Evaluating Inference Algorithms for the Prolog Factor Language

Tiago Gomes and Vítor Santos Costa

CRACS & INESC TEC, Faculty of Sciences, University of Porto
Rua do Campo Alegre, 1021/1055, 4169-007 Porto, Portugal
{tgomes,vsc}@fc.up.pt

Abstract. Over the last years there has been some interest in models that combine first-order logic and probabilistic graphical models to describe large scale domains, and in efficient ways to perform inference on these domains. Prolog Factor Language (PFL) is a extension of the Prolog language that allows a natural representation of these first-order probabilistic models (either directed or undirected). PFL is also capable of solving probabilistic queries on these models through the implementation of four inference algorithms: variable elimination, belief propagation, lifted variable elimination and lifted belief propagation. We show how these models can be easily represented using PFL and then we perform a comparative study between the different inference algorithms in four artificial problems.

1 Introduction

Over the last years there has been some interest in models that combine first-order logic and probabilistic graphical models to describe large scale domains, and in efficient ways to perform inference and learning using these models [1,2]. A significant number of languages and systems has been proposed and made available, such as Independent Choice Logic [3], PRISM [4,5], Stochastic Logic Programs (ICL) [6], Markov Logic Networks (MLNs) [7], CLP($\mathcal{BN}$) [8,9], ProbLog [10,11], and LPADs [12], to only mention a few. These languages differ widely, both on the formalism they use to represent structured knowledge, on the graphical model they encode, and on how they encode it. Languages such as ICL, Prism, or ProbLog use the distribution semantics to encode probability distributions. MLNs encode relationships through first-order formulas, such that the strength of a true relationship serves as parameter to a corresponding ground markov network. Last, languages such as CLP($\mathcal{BN}$) approach the problem in a more straightforward way, by using the flexibility of logic programming as a way to quickly encode graphical models.

Research in Probabilistic Logic Languages has made it very clear that it is crucial to design models that can support efficient inference. One of the most exciting developments toward this goal has been the notion of lifted inference [13,14]. The idea is to take advantage of the regularities in structured models and perform a number of operations in a fell swoop. The idea was first proposed as an

F. Riguzzi and F. Železný (Eds.): ILP 2012, LNAI 7842, pp. 74–85, 2013.

extension of variable elimination [13,14], and has since been applied to belief propagation [15,16] and in the context of theorem proving and model counting [17,18].

Most work in probabilistic inference computes statistics from a sum of products representation, where each element is known as a *factor*. Lifted inference approaches the problem by generalizing factors through the notion of *parametric factor*, commonly called *parfactor*. Parfactors can be seen as templates, or classes, for the actual factors found in the inference process. Lifted inference is based on the idea of manipulating these parfactors, thus creating intermediate parfactors in the process, and in general delaying as much as possible the use of fully instantiated factors.

Parfactors are a compact way to encode distributions and can be seen as a natural way to express complex distributions. This was recognized in the Bayesian Logic Inference Engine (BLOG) [19], and more recently has been the basis for proposals such as Relational Continuous Models [20], and parametrized randvars (random variables) [21]. In this same vein, and in the spirit of CLP($\mathcal{BN}$) we propose an extension of Prolog designed to support parfactors, the *Prolog Factor Language (PFL)*.

The PFL aims at two goals. First, we would like to use the PFL to understand the general usefulness of lifted inference and how it plays with logical and probabilistic inference. Second, we would like to use the PFL as a tool for multi-relational learning. In this work, we focus on the first task. The work therefore introduces two contributions: a new language for probabilistic logical inference, and an experimental evaluation of a number of state-of-the-art inference techniques.

The paper is organized as follows. First, we present the main ideas of the PFL. Second, we discuss how the PFL is implemented. Third, we present a first experimental evaluation of the PFL and draw some conclusions.

2 The Prolog Factor Language (PFL)

First, we briefly review parfactors. We define a parfactor as a tuple of the form:

$$< A, C, \phi >$$

where A is a set of atoms (atomic formulas), C is a set of constraints, and ϕ defines a potential function on $\mathbb{R}_0^+$. Intuitively, the atoms in A describe a set of random variables, the constraints in the C describe the possible instances for those random variables, and the ϕ describe the potential values. The constraints in C apply over a set of logical variables L in A.

