Pattern-Based Compaction for ProbLog Inference (Extended Abstract)

DIMITAR SHTERIONOV, THEOFRASTOS MANTADELIS, GERDA JANSSENS

Department of Computer Science, KU Leuven
(*e-mail:* firstname.secondname@cs.kuleuven.be)

submitted 10 April 2013; accepted 23 May 2013

1 Background and Motivation

Combining logic and probabilities resulted in the field of Probabilistic Logic Programming (PLP). PLP programs are logic programs in which some of the facts are annotated with probabilities. Each of these (probabilistic) facts can either be true or false. A probabilistic program gives rise to an exponential number of possible worlds (a specific decision on the truth values of all probabilistic facts). A query is true in a subset of the possible worlds: the models of the PLP program with respect to the query. In a model each fact has a weight: its probability if it is true or (1− its probability) if it is false. The product of these weights is the model's weight. The sum of all models' weights, the *weighted model count* (WMC), is the success probability of a query. ProbLog (De Raedt et al. 2007) is a Prolog-based PLP system, which uses (a form of) WMC as its main inference mechanism.

Enumerating all possible worlds is a computationally expensive task. Current state-of-the-art probabilistic inference methods can avoid the exponential enumeration of the possible worlds (De Raedt et al. 2007; Fierens et al. 2011; Sato and Kameya 1997). These methods first collect successful proofs for the query. Then, the set of proofs are *converted* into a suitable Boolean formula representation on which WMC can be done in polynomial time. The latter step is well studied in the context of knowledge representation and logic programming (Janhunen 2004; Darwiche and Marquis 2002; Darwiche 2009) and resulted in advanced systems (Somenzi 2005; Darwiche 2004; Muise et al. 2012). Unfortunately, the limits of these systems are being reached during probabilistic inference. Crucial for the efficiency of the approach are the number of atoms in the proofs and how they form conjunctions and disjunctions in the proofs.

We know from our earlier work (Mantadelis and Janssens 2010) that compressing AND-/OR- clusters in DNFs significantly decreases the size of Boolean formulae and increases the overall performance of the PLP system. The method, though, has several drawbacks: (i) it requires the proofs to be converted to a Boolean formula in normal form (such as DNF) prior to detecting and optimizing the subformulae and (ii) it cannot handle Boolean formulae generated by PLP systems which support general negation, such as ProbLog. The method which we introduce here is more general than the one presented in (Mantadelis and Janssens 2010). It operates on a more general representation of a Boolean formula – AND-OR graphs which facilitate *pattern detection and compaction* and support negation. Therefore, our approach compacts the collected proofs before converting them to a Boolean formula in normal form (CNF or DNF) and handles

negated atoms. Our algorithm preserves equivalence in terms of WMC. It is an enabling step that reduces the complexity of the generation of the Boolean formulae and thus decreases the execution time of the following steps of the inference.

An AND-OR tree is a natural representation for the search space of a query in a logic program: AND nodes indicate that all children nodes (resp. subgoals) have to be solved, while for an OR node only one node needs to be solved. When a loop is introduced in the AND-OR tree it becomes an AND-OR graph. The collected proofs of a ProbLog program can easily be represented by such a graph. Our method applies on a specific form of an AND-OR graph with explicit terminal nodes to represent goals that are proven by facts and with edges between OR-nodes to avoid AND nodes with only one child. We say that a parent node *depends* on a child node. An edge is directed from a child towards a parent.

Our AND-OR graph compaction can be incorporated in any PLP system. We implemented it and evaluated its impact for the two state-of-the-art ProbLog systems – MetaProbLog (Mantadelis 2012, Chapter 6) and ProbLog2 (Fierens et al. 2013). A detailed description, analysis and evaluation of our approach can be found in (Shterionov et al. 2013).

