ProbLog2: From Probabilistic Programming to Statistical Relational Learning

Joris Renkens Dimitar Shterionov Guy Van den Broeck Jonas Vlasselaer
Daan Fierens Wannes Meert Gerda Janssens Luc De Raedt
Department of Computer Science, KU Leuven
Celestijnenlaan 200A, B-3001 Heverlee, Belgium
firstname.lastname@cs.kuleuven.be

Abstract

ProbLog is a probabilistic programming language based on Prolog. The new ProbLog system - called ProbLog2 - can solve a range of inference and learning tasks typical for the Probabilistic Graphical Models (PGM) and Statistical Relational Learning (SRL) communities. The main mechanism behind ProbLog2 is a conversion of the given program to a weighted Boolean formula. We argue that this conversion approach can also be applied - with certain restrictions - to other probabilistic programming languages such as Church and Figaro.

1 Introduction

ProbLog [6] is a probabilistic extension of Prolog based on Sato's distribution semantics [19]. A ProbLog program consists of a set of probabilistic facts and a logic program, i.e. a set of rules. A probabilistic fact, written as `p::f`, is an atom f annotated with the probability p of this atom being true. A ProbLog program specifies a probability distribution over possible worlds, as dictated by the distribution semantics [19]: each ground probabilistic fact gives an atomic choice; a total choice σ is obtained by making an atomic choice for all ground probabilistic facts (the probability of σ is the product of the probabilities of the atomic choices); each total choice σ can be extended to a possible world, which contains σ together with all the atoms inferable from σ using the rules.

Example 1 *We are packing our baggage to fly to the NIPS conference. We have a set of items, each having a particular weight, and we pack each item with a certain probability. We want to compute the probability that we will have excess baggage, i.e., that the total weight of our baggage will exceed a given limit L. We can model this in ProbLog as follows.*

```
0.9::pack(skis).            weight(skis,5).
0.2::pack(helmet).          weight(helmet,2).
0.6::pack(gloves).          weight(gloves,1).
...                         ...

excess(L) :- excess([skis,helmet,gloves,...],L). % all possible items

excess([],L) :- L<0.
excess([I|R],L) :- pack(I), weight(I,W), L2 is L-W, excess(R,L2).
excess([I|R],L) :- not(pack(I)), excess(R,L).
```

Given a ProbLog program, one can solve a number of inference and learning tasks. The previous ProbLog system [14] was rooted in the (Inductive) Logic Programming (ILP) community and hence focusses on tasks that are most natural in an (I)LP setting. The main inference task in this system is to compute the so-called *succes probability* of a query [14], which is a probabilistic variant of proving a query in Prolog. For learning, the setting that was first considered is *learning from entailment*,

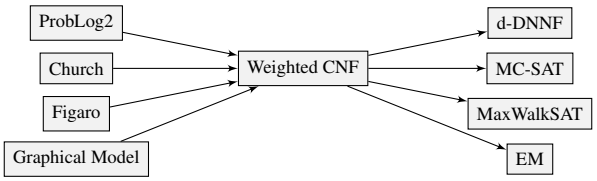

Figure 1: ProbLog2 pipeline architecture.

a probabilistic variant of the standard ILP setting (each example in the dataset consists of an atom together with its target probability) [11].

The new ProbLog system - called ProbLog2 - is more inspired by the work in the PGM and SRL community, leading to different tasks being tackled [8]. The main inference task in ProbLog2 is to compute the *marginal probabilities* of a set of query atoms given some evidence. This generalizes the traditional ProbLog task of computing a success probability in two ways: (1) evidence is considered, (2) multiple queries can be answered simultaneously. Another inference task considered is finding the *Most Probable Explanation (MPE)*: given some evidence, the goal is to find the most likely truth value of all the non-evidence atoms. For learning, ProbLog2 uses the *learning from interpretations* setting [12], which matches the typical learning setting from PGM and SRL (each example is a partial interpretation, i.e., we observe the truth value of a subset of all atoms in the program). ProbLog2 solves all these tasks by various algorithms that all operate on a weighted CNF, i.e., a weighted Boolean formula in Conjunctive Normal Form.

2 ProbLog2: Inference and Learning on Weighted CNFs

ProbLog2 has a pipeline architecture of grounding, followed by conversion to weighted CNF, followed by inference and learning on the weighted CNF [8]. The main motivation behind this new architecture is that, once a weighted CNF is obtained, inference and learning can be done using state-of-the-art algorithms from SRL and PGM. In this sense, ProbLog2 forms a bridge from probabilistic programming to SRL and PGM.

Grounding and conversion. In order to convert the ProbLog program to a weighted CNF, the program is first grounded. Grounding the entire program is usually prohibitive. Hence, our grounding algorithm only considers the rules relevant to the task at hand (depending on the query and evidence, or the learning examples). The resulting ground program is then converted to an equivalent weighted CNF. This conversion must compensate for the change in semantics from logic programming to propositional logic and can be carried out by a number of existing methods [13, 15]. The resulting CNF can then be 'conditioned' by conjoining it with the evidence, optionally followed by simplification (using unit propagation, etc).