As an example, a parfactor for the MLN language could be written as:

$$2.33 : Smokes(X) \wedge X \in \{\texttt{John}, \texttt{Mary}, \texttt{William}\}$$

in the example, $A = \{Smokes(X)\}$, and $L = \{X\}$. Each MLN factor requires a single parameter, in this case 2.33. The constraint $X \in \{\texttt{John}, \texttt{Mary}, \texttt{William}\}$ defines the possible instances of this parfactor, in the example the three factors:

$$2.33 : Smokes(\texttt{John}) \wedge 2.33 : Smokes(\texttt{Mary}) \wedge 2.33 : Smokes(\texttt{William})$$

Notice that all factors share the same potential values.

The main goal of the PFL is to enable one to use this compact representation as an extension of a logic program. The PFL inherits from previous work in CLP($\mathcal{BN}$), which in turn was motivated by prior work on probabilistic relational models (PRMs) [22]. A PRM uses a Bayesian network to represent the joint probability distribution over fields in a relational database. Then, this Bayesian network can be used to make inferences about missing values in the database. In Datalog, missing values are represented by Skolem constants; more generally, in logic programming missing values, or existentially-quantified variables, can be represented by terms built from Skolem functors. CLP($\mathcal{BN}$) represents the joint probability distribution as a function over terms constructed from the Skolem functors in a logic program. Thus, in CLP($\mathcal{BN}$), we see random variables as a special interpretation over a set V of function symbols, the skolem variables.

The PFL. The first insight of the PFL is that CLP($\mathcal{BN}$) skolem functions naturally map to atoms in parfactor formulae. That is, in a parfactor the formula $a(X) \wedge b(X)$ can be seen as identifying two different skolem functions $X \rightarrow a(X)$ and $X \rightarrow b(X)$. The second observation is that both the constraints and the potential values can be obtained from a logic program. Thus, an example of a parfactor described by PFL is simply:

```
bayes ability(K)::[high,medium,low] ;
    [0.50, 0.40, 0.10] ;
    [professor(K)].
```

This parfactor defines an atom `ability(K)`, within the context of a directed network, with $K \in \{professor(K)\}$ as the constraint. The random variables instantiated by `ability(K)` will have `high`, `medium` and `low` as their domain.

More precisely, the PFL syntax for a factor is

$$Type\ F;\ \phi;\ C.$$

Thus, a PFL factor has four components:

- *Type* refers the type of the network over the parfactor is defined. It can be `bayes`, for directed networks, or `markov`, for undirected ones.
- F is a sequence of Prolog terms that define sets of random variables under the constraints in C. Each term can be seen as the signature of a skolem function whose arguments are given by the unbound logical variables. The set of all logical variables in F is named L. The example includes a single term, $ability(K)$; the only logical variable is K.
- The table ϕ is either a list of potential values or a call to a Prolog goal that will unify its last argument with a list of potential values.
- C is a list of Prolog goals that will impose bindings on the logical variables in L. In other words, the successful substitutions for the goals in C are the valid values for the variables in L. In the example, the goals constrain K to match a professor.

A more complex example is a parfactor for a student's grade in a course:

```
bayes grade(C,S)::[a,b,c,d], intelligence(S), difficulty(C) ;
      grade_table ;
      [registration(_,C,S)].
```

In the example, the `registration/3` relation is part of the extensional data-base.

The next example shows an encoding for the competing workshops problem [23]:

```
markov attends(P), hot(W) ;
       [0.2, 0.8, 0.8, 0.8] ;
       [c(P,W)].

markov attends(P), series ;
       [0.501, 0.499, 0.499, 0.499] ;
       [c(P,_)].
```

One can observe that the model in this case is undirected. The encoding defines two parfactors: one connects workshop attendance with the workshop being hot, the other with the workshop being a series. In this case, the domains of the random variables are not being specified. Thus, they will default to boolean.

2.1 Querying and Generating Evidence for the PFL

In our approach, each random variable in PFL is implicitly defined by a predicate with the same name and an extra argument. More formally, the value V for a random variable or skolem function $R(A_1, \ldots, A_n)$, is given by the predicate $R(A_1, \ldots, A_n, V)$. Thus in order to query a professor's ability it is sufficient to ask:

```
?- ability(p0, V).
```

This approach follows in the lines of PRISM [4] and CLP($\mathcal{BN}$).

