However the partition function Z sums over all possible worlds, including those in which the constraint is satisfied.
- As $w_3 = \infty$, the partition function takes infinite value and the possible worlds gets zero probability.

Hard constraint: assignment 2



Computing joint probability

$$p(x) = \frac{1}{Z} \exp(2w_1 + 4w_2 + 2w_3) \neq 0$$

- The only non-zero probability possible worlds are those always satisying hard constraints
- Infinite weight features cancel out between numerator and possible worlds at denominator which also satisfy the constraints, while those which do not become zero

Assumptions

- For simplicity of presentation, we will consider MLN in form:
  - function-free (only predicates)
  - clausal
- However the methods can be applied to other forms as well
- We will use general first-order logic form when describing applications

MPE inference

- One of the basic tasks consists of predicting the most probable state of the world given some evidence (the *most probable explanation*)
- The problem is a special case of MAP inference (*maximum a posteriori* inference), in which we are interested in the state of a subset of variables which do not necessarily include all those without evidence.

Inference

MPE inference in MLN

- MPE inference in MLN reduces to finding the truth assignment for variables (i.e. nodes) without evidence maximizing the weighted sum of satisfied clauses (i.e. features)
- The problem can be addressed with any weighted satisfiability solver
- MaxWalkSAT has been successfully used for MPE inference in MLN.

MaxWalkSAT

Description

- Weighted version of WalkSAT
- Stochastic local search algorithm:
  1. Pick an unsatisfied clause at random
  2. Flip the truth value of an atom in the clause
- The atom to flip is chosen in one of two possible ways with a certain probability:
  - randomly
  - in order to maximize the weighted sum of the clauses satisfied with the flip
- The stochastic behaviour (hopefully) allows to escape local minima

MaxWalkSAT pseudocode

```
procedure MaxWalkSAT(weighted_clauses,max_flips,max_tries,target,p)
    vars ← variables in weighted_clauses
    for i ← 1 to max_tries do
        soln ← a random truth assignment to vars
        cost ← sum of weights of unsatisfied clauses in soln
        for j ← 1 to max_flips do
            if cost ≤ target then
                return “Success, solution is”, soln
            end if
            c ← a randomly chosen unsatisfied clause
            if Uniform(0,1) < p then
                v_f ← a randomly chosen variable from c
            else
                for all variable v in c do
                    compute DeltaCost(v)
                end for
                v_f ← v with lowest DeltaCost(v)
            end if
            soln ← soln with v_f flipped
            cost ← cost+ DeltaCost(v_f)
        end for
    end for
    return “Failure, best assignment is”, best soln found
end procedure
```

MaxWalkSAT

Ingredients

- *target* is the maximum cost considered acceptable for a solution
- *max_tries* is the number of walk restarts
- *max_flips* is the number of flips in a single walk
- *p* is the probability of flipping a random variable
- Uniform(0,1) picks a number uniformly at random from [0,1]
- DeltaCost(*v*) computes the change in cost obtained by flipping variable *v* in the current solution

Inference

Marginal and conditional probabilities

- Another basic inference task is that of computing the marginal probability that a formula holds, possibly given some evidence on the truth value of other formulas
- Exact inference in generic MLN is intractable (as it is for the generic MN obtained by the grounding)
- MCMC sampling techniques have been used as an approximate alternative

Constructing the ground MN

- In order to perform a specific inference task, it is not necessary in general to ground the whole network, as parts of it could have no influence on the computation of the desired probability
- Grounding only the needed part of the network can allow significant savings both in memory and in time to run the inference

Partial grounding: intuition

- A standard inference task is that of computing the probability that F_1 holds given that F_2 does.
- We will focus on the common simple case in which F_1, F_2 are conjunctions of ground literals:
  1. All atoms in F_1 are added to the network one after the other
  2. If an atom is also in F_2 (has evidence), nothing more is needed for it
  3. Otherwise, its Markov blanket is added, and each atom in the blanket is checked in the same way