Example 2 *In our baggage example, if we ask the query* $excess(6)$*, part of the resulting Boolean formula looks a follows (for brevity we consider only 3 items, we abbreviate 'skis' to 's', etc).*

$$excess(6) \leftrightarrow excess([s,h,g],6)$$
$$excess([s,h,g],6) \leftrightarrow (pack(s) \wedge excess([h,g],1)) \vee (\neg pack(s) \wedge excess([h,g],6))$$

This formula can be converted to CNF using standard methods. The weight function for the CNF follows from the probabilistic facts (e.g., the weight for $pack(s)$ *is 0.9, for* $\neg pack(s)$ *0.1, etc) [8].*

Inference on weighted CNFs. Once we have a weighted formula, we can apply well established methods from SRL and PGM. Computing marginal probabilities reduces to weighted model counting [18, 8], which can be solved in various ways. We use knowledge compilation [5, 2, 1]: we compile the CNF to a d-DNNF circuit; probabilities can then be computed in time linear in the circuit size. Finding the MPE solution can also be done with knowledge compilation [4], or with algorithms for weighted MAX-SAT such as MaxWalkSAT [16, 7]. Even lifted inference, which exploits symmetries in the probabilistic programs during inference, can be performed on weighted first-order CNFs [21, 9, 20].

Learning on weighted CNFs. ProbLog2 currently supports parameter learning in the *learning from interpretations* setting [12]. Given is a ProbLog program with unknown parameters (fact probabil-

ities) and a set $\mathcal{I}$ of partial interpretations. Each partial interpretation $I \in \mathcal{I}$ consists of a set of true atoms I^+ and a set false atoms I^-. The goal is to learn the maximum likelihood parameter estimates, i.e. to maximize $\prod_{I \in \mathcal{I}} P(I)$, with $P(I) = \prod_{f \in I^+} P(f) \prod_{f \in I^-} (1 - P(f))$. This corresponds to the standard learning setting in SRL and PGM. We use Expectation Maximization (EM) to solve this [12]. In the E-step, several marginal probabilities need to be calculated. This is done using the above inference approach, namely via knowledge compilation. The advantage of knowledge compilation in the context of parameter learning is that the compiled circuits depend only on the structure of the program but not on its parameters. Hence, we only need to compile the circuits once (upfront) and can then use them in each iteration of the EM algorithm.

3 Weighted CNFs for Other Probabilistic Programming Languages

The probabilistic programming community has given rise to a diverse set of languages, including functional, logic and object-oriented programming languages. As we showed, probabilistic logic programming can benefit from using a representation like weighted CNFs. We argue that the same is true for other probabilistic programming languages, even those that have no obvious connection to logic. There is of course one important prerequisite, namely that the given probabilistic program can indeed be transformed to a weighted CNF. If that is the case, all the algorithms used by ProbLog apply directly.

Let us now consider the problem of transforming a functional probabilistic program written in Church [10] into a weighted CNF. Church is a probabilistic extension of Scheme for modeling stochastic generative processes.

Example 3 *The distribution of Example 1 is modeled in Church by the following four functions.*

```
(define skis (if (flip 0.9) 5 0))
(define helmet (if (flip 0.2) 2 0))
(define gloves (if (flip 0.6) 1 0))
(define (excess l) (> (+ skis helmet gloves) l))
```

For a given query, there are many weighted logical theories that correspond to this distribution. They contain the propositions $skis\text{-}flip$, $helmet\text{-}flip$ and $gloves\text{-}flip$, representing the random flips in the program. The most basic weighted theory is in disjunctive normal form. It enumerates all subsets of items that exceed the weight limit (one of which is $skis\text{-}flip \wedge helmet\text{-}flip \wedge gloves\text{-}flip$) and disjoins these terms, stating that one of these subsets has to be included for the query to succeed. A more compact representation is obtained by executing the Church program and backtracking on flip statements, constructing a computation graph of the program. This graph can then be transformed into a weighted CNF, similar to the ProbLog approach [8]. This transformation introduces additional propositions, such as $excess([s, h, g], 6)$ in Example 2, as a shorthand for a complex state of other random variables.

A similar technique can be applied to object-oriented probabilistic programming languages, such as Figaro [17]. In Figaro, we can model the distribution of Example 1 with a class for each item and a property for each item that represents its weight. These properties are actually probability distributions. Arithmetic operations on these distributions can be modeled using the apply method. Once we convert the Figaro program to a weighted CNF, again, existing algorithms can be applied.