Evidence can be given as facts for the intentional predicates. Hence,

```
ability(p0,high).
```

can be used to indicate that we have evidence on `p0`'s (high) ability. Conditional evidence can also be given as part of a query, hence:

```
?- ability(p0,high), pop(p0,X).
```

would return the marginal probability distribution for professor `p0`'s popularity given he has a high ability.

3 Inference in the PFL

One of our main motivations in designing the PFL is to research on the interplay between logical and probabilistic inference. Indeed, how to execute a PFL program very much depends on the probabilistic inference method used.

Solving the main inference tasks in first-order probabilistic models can be done by first grounding the network and then applying a ground solver, such as variable elimination or belief propagation. However, the cost of this operation will strongly depend on the domain size (number of objects). It may be much more efficient to solve this problem in a *lifted* way, that is, by exploiting the repeated structure of the model to speed up inference by potentially several orders of magnitude.

The PFL thus supports both lifted and grounded inference methods. In *fully grounded solving* the PFL implementation creates a ground network and then calls the solver to obtain marginals. In *lifted solving* the PFL implementation first finds out a graph of parfactors that are needed to address the current query, an then calls the lifted solver.

The fully grounded algorithm implements a transitive closure algorithm (similar to [24]), and is as follows. First, given a query goal Q, it obtains the corresponding random variable $\{V\}$. It also collects the set of evidence variables E. The algorithm then maintains two sets: an open set of variables, initially $O \leftarrow E \cup \{V\}$, and an explored set of variables X, initially empty ($X \leftarrow \{\}$). It proceeds as follows:

1. It selects a variable V from the open-set (O) and removes it from O;
2. Adds V to X;
3. For all parfactors *defining* the variable V, it computes the constraints as a sequence of Prolog goals, and it adds the other random variables V' to O if $V' \notin X$;
4. It terminates when O empty.

The PFL uses the convention that in Bayesian networks a single parfactor defines each variable, and in markov networks all parfactors define the variable. This implementation is currently used by variable elimination, belief propagation and counting belief propagation, and the CLP($\mathcal{BN}$) solvers [25].

Lifted solving performs a first step of computing transitive closure, but only on the graph of parfactors. No constraints are called, and no grounding is made. A second step then computes the extension of the constraints, so the groundings are kept separate from the graph. This implementation is used by lifted variable elimination.

The probabilistic inference algorithms used in the PFL are discussed next.

4 Probabilistic Inference Algorithms

A typical inference task is to compute the marginal probabilities of a set of random variables given the observed values of others (evidence). Variable elimination (VE) and belief propagation (BP) are two popular ways to solve this problem.

Next we briefly describe these two algorithms as well as their lifted versions.

4.1 Variable Elimination

Variable elimination [26] is one of the simplest exact inference methods. As the name indicates, it works by successively eliminating the random variables that appears in the factors until only the query variable remains.

At each step, each variable X is eliminated by first collecting the factors where X appears, then calculating the product of these factors and finally summing out X. The product of two factors works like a join operation in relational algebra, where the set of random variables of the resulting factor is the union of the random variables of the two operands, and the product is performed such that the values of common random variables match with each other. Summing out a variable X is done through the summation of all probabilities where the values of X varies, while the values of the other variables remains fixed.

Unfortunately, the cost of variable elimination is exponential in relation to the *treewidth* of the graph. The treewidth is the size of the largest factor created during the operation of the algorithm using the best order in which the variables are eliminated. And it gets worse: finding the best elimination order is a NP-Hard problem.

4.2 Belief Propagation

Belief propagation [27] is a very efficient way to solve probabilistic queries in models where exact inference is intractable. Here we describe the implementation of the algorithm over factor graphs. The algorithm consists in iteratively exchanging local messages between variable and factor nodes of a factor graph, until convergence be achieved (the probabilities for each variable stabilize). These messages consists in vectors of probabilities that measures the influence that variables have among others.

Belief propagation is known to converge to the exact answers where the factor graph is acyclic. Otherwise, there is no guarantee on the convergence of the algorithm, or that the results will be good approximations of the exact answers. However, experimental results show that often it converges to good approximations and can finish several times faster than other methods [28].

