MetaProbLog

Theofrastos Mantadelis[1] and Gerda Janssens[2]

[1]University of Porto, theo.mantadelis@dcc.fc.up.pt
[2]KU Leuven, gerda.janssens@cs.kuleuven.be

April 7, 2015

1 Introduction

MetaProbLog is a framework of the ProbLog [2, 5] probabilistic logic programming language. ProbLog extent Prolog programs by annotating facts with probabilities. In that way it defines a probability distribution over all Prolog programs. ProbLog follows the distribution semantics presented by Sato [8]. MetaProbLog extends the semantics of ProbLog by defining a "ProbLog engine" which permits the definitions of probabilistic meta calls [6, 7]. MetaProbLog inference, currently allows the computation of marginal probabilities with or without evidence. Furthermore, it allows the computation of marginal probabilities for the answers of non-ground queries.

MetaProbLog first appeared in [6] where the focus was to solve meta calls for ProbLog. From that time, MetaProbLog's implementation has been extended to include many new features. The MetaProbLog framework has many similarities with the ProbLog framework [5] such as support for annotated disjunctions, tabling, general negation. MetaProbLog introduces several unique features such as meta calls, datasets. Furthermore, MetaProbLog shares some features that appear in ProbLog 2 [3] such as multiple queries, evidence.

The aim of this article is to demonstrate the unique features of MetaProbLog with simple examples.

2 State-of-the-art Inference

MetaProbLog has two primary inference methods: exact inference and program sampling. The exact inference method, uses knowledge compilation [1]. MetaProbLog implements two different knowledge compilation approaches. The first approach is compiling into Reduced Ordered Binary Decision Diagrams (ROBBDs) and the second approach is compiling into smooth deterministic, Decomposable Negation Normal Forms (sd-DNNFs). Currently, for the majority of problems ROBDDs perform much faster than sd-DNNFs for MetaProbLog.

Both compilation approaches are the current state-of-the-art in knowledge compilation.

Extensive experiments showed that sd-DNNFs perform better on specific problems and more specifically on queries with evidence. Comparison with several different ProbLog pipelines [11], showed that there is not a clear universal winner in performance. Each ProbLog pipeline has a class of problems which performs notably better than the other pipelines. For that reason MetaProbLog provides to the user a wide selection of options that can optimize the performance over specific problems.

In addition, MetaProbLog implements a program sampling [5] inference method. This elegant simple sampling approach is very natural to ProbLog and has been proven very useful when exact inference is intractable.

Both exact and program sampling inference support general negation, multiple queries and queries with evidence. Furthermore, MetaProbLog supports for all inference methods probabilistic meta calls and non-ground queries that return answers through backtracking.

3 Example Program & Queries

To illustrate some of the unique features of MetaProbLog, we present a small MetaProbLog program that defines two different probabilistic graphs and uses them as different datasets. To illustrate the modularity of MetaProbLog programs we use three different files. MetaProbLog not only fully complies with the module functionality of Yap Prolog, but it also implements some extra functionality, namely datasets, that allows MetaProbLog to load different datasets. The user can choose which dataset(s) are visible to each module at runtime.

```
:- module(probabilistic_graph, [path/2,
                                import_dataset/1,
                                import_all_datasets/0,
                                problog_inference/3]).
:- use_module(metaproblog).
:- use_module(problog_datasets).
:- use_dataset(graph01).
:- use_dataset(graph02).

:- problog_table path/2.
path(X,Y) :- drc(X,Y).
path(X,Y) :-
  drc(X, Z),
  Z \== Y,
  path(Z, Y).

% Probabilistic meta calls.
% problog_inference/2 inherits the inference method from the call.
P::path_prob(P).
```

```
most_probable_path(From, To1, To2) :-
  problog_inference(path(From, To1), P1),
  problog_inference(path(From, To2), P2),
  BestP is max(P1, P2),
  path_prob(BestP).

% Syntactic Sugar for easier use.
import_dataset(DataSet) :-
  problog_import_only(DataSet, [drc/2]).
import_all_datasets :-
  problog_import_only(graph01, [drc/2]),
  problog_import(graph02, [drc/2]).

% These are two different files that define the datasets.
% file: graph01.yap                    file: graph02.yap
:- dataset(graph01, [drc/2]).        :- dataset(graph02, [drc/2]).
0.15::drc(1, 2).                     0.52::drc(1, 3).
0.24::drc(1, 4).                     0.43::drc(1, 5).
0.33::drc(1, 6).                     0.34::drc(2, 4).
0.42::drc(2, 3).                     0.25::drc(2, 6).
0.51::drc(2, 5).                     0.25::drc(3, 4).
0.42::drc(3, 4).                     0.34::drc(4, 2).
0.33::drc(3, 6).                     0.43::drc(5, 2).
0.24::drc(4, 5).                     0.52::drc(6, 2).
                                     0.15::drc(5, 6).
```

ProbLog datasets function similarly with Prolog modules, but loading a dataset does not automatically import the exported predicates of the dataset. For our example, in order to access the graph of a dataset we must also import it.

```
   ?- import_dataset(graph01).
Yes.
```

We can then query our probabilistic graph. ProbLog's inference is based on calculating the success probability of a query. For our example, the probability that a path exists from node 1 to node 4 at the imported graph.

```
   ?- problog_inference(problog_exact, path(1, 4), P).
P = 0.2601096
```