2 Description and Evaluation of the AND-OR Graph Compaction Method

The aim of our method is to detect patterns among the nodes of an AND-OR graph and compact them accordingly such that the next steps in the WMC inference become simpler. That is, we look for specific subgraphs of AND-, OR- and terminal nodes and transform them to smaller-sized but equivalent (preserving the WMC) subgraphs. The transformations decrease the size of the whole AND-OR graph, which consequently will decrease the execution time of the following inference steps. Consider a graph $G_1 = (V_1, E_1)$ with query q as the graph root. Applying a set of our pattern-based transformations on G_1 results on a graph $G_2 = (V_2, E_2)$ with query q as its root, such that $|G_1| \geqslant |G_2|$, where $|G| = |V| + |E|$ is the size of a graph $G = (V, E)$. Converting G_1 and G_2 then into the Boolean formulae BF_1 and BF_2, preserves the WMC w.r.t. q ($WMC(BF_1) = WMC(BF_2)$) and $|BF_1| \geqslant |BF_2|$. BF_2 could then be used to calculate the success probability of q instead of BF_1.

We detect and compact five patterns. They correspond to AND-/OR- clusters (cf. Definition 1 and Definition 2) and to patterns which when compacted enable the detection (and compaction) of AND-/OR- clusters. We can proof that our detection and compaction algorithm is **sound** as it *preserves WMC*. It is also **deterministic**: it ensures the same output for the same input. We know of at least one pattern which our algorithm does neither detect nor compact. The transformed graph is not of minimal size, and therefore our algorithm is **not complete**. Our algorithm is implemented in Prolog and ensures reading, adding or deleting in constant time. Detecting a pattern is the bottleneck. The complexity of our algorithm in the worst case is polynomial (third degree) in the number of nodes.

Definition 1

AND-Cluster Pattern: an AND node A with a (sub)set of its child nodes CH_A such that the nodes $ch_a \in CH_A$ are terminal nodes and no other node B depends on ch_a.

```
0.6::e(a, b).   p(X, Y):-
0.8::e(b, c).     e(X, Y).
0.3::e(a, d).   p(X, Y):-
0.7::e(c, d).     e(X, X1),
0.4::e(d, e).     p(X1, Y).
0.4::e(d, f).
0.2::e(e, f).   query(p(a,f)).
```

a. A ProbLog program which encodes a probabilistic graph.

```
0.6::e(a, b).   p(d,f) :- e(d,f).
0.8::e(b, c).   p(c,f) :- e(c,d),p(d,f).
0.3::e(a, d).   p(b,f) :- e(b,c),p(c,f).
0.7::e(c, d).   p(a,f) :- e(a,b),p(b,f).
0.4::e(d, e).   p(e,f) :- e(e,f).
0.4::e(d, f).   p(d,f) :- e(d,e),p(e,f).
0.2::e(e, f).   p(a,f) :- e(a,d),p(d,f).
```

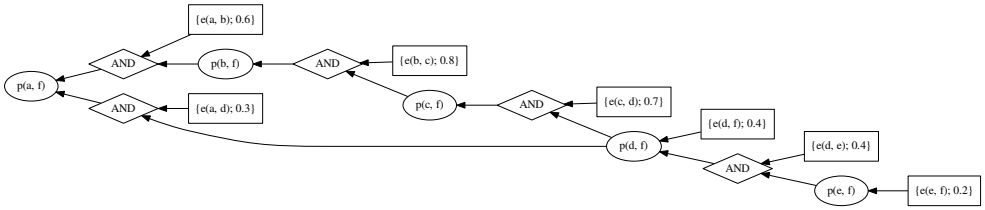
A ground ProbLog program which represents the proofs of query q = p(a,f).

Fig. 1. An example program which encodes a probabilistic graph with 7 edges.

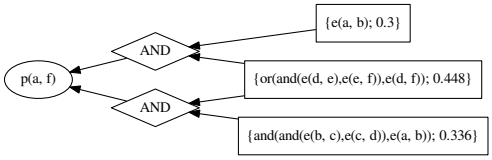
Definition 2

OR-Cluster Pattern: an OR node A with a (sub)set of its child nodes CH_A such that the nodes $ch_a \in CH_A$ are terminal nodes no other node B depends on ch_a.

Figure 1 illustrates an example program on which we apply our compaction approach. The result is shown in Figure 2.



a. Initial AND-OR graph.



b. Compacted AND-OR graph.

Fig. 2. The initial (a.) and the compacted (b.) AND-OR graphs for the ground probabilistic program in Figure 1 with respect to query q = p(a,f).