Next, we describe how the messages are calculated and how to obtain the marginals for each variable. The message from a variable X to a factor f is defined by:

$$u_{X \to f}(x) = \prod_{g \in \text{neighbors}(X) \setminus \{f\}} u_{g \to X}(x) \ , \tag{1}$$

that is, it consists in the product of the messages received from X's other neighbor factors. The message that a factor f sends to a variable X is defined by:

$$u_{f \to X}(x) = \sum_{y_1, \ldots, y_k} f(x, y_1, \ldots, y_k) \prod_{i=1}^{k} u_{Y_i \to f}(y_i) \ , \tag{2}$$

where $Y_1, \ldots, Y_k$ are other f's neighbor variables. In other words, it is the product of the messages received from other f's neighbor variables, followed by summing out all variables except X.

Finally, we estimate the marginal probabilities for a variable X by computing the product of the messages received by all X's neighbor factors:

$$P(x) \propto \prod_{f \in \text{neighbors}(X)} u_{f \to X}(x) \ . \tag{3}$$

Depending on the graphical model, we may also need to normalize the probabilities. That is, to scale them to sum to 1. In our implementation, initially all messages are initialized in a uniform way.

4.3 Lifted Variable Elimination

Lifted variable elimination exploits the symmetries present in first-order probabilistic models, so that it can apply the same principles behind variable elimination to solve a probabilistic query without grounding the model. However, instead of summing out a random variable at a time, it works out by summing out a whole group of interchangeable random variables.

Initially a shattering operation is applied in all of the parfactors. Shattering consists in splitting the parfactors until all atoms represent under the constraint either identical or disjoint sets of ground random variables. Intuitively, this is necessary to ensure that the same reasoning steps are applied to all ground factors represented by a parfactor. Sometimes it may also be necessary to further split the parfactors for some lifted operation be correct, that is, equal to a set of multiple ground operations.

Work on lifting variable elimination started with Poole [13] and was later extended by de Salvo [29]. Milch [23] increased the scope of lifted inference by introducing counting formulas, that can be seen as a compact way to represent a product of repeated ground factors. The current state of art on lifted variable elimination is GC-FOVE [21]. GC-FOVE extends previous work by allowing more flexibility about how the groups of interchangeable random variables are defined.

4.4 Lifted Belief Propagation

There are currently two main approaches for applying belief propagation in a lifted level: lifted first-order belief propagation [15] and counting belief propagation [16]. While the first requires a first-order probabilistic model as input, the second does not (is a generalization of the first). Here we focus only on counting belief propagation.

Counting belief propagation is defined over factor graphs and employs two steps. Firstly, variable and factor nodes that are indistinguishable in terms of messages sent and received are respectively grouped in clusters of variables and clusters of factors, creating with this process a compressed factor graph. To

achieve this, counting belief propagation simulates the use of belief propagation by using a color based scheme, where colors are transmitted instead of real probabilities and identical colors identify identical messages.

Secondly, an adapted version of belief propagation is executed over the compressed factor graph. Adapted because it needs to consider that an edge in the compressed factor graph can represent multiples edges in the original factor graph.

As the size of compressed factor graph can be a lot smaller than the original factor graph, the time required to solve some query can substantially decrease.

5 Experimental Evaluation

We compare the performance of variable elimination, belief propagation and their lifted versions in four benchmark problems described using PFL: *workshop attributes* [23], *competing workshops* [23], *city* [13] and *social domain* [30].

Our implementation of lifted belief propagation is based on counting belief propagation (CBP), while the implementation of lifted variable elimination corresponds to GC-FOVE with the difference that we use a simple tree to represent the constraints instead of a constraint tree [21]. All algorithms are written in `C++` and are available with the YAP Prolog system[1] [31].

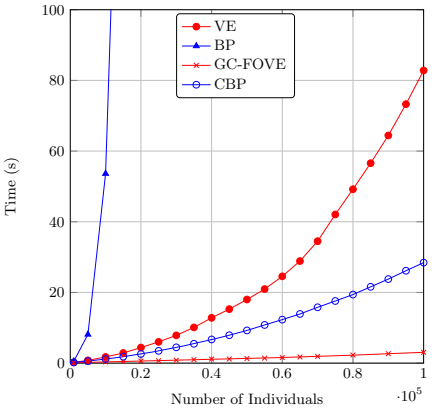
The test environment was a machine with 2 Intel® Xeon® X5650@2.67GHz processors and 99 Gigabytes of main memory, running a Linux kernel 2.6.34.9-69.fc13.x86_64. The times for solving each query presented here are the minimum times of a series of multiple runs. All tests are deterministic.