Similarly we are now going to only import the dataset that contains graph02. This practically changes the background knowledge of the ProbLog program allowing us to have different datasets to query.

```
   ?- import_dataset(graph02).
Yes.
   ?- problog_inference(problog_exact, path(1, 4), P).
P = 0.1903485
```

Notice that the probability for the success of the query altered. Finally, MetaProbLog datasets allow us to import more than one datasets at the same time. In our case both graph01 and graph02 can be imported. For the next query we also illustrate the use of program sampling.

```
   ?- import_all_datasets.
Yes.
   ?- problog_inference(problog_exact, path(1, 4), P).
P = 0.5724326
   ?- problog_inference(problog_program_sampling, path(1, 4), P).
P = 0.570875
```

Obviously, now that both graphs are available the probability for the success of the query increases. The merge of the two graphs creates several extra paths between the two nodes which results to a higher probability than just the sum of the probabilities when using each graph separately. We want to point out that the explanations (SLD derivations) might grow exponentially with the addition of edges.

We also present a query that uses `problog_answers` this is a different inference task that allows MetaProbLog to calculate the answer of a non-ground call and return the marginal probability of its success. While before we were querying for the success probability, now we query to get the answer of an unbound variable and the success probability of that answer for the ProbLog program. MetaProbLog, uses backtracking in order to return all possible answers and their respective probabilities.

```
   ?- problog_inference(problog_answers(problog_exact), path(1, A), P).
A = 2,
P = 0.5683266 ? ;
A = 4,
P = 0.5724326 ? ;
A = 6,
P = 0.552904 ? ;
A = 3,
P = 0.6187836 ? ;
A = 5,
P = 0.609305 ? ;
```

Furthermore, MetaProbLog is capable to perform probabilistic meta calls. The following query calls inference within inference isolating the results in order to be used for decisions. In this case to choose the highest probability.

```
   ?- problog_inference(problog_exact, most_probable_path(1,3,5),P).
P = 0.6187836
```

Finally, we present a query that uses evidence which is one of the newest MetaProbLog features. We again query for the success of `path(1, 4)` but this time we state that a path exists from node 2 to node 3.

```
   ?- problog_inference(problog_exact, path(1, 4)/path(2, 3), P).
P = 0.606907380952381
```

4 Applications

Until now, many common ProbLog applications have been used in order to verify and improve MetaProbLog, examples include: link discovery in Biomine Alzheimer database [9, 2]; querying WebKB [3]; querying a probabilistic dictionary [10]. None of these applications is novel; as they are examined from the original ProbLog implementation.

MetaProbLog is able to model all statistical graphical models such as Hidden Markov Models, Bayesian Networks, Probabilistic Graphs. Furthermore, any Prolog program could be extended with MetaProbLog to use probabilities and take decisions with them.

5 Future Features & Improvements

Some of our future directions are:

1. Use MetaProbLog in a rule learner.
2. Extend MetaProbLog with a parameter learning algorithm similar with [4].
3. Improve the performance of multiple queries and queries with evidence.
4. Migrate inference methods from the original ProbLog.

Furthermore, we are interested in implementing novel applications and analyze probabilistic databases. We are interested in finding new datasets and problems that can be nicely modeled by our system.

6 Distribution

MetaProbLog's website (`https://www.dcc.fc.up.pt/metaproblog/tiki-index.php`).

References

[1] A. Darwiche and P. Marquis. A knowledge compilation map. *J. Artif. Intell. Res. (JAIR)*, 17:229–264, 2002.

[2] L. De Raedt, A. Kimmig, and H. Toivonen. ProbLog: a probabilistic Prolog and its application in link discovery. In *IJCAI*, pages 2468–2473. AAAI Press, 2007.

[3] D. Fierens, G. V. D. Broek, J. Renkens, D. Shterionov, B. Gutmann, I. Thon, G. Janssens, and L. de Raedt. Inference and learning in probabilistic logic programs using weighted boolean formulas. *TPLP*, 2013.

[4] B. Gutmann, A. Kimmig, K. Kersting, and L. De Raedt. Parameter learning in probabilistic databases: A least squares approach. In *ECML/PKDD (1)*, pages 473–488, 2008.

[5] A. Kimmig, B. Demoen, L. D. Raedt, V. S. Costa, and R. Rocha. On the implementation of the probabilistic logic programming language ProbLog. *TPLP*, 11:235–262, 2011.

[6] T. Mantadelis and G. Janssens. Nesting probabilistic inference. *Computing Research Repository*, abs/1112.3785, 2011.

[7] L. D. Raedt and A. Kimmig. Probabilistic programming concepts. *CoRR*, abs/1312.4328, 2013.

[8] T. Sato. A statistical learning method for logic programs with distribution semantics. In *Proceedings of International Conference on Logic Programming*, pages 715–729, 1995.

[9] P. Sevon, L. Eronen, P. Hintsanen, K. Kulovesi, and H. Toivonen. Link discovery in graphs derived from biological databases. In *DILS*, pages 35–49, 2006.

[10] D. Shterionov and G. Janssens. Data acquisition and modeling for learning and reasoning in probabilistic logic environment. In L. Antunes, H. S. Pinto, R. Prada, and P. Trigo, editors, *Proceedings of the 15th Portuguese Conference on Artificial Intelligence*, pages 298–312, 2011.

[11] D. Shterionov and G. Janssens. Crucial components in probabilistic inference pipelines. In *ACM/SIGAPP Symposium On Applied Computing, Salamanca, Spain, 13 - 17 April 2015*, 2015. Accepted.