Partial grounding: pseudocode

procedure ConstructNetwork(F_1,F_2,L,C)
inputs:
F_1 a set of query ground atoms
F_2 a set of evidence ground atoms
L a Markov Logic Network
C a set of constants
output: M a ground Markov Network
calls: $MB(q)$ the Markov blanket of q in $M_{L,C}$
    $G \leftarrow F_1$
    while $F_1 \neq \emptyset$ **do**
        for all $q \in F_1$ **do**
            if $q \notin F_2$ **then**
                $F_1 \leftarrow F_1 \cup (MB(q) \setminus G)$
                $G \leftarrow G \cup MB(q)$
            end if
            $F_1 \leftarrow F_1 \setminus \{q\}$
        end for
    end while
    return M the ground MN composed of all nodes in G and all arcs between them in $M_{L,C}$, with features and weights of the corresponding cliques
end procedure

Inference

Gibbs sampling

- Inference in the partial ground network is done by Gibbs sampling.
- The basic step consists of sampling a ground atom given its Markov blanket
- The probability of X_l given that its Markov blanket has state $\boldsymbol{B}_l = \boldsymbol{b}_l$ is $p(X_l = x_l|\boldsymbol{B}_l = \boldsymbol{b}_l) =$

  $$\frac{\exp \sum_{f_i \in F_l} w_i f_i(X_l = x_l, \boldsymbol{B}_l = \boldsymbol{b}_l)}{\exp \sum_{f_i \in F_l} w_i f_i(X_l = 0, \boldsymbol{B}_l = \boldsymbol{b}_l) + \exp \sum_{f_i \in F_l} w_i f_i(X_l = 1, \boldsymbol{B}_l = \boldsymbol{b}_l)}$$

  where:
  - F_l is the set of ground formulas containing X_l
  - $f_i(X_l = x_l, \boldsymbol{B}_l = \boldsymbol{b}_l)$ is the truth value of the ith formula when $X_l = x_l$ and $\boldsymbol{B}_l = \boldsymbol{b}_l$
- The probability of the conjuction of literals is the fraction of samples (at chain convergence) in which all literals are true

Inference

Multimodal distributions

- As the distribution is likely to have many modes, multiple independently initialized chains are run
- Efficiency in modeling the multimodal distribution can be obtained starting each chain from a mode reached using MaxWalkSAT

Handling hard constraints

- Hard constraints break the space of possible worlds into separate regions
- This violate the MCMC assumption of reachability
- Very strong constraints create areas of very low probability difficult to traverse
- The problem can be addressed by *slice sampling* MCMC, a technique aimed at sampling from slices of the distribution with a frequency proportional to the probability of the slice

Maximum likelihood parameter estimation

- Parameter estimation amounts at learning weights of formulas
- We can learn weights from training examples as possible worlds.
- Let's consider a single possible world as training example, made of:
    - a set of constants $\mathcal{C}$ defining a specific MN from the MLN
    - a truth value for each ground atom in the resulting MN
- We usually make a *closed world* assumption, where we only specify the true ground atoms, while all others are assumed to be false.
- As all groundings of the same formula will share the same weight, learning can be also done on a single possible world

Maximum likelihood parameter estimation

- Weights of formulas can be learned maximizing the likelihood of the possible world:

$$w^{\max} = \mathrm{argmax}_w p_w(x) = \mathrm{argmax}_w \frac{1}{Z} \exp\left(\sum_{i=1}^{F} w_i n_i(x)\right)$$

- As usual we will equivalenty maximize the log-likelihood:

$$\log(p_w(x)) = \sum_{i=1}^{F} w_i n_i(x) - \log(Z)$$

Priors

- In order to combat overfitting Gaussian priors can be added to the weights as usual (see CRF)

Learning

Maximum likelihood parameter estimation

- The gradient of the log-likelihood wrt weights becomes:

$$\frac{\partial}{\partial w_i} \log p_w(x) = n_i(x) - \sum_{x'} p_w(x') n_i(x')$$

  where the sum is over all possible worlds x', i.e. all possible truth assignments for ground atoms in the MN
- Note that $p_w(x')$ is computed using the current parameter values w
- The i-th component of the gradient is the difference between number of true grounding of the i-th formula, and its expectation according to the current model

Logic Programming

As disjunction	As implication	In Prolog
¬ `bird(X)` ∨ `flies(X)`	`bird(X)` ⇒ `flies(X)`	`flies(X) :- bird(X).`
¬ `predates(X,Y)` ∨ ¬ `bird(Y)` ∨ `bird(X)`	`predates(X,Y)` ∧ `bird(Y)` ⇒ `bird(X)`	`bird(X) :- predates(X,Y), bird(Y).`