Figure 1 displays the running times to solve a query on *series* for both workshop attributes and competing workshops problems with an increased number of individuals [23]. GC-FOVE is the method that performs better, followed by CBP. For CBP, the size of the compressed factor graph remains constant as we increase the number of individuals. Hence its cost is mostly dominated by the time used to compress the factor graph. Variable elimination performs reasonable well even in the presence of large domains, as the treewidth remains constant for all number of individuals. Although it converges in few iterations, belief propagation is clearly the slowest method.

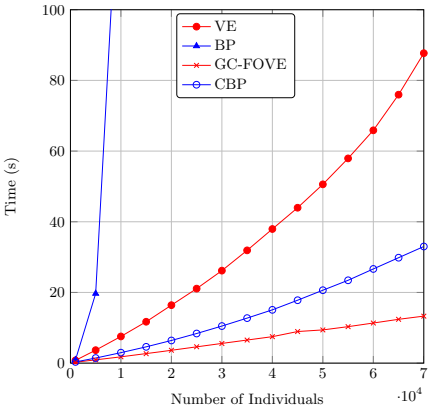
Figure 2(a) shows the running times to solve a query on *guilty* for some individual in the city problem, given the descriptions of the others [13]. Here the performances of CBP and GC-FOVE are close, as well as their non-lifted versions. More interesting is the social domain problem [30] where non-lifted exact inference is intractable, since the treewidth of the graph increases exponentially in relation to the number of individuals. Figure 2(b) displays the performances of belief propagation, GC-FOVE and CBP for a query on *friends*. We found that GC-FOVE can solve very efficiently this problem and in a faster way than CBP.

At last, we are interested in study how evidence causes the parfactors to be split for GC-FOVE and how it influences the size of the compressed factor graph for CBP. For this experiment we took the social domain network, fixed

[1] Available from `http://yap.git.sourceforge.net/git/gitweb.cgi?p=yap/yap-6.3`

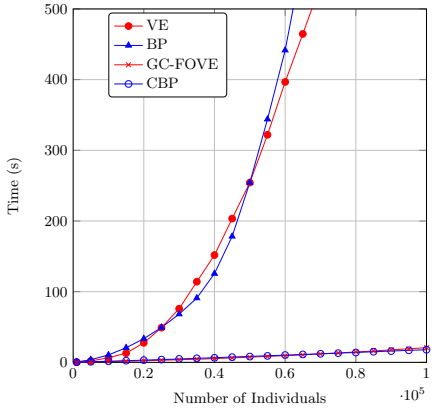


(a) Workshop attributes

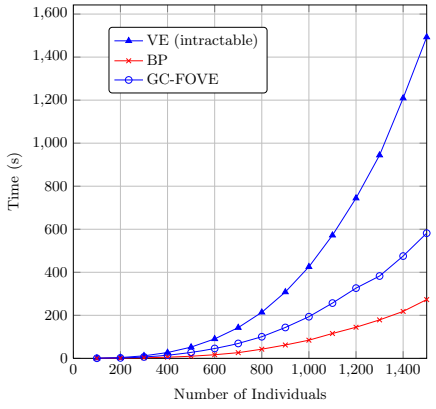


(b) Competing workshops

Fig. 1. Performance on workshops attributes and competing workshops problems with an increased number of individuals



(a) City



(b) Social domain

Fig. 2. Performance on city and social domain problems with an increased number of individuals

