Approximate Inference for Logic Programs with Annotated Disjunctions

Stefano Bragaglia and **Fabrizio Riguzzi**
DEIS – University of Bologna, ENDIF – University of Ferrara.
{stefano.bragaglia@unibo.it, fabrizio.riguzzi@unife.it}

Combining logic and probability is a field of research that has received much attention in the last few years.

In this vein, many formalisms have been introduced such as Markov Logic Networks, ProbLog and Logic Programs with Annotated Disjunction (LPADs), that allow to represent probabilistic information in logic.

LPADs are particularly interesting because they can express cause-effect relationships among events, possible effects of a cause and the contemporary contribution of more causes to the same effect in a very natural way.

From a syntactic point of view, an LPAD consists of a set of disjunctive clauses in which each atom in the head is annotated with a probability value between 0 and 1. The atoms of the head represent the mutually exclusive and exhaustive set of effects of the event represented by the body. The sum of the probabilities associated to the head atoms must be 1.

Their semantic is based on the concept of *instance*, that is a normal logic program obtained by choosing a logical atom from the head of each grounding of every clause of the LPAD. The probability of an instance is computed by multiplying the probability values of all the atoms chosen for that instance. The probability of a query is given by the sum of the probabilities of each instance where the query is true.

Inference with LPADs can be performed with the `cplint` system [1] that first computes explanations for a query and then computes the probability of the query by making the explanations mutually exclusive by means of Binary Decision Diagrams (BDDs). An explanation for a query is a set of choices for head atoms such that the query is true in all the instances that respect the choices.

`cplint` finds the explanation by using a meta-interpreter approach that performs resolution and keeps the current set of choices. Each time the selected goal is resolved with a disjunctive clause, the set of choices is enlarged with the new choice performed. A derivation branch may fail because no resolution is applicable or because the set of choices becomes inconsistent.

Once all the explanations for the query have been found, a BDD is built that allows to compute the probability by using a dynamic programming algorithm that traverses the diagram.

In some domains, exact inference may be impossible. Therefore we considered approximate algorithms, both deterministic and stochastic.

The approximate deterministic algorithm based on *iterative deepening* builds the set of explanations incrementally by building the proof tree partially. Given a partial proof tree, the branches corresponding to successful derivations provide a lower bound of the probability of the query, while the branches corresponding to successful or incomplete derivations provide an upper bound of the probability of the query.

The algorithm stops when the difference between the upper and the lower bound is below a user defined threshold. Otherwise, it further extends the proof tree.

Another approximate deterministic algorithm uses *branch and bound* to find the k most probable explanations. These provide a lower bound of the probability of the query.

We considered also a stochastic algorithm based on a Monte Carlo approach: instances are repeatedly sampled from the LPAD and the probability of the query is given by the fraction of sampled instances where the query is true. Sampling is done efficiently by considering only the clauses that are relevant for the query. The algorithm uses a meta-interpreter that chooses stochastically head atoms of resolving clauses.

Each algorithm has been applied to several datasets, both artificial and real.

One of the real world dataset consists of biological graphs sampled from a network with 5220 nodes and 11530 edges that describes the biological entities responsible for the Alzheimer's disease together with their relationships.

The results we gathered from experiments on this dataset show that the Monte Carlo algorithm can solve almost three times as many sampled graphs than the standard inference. In fact Monte Carlo can easily handle any sampled graph up to 3400 edges while standard inference starts to fail with only 1200 edges.

The experimental results show that the CPU time required by Monte Carlo does not grow proportionally with the problem size as it happens with the standard inference. In addition, despite being slower on smaller problems, Monte Carlo algorithm is less demanding in terms of computational resources and it achieves better performances than standard inference on problems with more than 2400 edges.

[1] `http://www.ing.unife.it/software/cplint/`