Of course this strategy is only applicable to programs that can indeed be transformed to a weighted CNF. This is the case for all ProbLog programs, but not necessarily for all Church or Figaro programs (or BLOG programs, etc). Concretely, one restriction is that we need to be able to derive a finite CNF, which requires the given program to be evidence-finite [3]. Furthermore, it is unclear how to do the transformation for open world models as seen frequently in BLOG. Finally, while the transformation works for programs with Boolean random variables, and can be extended for non-Boolean variables, it is inapplicable for continuous random variables.

Even with these restrictions on the supported programs, we believe this strategy of converting to a weighted CNF to be a promising one for the field of probabilistic programming. It shows that different languages like Figaro, Church and ProbLog - despite their syntactical and semantical differences - can potentially use the same underlying inference and learning algorithms.

Acknowledgments

DF and GVdB are supported by the Research Foundation-Flanders (FWO-Vlaanderen). JR is supported by PF-10/010 NATAR. JV is supported by IWT-SBO-100031 POM2. Research supported by GOA 13/010 Research Fund KULeuven 'Declarative Modeling Languages for Machine Learning and Data Mining'.

References

[1] M. Chavira, A. Darwiche, and M. Jaeger. Compiling relational bayesian networks for exact inference. *International Journal of Approximate Reasoning*, 42(1):4–20, 2006.

[2] A. Darwiche. New advances in compiling CNF into decomposable negation normal form. In *Proceedings of the 16th European Conference on Artificial Intelligence*, pages 328–332, 2004.

[3] D. Koller, D. McAllester, and A. Pfeffer. Effective Bayesian Inference for Stochastic Programs. In *Proceedings of the 14th National Conference on Artificial Intelligence*, 1997.

[4] A. Darwiche. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press, 2009. Chapter 12.

[5] A. Darwiche and P. Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17(1):229–264, Sept. 2002.

[6] L. De Raedt, A. Kimmig, and H. Toivonen. ProbLog: a probabilistic Prolog and its application in link discovery. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 2468–2473. AAAI Press, 2007.

[7] P. Domingos, S. Kok, D. Lowd, H. Poon, M. Richardson, and P. Singla. *Probabilistic Inductive Logic Programming - Theory and Applications*, chapter 'Markov Logic'. Lecture Notes in Computer Science. Springer, 2008.

[8] D. Fierens, G. Van den Broeck, I. Thon, B. Gutmann, and L. De Raedt. Inference in Probabilistic Logic Programs using Weighted CNF's. In *Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence*, pages 211–220, 2011.

[9] V. Gogate and P. Domingos. Probabilistic Theorem Proving. In *Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence*, pages 247–255, 2011.

[10] N. D. Goodman, V. K. Mansinghka, D. M. Roy, K. Bonawitz, and J. B. Tenenbaum. Church: A language for generative models. In *In UAI*, pages 220–229, 2008.

[11] B. Gutmann, A. Kimmig, K. Kersting, and L. De Raedt. Parameter learning in probabilistic databases: A least squares approach. In *In Proceedings of the European Conference on Machine Learning and Knowledge Discovery*, volume 5211 of *Lecture Notes in Computer Science*, pages 473–488. Springer, 2008.

[12] B. Gutmann, I. Thon, and L. De Raedt. Learning the parameters of probabilistic logic programs from interpretations. In *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 581–596, 2011.

[13] T. Janhunen. Representing normal programs with clauses. In *In Proceedings of the 16th European Conference on Artificial Intelligence*, pages 358–362. IOS Press, 2004.

[14] A. Kimmig, B. Demoen, L. De Raedt, V. Santos Costa, and R. Rocha. On the Implementation of the Probabilistic Logic Programming Language ProbLog. *Theory and Practice of Logic Programming*, 11:235–262, 2010.

[15] T. Mantadelis and G. Janssens. Dedicated tabling for a probabilistic setting. In *Technical Communications of the 26th International Conference on Logic Programming*, pages 124–133, 2010.

[16] J. D. Park. Using weighted MAX-SAT engines to solve MPE. In *Proceedings of the 18th National Conference on Artificial Intelligence*, pages 682–687, 2002.

[17] A. Pfeffer. Figaro: An object-oriented probabilistic programming language. *Charles River Analytics Technical Report*, 2009.

[18] T. Sang, P. Beame, and H. Kautz. Solving Bayesian networks by Weighted Model Counting. In *Proceedings 20th National Conference on Artificial Intelligence*, pages 475–482, 2005.

[19] T. Sato. A statistical learning method for logic programs with distribution semantics. In *Proceedings of the 12th International Conference on Logic Programming*, pages 715–729. MIT Press, 1995.

[20] G. Van den Broeck, A. Choi, and A. Darwiche. Lifted relax, compensate and then recover: From approximate to exact lifted probabilistic inference. In *Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence*, 2012.

[21] G. Van den Broeck, N. Taghipour, W. Meert, J. Davis, and L. De Raedt. Lifted probabilistic inference by first-order knowledge compilation. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume Three*, pages 2178–2185. AAAI Press, 2011.