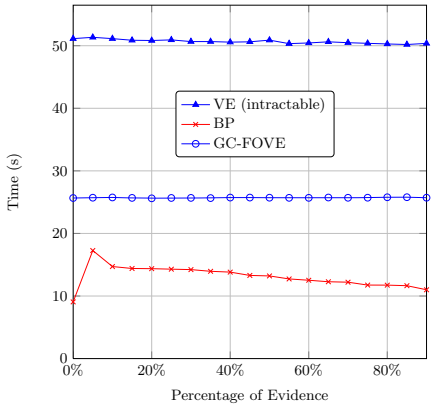


Fig. 3. Performance on social domain varying the percentage of evidence

the number of individuals at 500 and varied the percentage of evidence on random variables produced by *smokes* (observed random variables as well as their observed values were choose randomly). The results are show in Figure 3. The run times for belief propagation and CBP remain stable for all different percentages of evidence. Therefore the size of the compressed factor graph does not considerably increase for CBP. For GC-FOVE, the first set of evidence causes an increase on the run time as some parfactors will be split. However adding more evidence will not cause further splitting and instead the run times will decrease gradually, since the absorption of the evidence will cause some operations used by GC-FOVE to become slightly cheaper.

6 Conclusions and Future Work

We introduce the PFL, an extension of Prolog to manipulate complex distributions represented as products of factors. The PFL has been implemented as an extension to the YAP Prolog system, and is publicly available in YAP's development version. As an initial experiment, we use the PFL to represent several lifted inference benchmarks, and compare a number of inference techniques. Our results show benefit in using first-order variable elimination, even given the complexity of the algorithm.

We see the PFL as a tool in experimenting with ways to combine probabilistic and logical inference, and plan to continue experimenting with different approaches, such as other forms of lifted belief propagation [15]. We also plan to look in more detail to aggregates. Last, and not least, our ultimate focus is to be able to learn PFL programs.

Acknowledgments. This work is funded (or part-funded) by the ERDF - European Regional Development Fund through the COMPETE Programme (operational programme for competitiveness) and by National Funds through the

FCT - Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within project HORUS (PTDC/EIA/100897/2008) and project LEAP (PTDC/EIA-CCO/112158/2009).

References

1. Getoor, L., Taskar, B.: Introduction to Statistical Relational Learning. MIT Press (2007)
2. De Raedt, L., Frasconi, P., Kersting, K., Muggleton, S. (eds.): Probabilistic Inductive Logic Programming. LNCS (LNAI), vol. 4911. Springer, Heidelberg (2008)
3. Poole, D.: The Independent Choice Logic for Modelling Multiple Agents Under Uncertainty. Artif. Intell. 94(1-2), 7–56 (1997)
4. Sato, T., Kameya, Y.: PRISM: A Language for Symbolic-Statistical Modeling. In: Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, IJCAI 1997, Nagoya, Japan, August 23-29, vols. 2, pp. 1330–1339. Morgan Kaufmann (1997)
5. Sato, T., Kameya, Y.: New Advances in Logic-Based Probabilistic Modeling by PRISM. In: [2], pp. 118–155
6. Muggleton, S.: Stochastic Logic Programs. In: De Raedt, L. (ed.) Advances in Inductive Logic Programming. Frontiers in Artificial Intelligence and Applications, vol. 32, pp. 254–264. IOS Press, Amsterdam (1996)
7. Richardson, M., Domingos, P.: Markov logic networks. Machine Learning 62(1-2), 107–136 (2006)
8. Santos Costa, V., Page, D., Qazi, M., Cussens, J.: CLP(BN): Constraint Logic Programming for Probabilistic Knowledge. In: Meek, C., Kjærulff, U. (eds.) Proceedings of the 19th Conference in Uncertainty in Artificial Intelligence, UAI 2003, Acapulco, Mexico, August 7-10, pp. 517–524. Morgan Kaufmann (2003)
9. Santos Costa, V., Page, D., Cussens, J.: CLP(*BN*): Constraint Logic Programming for Probabilistic Knowledge. In: [2], pp. 156–188
10. De Raedt, L., Kimmig, A., Toivonen, H.: ProbLog: A Probabilistic Prolog and Its Application in Link Discovery. In: Veloso, M.M. (ed.) Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI 2007, Hyderabad, India, January 6-12, pp. 2462–2467 (2007)
11. Kimmig, A., Demoen, B., De Raedt, L., Santos Costa, V., Rocha, R.: On the implementation of the probabilistic logic programming language ProbLog. TPLP 11(2-3), 235–262 (2011)
12. Riguzzi, F., Swift, T.: The PITA system: Tabling and answer subsumption for reasoning under uncertainty. TPLP 11(4-5), 433–449 (2011)
13. Poole, D.: First-order probabilistic inference. In: Gottlob, G., Walsh, T. (eds.) IJCAI, pp. 985–991. Morgan Kaufmann (2003)
14. de Salvo Braz, R., Amir, E., Roth, D.: Lifted First-Order Probabilistic Inference. In: Kaelbling, L.P., Saffiotti, A. (eds.) IJCAI, pp. 1319–1325. Professional Book Center (2005)
15. Singla, P., Domingos, P.: Lifted first-order belief propagation. In: Proceedings of the 23rd National Conference on Artificial Intelligence, vol. 2, pp. 1094–1099 (2008)
16. Kersting, K., Ahmadi, B., Natarajan, S.: Counting belief propagation. In: Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, pp. 277–284. AUAI Press (2009)