Horn clauses

- Clauses (disjunctions of literals) with at most one positive literal
- Variables are implicitly universally quantified
- Can be written as implications (the head of the implication is a single atom)
- Amenable to efficient inference by SLD resolution (Prolog programming language)

ProbLog

```
bird(sparrow).
bird(eagle).
bird(ostrich).
predates(eagle,sparrow).
predates(cheetah, ostrich).

0.9 :: flies(X) :- bird(X).
0.8 :: bird(X) :- predates(X,Y), bird(Y).
```

Probabilistic Logic Programming

- **rules**: definite clauses h :- $b_1, \ldots, b_n$ where h is the head and $b_1, \ldots, b_n$ is the body of the rule.
- **facts**: atoms a representing deterministic outcomes.
- **probabilistic rules**: definite clauses $p :: h$:- $b_1, \ldots, b_n$ where $p \in [0, 1]$ is the probability of the rule.
- **probabilistic facts**: $p :: a$ representing probabilistic outcomes.

ProbLog

```
bird(sparrow).
bird(eagle).
bird(ostrich).
predates(eagle,sparrow).
predates(cheetah, ostrich).

0.9 :: flies(X) :- bird(X).
0.8 :: bird(X) :- predates(X,Y), bird(Y).

query(flies(cheetah)).
```

Probabilistic Queries

P(`flies(cheetah)`) = 0.72

ProbLog: probabilistic inference

```
bird(sparrow).
bird(eagle).
bird(ostrich).
predates(eagle,sparrow).
predates(cheetah, ostrich).

0.9 :: bird_fly(X).
flies(X) :- bird(X), bird_fly(X).

0.8 :: bird_predator_is_bird(X,Y).
bird(X) :- predates(X,Y), bird(Y), bird_predator_is_bird(X,Y).
```

Take probabilities out of rules

probabilistic rules can be made deterministic by introducing auxhiliary probabilistic facts.

- probabilistic rule: $p :: h$:- $b_1, \ldots, b_n$
- deterministic version of the rule: h :- $b_1, \ldots, b_n, a$
- auxhiliary probabilistic fact: $p :: a$ (a must be parameterized by the logical variables in the rule)

ProbLog: probabilistic inference

$$P(\omega) = \prod_{\substack{p\,:\,a \in \mathcal{F},\\ a \in A(\omega)}} p \prod_{\substack{p\,:\,a \in \mathcal{F},\\ a \notin A(\omega)}} (1-p)$$

Probability of a possible world

- $\mathcal{F}$ is the set of ground instances of the probabilistic facts in the logic program.
- ω is a possible world, i.e., a truth assignments to the elements of $\mathcal{F}$.
- $A(\omega)$ is the set of ground instances in $\mathcal{F}$ that are true according to ω.

ProbLog: probabilistic inference

$$P(\omega) = \prod_{\substack{p\,:\,a \in \mathcal{F}, \\ a \in A(\omega)}} p \prod_{\substack{p\,:\,a \in \mathcal{F}, \\ a \notin A(\omega)}} (1-p)$$

```
bird(sparrow).
bird(eagle).
bird(ostrich).

0.9 :: bird_fly(X).
flies(X) :- bird(X), bird_fly(X).
```

Probability of a possible world: example

- $\mathcal{F} = \{$`bird_fly(sparrow), bird_fly(eagle), bird_fly(ostrich)`$\}$
- $A(\omega) = \{$`bird_fly(sparrow), bird_fly(eagle)`$\}$
- $P(\omega) = 0.9 \cdot 0.9 \cdot (1 - 0.9) = 0.081$

ProbLog: probabilistic inference

$$P(\phi) = \sum_{\omega \models \phi} P(\omega)$$

Probability of a formula (query)

The probability of a formula ϕ is the sum of the probabilities of the possible worlds where the formula holds.