We experimented on the data set from (De Raedt et al. 2007) after excluding duplicate edges. We used both MetaProbLog and ProbLog2 running exact probabilistic inference. To determine the memory gain we use the compaction rate $C^+ = (|G_i| - |G_c|)/|G_i|$, where G_i and G_c are the initial and the compacted AND-OR graphs. On the average we observed a compaction rate C^+ of 38.69%. To judge the time gain we measured the time gain ratio $T^+ = (T_{NoC} - T_C)/T_{NoC}$ where T_{NoC} is the execution time without compaction, and T_C – with compaction. The detection and compaction time is not included. We observed: (i) a positive T^+ in 86% of the cases which terminated; (ii) the time gain differs for MetaProbLog and ProbLog2 for the former T^+ averages around 20%, for the latter around 35%; (iii) there are 4 fewer time-outs and 1 time-out more, due to compaction; (iv) the time required for (pattern) detection and compaction is polynomial to the graph size, and is consistent with the aforementioned complexity. We ought to stress that compacting

the AND-OR graph doesn't always lead to a decrease of the execution time. That is because the next steps use heuristics which depend on the order and connectivity of the variables (nodes in the AND-OR graph).

Although our algorithm is incomplete, experimentally compared to (Mantadelis and Janssens 2010) and (Mantadelis 2012, Chapter 5) it shows similar efficiency gains. Related work such as (Gogate and Domingos 2010; Hintsanen 2007; Dechter and Mateescu 2007) also exploits the model structure to improve the efficiency of the inference algorithm.

References

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

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

Darwiche, A. and Marquis, P. 2002. A knowledge compilation map. *Journal of Artificial Intelligence Research 17*, 229–264.

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

Dechter, R. and Mateescu, R. 2007. And/or search spaces for graphical models. *Artificial Intelligence 171,* 2-3, 73–106.

Fierens, D., Broek, G. V. D., Renkens, J., Shterionov, D., Gutmann, B., Thon, I., Janssens, G., and de Raedt, L. 2013. Inference and learning in probabilistic logic programs using weighted boolean formulas. *Theory and Practice of Logic Programming, Special Issue on Probability, Logic and Learning*.

Fierens, D., Van den Broeck, G., Thon, I., Gutmann, B., and De Raedt, L. 2011. Inference in probabilistic logic programs using weighted cnf's. In *UAI*. 211–220.

Gogate, V. and Domingos, P. 2010. Formula-based probabilistic inference. In *UAI 2010, Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence*. AUAI Press, 210–219.

Hintsanen, P. 2007. The most reliable subgraph problem. In *PKDD 2007: Proceedings of the 11th European conference on Principles and Practice of Knowledge Discovery in Databases*. Springer-Verlag, Berlin, Heidelneg, 471–478.

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

Mantadelis, T. 2012. Efficient algorithms for prolog based probabilistic logic programming. Ph.D. thesis, Arenberg Doctoral School of Science, Engineering & Technology, KULeuven.

Mantadelis, T. and Janssens, G. 2010. Variable compression in problog. In *Proceedings of the 17th international conference on Logic for programming, artificial intelligence, and reasoning*. LPAR'10. Springer-Verlag, Berlin, Heidelberg, 504–518.

Muise, C., Mcilraith, S. A., Beck, J. C., and Hsu, E. 2012. DSHARP: Fast d-DNNF compilation with sharpSAT. In *Canadian Conference on Artificial Intelligence*. Canadian Conference on Artificial Intelligence.

Sato, T. and Kameya, Y. 1997. Prism: a language for symbolic-statistical modeling. In *In Proceedings of the 15th International Joint Conference on Artificial Intelligence*. 1330–1335.

Shterionov, D., Mantadelis, T., and Janssens, G. June, 2013. Pattern-based compaction for problog inference. http://www.cs.kuleuven.be/publicaties/rapporten/cw/cw639.pdf. Tech. Rep. CW639, KULeuven.

Somenzi, F. 2005. Cudd: Colorado university decision diagram package, programmer's manual. http://vlsi.colorado.edu/ fabio/cudd/.