17. Gogate, V., Domingos, P.: Probabilistic Theorem Proving. In: Cozman, F.G., Pfeffer, A. (eds.) Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence, UAI 2011, Barcelona, Spain, July 14-17, pp. 256–265. AUAI Press (2011)
18. Van den Broeck, G., Taghipour, N., Meert, W., Davis, J., De Raedt, L.: Lifted Probabilistic Inference by First-Order Knowledge Compilation. In: Walsh, T. (ed.) Proceedings of the 22nd International Joint Conference on Artificial Intelligence, IJCAI 2011, Barcelona, Catalonia, Spain, July 16-22, pp. 2178–2185. IJCAI/AAAI (2011)
19. Milch, B., Marthi, B., Russell, S.J., Sontag, D., Ong, D.L., Kolobov, A.: BLOG: Probabilistic Models with Unknown Objects. In: Kaelbling, L.P., Saffiotti, A. (eds.) Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, IJCAI 2005, Edinburgh, Scotland, UK, July 30-August 5, pp. 1352–1359. Professional Book Center (2005)
20. Choi, J., Amir, E., Hill, D.J.: Lifted Inference for Relational Continuous Models. In: Grünwald, P., Spirtes, P. (eds.) Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence, UAI 2010, Catalina Island, CA, USA, July 8-11, pp. 126–134. AUAI Press (2010)
21. Taghipour, N., Fierens, D., Davis, J., Blockeel, H.: Lifted variable elimination with arbitrary constraints. In: Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics (2012)
22. Getoor, L., Friedman, N., Koller, D., Pfeffer, A.: Learning Probabilistic Relational Models. In: Relational Data Mining, pp. 307–335. Springer (2001)
23. Milch, B., Zettlemoyer, L., Kersting, K., Haimes, M., Kaelbling, L.: Lifted probabilistic inference with counting formulas. In: Proc. 23rd AAAI, pp. 1062–1068 (2008)
24. Kersting, K., De Raedt, L.: Bayesian logic programs. CoRR cs.AI/0111058 (2001)
25. Santos Costa, V.: On the Implementation of the CLP($\mathcal{BN}$) Language. In: Carro, M., Peña, R. (eds.) PADL 2010. LNCS, vol. 5937, pp. 234–248. Springer, Heidelberg (2010)
26. Zhang, N.L., Poole, D.: Exploiting causal independence in bayesian network inference. Journal of Artificial Intelligence Research 5, 301–328 (1996)
27. Kschischang, F., Frey, B., Loeliger, H.: Factor graphs and the sum-product algorithm. IEEE Transactions on Information Theory 47(2), 498–519 (2001)
28. Murphy, K., Weiss, Y., Jordan, M.: Loopy belief propagation for approximate inference: An empirical study. In: Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence, pp. 467–475. Morgan Kaufmann Publishers Inc. (1999)
29. de Salvo Braz, R., Amir, E., Roth, D.: Lifted first-order probabilistic inference. In: Getoor, L., Taskar, B. (eds.) Introduction to Statistical Relational Learning, pp. 433–451. MIT Press (2007)
30. Jha, A.K., Gogate, V., Meliou, A., Suciu, D.: Lifted Inference Seen from the Other Side: The Tractable Features. In: Lafferty, J.D., Williams, C.K.I., Shawe-Taylor, J., Zemel, R.S., Culotta, A. (eds.) NIPS, pp. 973–981. Curran Associates, Inc. (2010)
31. Santos Costa, V., Damas, L., Rocha, R.: The YAP Prolog system. Theory and Practice of Logic Programming 12(Special Issue 1-2), 5–34 (2012)