ProbLog: probabilistic inference

```
bird(sparrow).
bird(eagle).
bird(ostrich).
predates(eagle,sparrow).
predates(cheetah, ostrich).

0.9 :: flies(X) :- bird(X).

0.8 :: bird(X) :- predates(X,Y), bird(Y).
```

Probability of a query: example

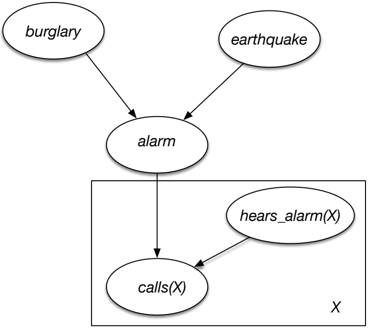
P(`flies(cheetah)`) = 0.9 * 0.8 = 0.72

- In all possible worlds where `flies(cheetah)` holds, `bird(ceetach)` also holds (it's the only way to prove `flies(cheetah)`).
- The probabilities associated to the other ground instances sum to one as all combinations are present in the possible worlds where `flies(cheetah)` holds.

Bayesian Logic Networks in ProbLog

```
0.1 :: burglary.
0.2 :: earthquake.
0.7 :: hears_alarm(john).
0.7 :: hears_alarm(mary).

alarm :- burglary.
alarm :- earthquake.
calls(X) :- alarm, hears_alarm(X).
```



ProbLog: efficient probabilistic inference

Probablistic inference as Weighted Model Counting (WMC)

$$P(\phi) = WMC(\phi) = \sum_{\omega \models \phi} \prod_{\omega \models \ell} w(\ell)$$

- The logic program + the query (and/or evidence) are grounded (instanting variables to constants) and converted into a format amenable to efficient computation (Clark's completion).
- Ground probabilistic facts (and their negation) are given as weights the probability of the fact (or 1 minus it if negated).
- All other literals have weight equal to 1.

ProbLog: efficient probabilistic inference

WMC by knowledge compilation (d-DNNF)

The ground weighted program (+ query/evidence) is compiled into a compact graphical representation, like a d-DNNF:

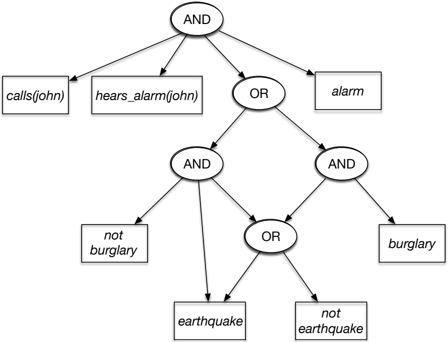
- **NNF**: each leaf is a literal, each internal node is AND or OR
- **DNNF**: decomposable NNF, no two children of an AND node share any atom (can multiply)
- **d-DNNF**: deterministic DNNF, for any OR node, each pair of children should represent logically inconsistent alternatives (can sum)
- **smooth d-DNNF**: all children of an OR node should use exactly the same set of atoms.

Knowledge compilation example: d-DNNF

```
0.1 :: burglary.
0.2 :: earthquake.
0.7 :: hears_alarm(john).
0.7 :: hears_alarm(mary).

alarm :- burglary.
alarm :- earthquake.
calls(X) :- alarm,
            hears_alarm(X).

query(calls(john)).
```



ProbLog: efficient probabilistic inference

WMC by knowledge compilation (d-DNNF)

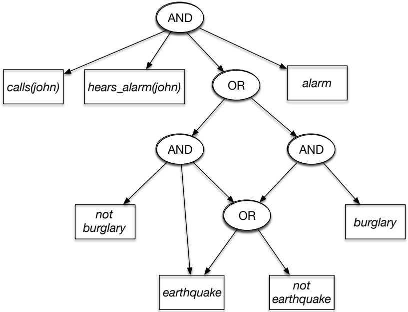
The d-DNNF is converted into an Algebraic Circuit (AC):

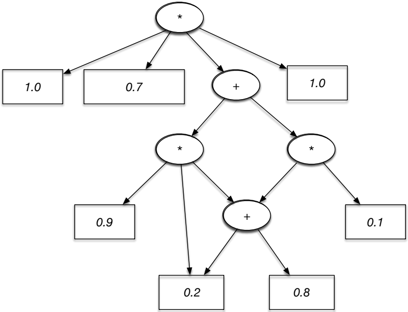
- AND are replaced by products
- OR are replaced by sums
- Literals are replaced by their weight

Knowledge compilation example: d-DNNF to AC

```
0.1 :: burglary.
0.2 :: earthquake.
0.7 :: hears_alarm(john).
0.7 :: hears_alarm(mary).
```

```
alarm :- burglary.
alarm :- earthquake.
calls(X) :- alarm, hears_alarm(X).
query(calls(john)).
```





ProbLog: efficient probabilistic inference

Further improvements

- **Lifted inference**: exploit symmetries (individuals behaving the same) to avoid full grounding (sets of individuals grouped together).
- **Approximate inference**: using e.g. sampling techniques (as in MLN), possibly combined with decomposition strategies (hashing functions).

ProbLog: parameter learning

```
w1 :: burglary.
w2 :: earthquake.
w3 :: hears_alarm(X).
```

```
alarm :- burglary.
alarm :- earthquake.
calls(X) :- alarm, hears_alarm(X).
```

Maximum likelihood parameter learning

$$\mathbf{w}^* = \operatorname*{argmax}_{\mathbf{w}} \prod_{\omega \in \mathcal{T}} P(\omega; \mathbf{w})$$

- Probabilities associated to probabilistic facts are unknown (parameters)
- There exists a training set $\mathcal{T}$ of (possibly partial) intepretations (i.e., possible worlds)
- Learning amounts at finding parameters maximizing the likelihood of $\mathcal{T}$

ProbLog: parameter learning

```
w1 :: burglary.
w2 :: earthquake.
w3 :: hears_alarm(X).
```

```
alarm :- burglary.
alarm :- earthquake.
calls(X) :- alarm, hears_alarm(X).
```

Complete interpretations: fractional counts

$$w_k^* = \frac{\sum_{\omega \in \mathcal{T}} n_k(A(\omega))}{\sum_{\omega \in \mathcal{T}} n_k(\mathcal{F}(\omega))}$$

- The parameter of a probabilistic fact is estimated as the fraction of its groundings that hold in the training set over the total number of its possibile groundings (same as for BN).
- $n_k(A(\omega))$ is the number of groundings of the k-th probabilistic fact that hold in possible world ω.
- $n_k(\mathcal{F}(\omega))$ is the total number of groundings of the k-th probabilistic fact for possible world ω (true and false).

ProbLog: parameter learning

```
w1 :: burglary.
w2 :: earthquake.
w3 :: hears_alarm(X).
```

```
alarm :- burglary.
alarm :- earthquake.
calls(X) :- alarm, hears_alarm(X).
```

Partial interpretations: Expectation-Maximization

$$w_k^{i+1} = \frac{\sum_{\omega \in \mathcal{T}} \sum_{f'_k \in \mathcal{F}_k(\omega)} P(f'_k | \mathbf{E}(\omega) = \mathbf{e}(\omega); w^i)}{\sum_{\omega \in \mathcal{T}} n_k(\mathcal{F}(\omega))}$$

- $\mathbf{E}(\omega)$ are the observed groundings in ω, and $\mathbf{e}(\omega)$ their values.
- $\mathcal{F}_k(\omega)$ is the subset of $\mathcal{F}(\omega)$ containing groundings of the k-th probabilistic fact.
- f'_k ranges over these groundings.
- $P(f'_k | \mathbf{E}(\omega) = \mathbf{e}(\omega); w^i)$ is the probability that f'_k holds in ω given the observed facts and the current estimate of the parameters (initialized randomly).

References

Bibliography

- Luc De Raedt, Kristian Kersting, Sriraam Natarajan, David Poole, *Statistical Relational Artificial Intelligence: Logic, Probability, and Computation* , Morgan & Claypool, 2016.
- Domingos, Pedro and Kok, Stanley and Lowd, Daniel and Poon, Hoifung and Richardson, Matthew and Singla, Parag, *Markov Logic*. In Probabilistic Inductive Logic Programming. New York: Springer, 2007.
- Daan Fierens, Guy Van den Broeck, Joris Renkens, Dimitar Shterionov, Bernd Gutmann, Ingo Thon, Gerda Janssens and Luc De Raedt, *Inference and Learning in Probabilistic Logic Programs using Weighted Boolean Formulas*, In Theory and Practice of Logic Programming, volume 15, 2015.

References

Software

- Markov Logic Networks (Alchemy) [http://alchemy.cs.washington.edu/]
- Problog [https://dtai.cs.kuleuven.be/problog/]

Temporary page!

LaTeX was unable to guess the total number of pages correctl
As there was some unprocessed data that should have been
added to the final page this extra page has been added to
receive it.
If you rerun the document (without altering it) this surplus pa
will go away, because LaTeX now knows how many pages to
expect for this document.