Constraint-Based Inference in Probabilistic Logic Programs[⋆] (Extended Abstract)

Arun Nampally, C. R. Ramakrishnan

Department of Computer Science, Stony Brook University, Stony Brook, NY 11794
{anampally,cram}@cs.stonybrook.edu

1 Introduction

A wide variety of models that combine logical and statistical knowledge can be expressed succinctly in the Probabilistic Logic Programming (PLP) paradigm. Specifically, models in standard statistical formalisms such as probabilistic graphical models (PGMs) (e.g. Bayesian Networks), can be easily encoded as PLP programs. For instance, Fig. 1(a) shows a program in PRISM, a pioneering PLP language [16]. A widget, represented by a random variable X, is tested by two different processes b_1 and b_2. The outcomes of these tests are represented by random variables Y and Z, respectively. In PRISM, a special predicate of the form `msw(a,X)` associates random variable X with a random process `a`. Consider the problem of determining the distribution of X given that Y and Z are identical. Note that evidence is defined as a constraint over instantiations of the random variables, in contrast to a *specific* instantiation as in traditional PGMs. However, such evidence can be easily specified in PLP (see predicate `e/0`). The probability of a specific instantiation of X can also be computed based on a PLP query (e.g. `q(1)` using predicate `q/1`). This simple example illustrates how logical clauses can be used to specify evidence and queries in PLP that go beyond what is possible in traditional PGMs.

The Driving Problem. The expressiveness of PLP comes at a cost. Since PLP is an extension to traditional logic programming, inference in PLP is undecidable in general. Probabilistic inference for a large class of statistical models (e.g. Bayesian networks) is intractable. Even problems for which inference is tractable can be encoded in multiple ways in PLP, with different inference complexity. For instance, consider the PRISM program in Fig. 1(b). In that program, `genlist/2` defines a list of the outcomes of N identically distributed random variables ranging over {`a,b`}. Predicate `palindrome/1` tests, using a definite clause grammar definition, if a given list is a palindrome; and `count_as/2` tests if a given list contains k (not necessarily consecutive) "`a`"s. Using these predicates, consider the inference of the conditional probability of `query`(n, k) given `evidence`(n): *i.e.*, the probability that an n-element palindrome has k "`a`"s.

⋆ This research was supported in part by NSF grants CCF-1018459 and IIS-1447549.

```
% Model: Y and Z depend on X.
w(X,Y,Z) :-
    msw(a, X),
    msw(b1(X), Y),
    msw(b2(X), Z).
% Evidence: Y and Z are same.
e :- w(_,S,S).
% Query: value of X.
q(X) :- w(X,_,_).
% Domains:
values(a, [1,2]).
values(b1(_), [1,2,3]).
values(b2(_), [2,3,4]).
% Distribution parameters:
set_sw(a, [0.4,0,6]).
set_sw(b1(1), [0.1,0.3,0.6]).
set_sw(b1(2), [0.2,0.4,0.4]).
set_sw(b2(1), [0.5,0.3,0.2]).
set_sw(b2(2), [0.6,0.1,0.3]).
```

(a) Bayesian Network PLP

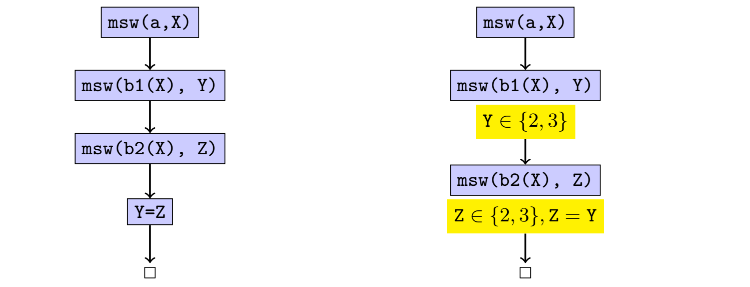
```
% generate a list of N random variables.
genlist(N, L) :- (N=0 -> L= []
                       ; msw(flip, N, X),
                         L = [X|L1], N1 is N-1,
                         genlist(N1, L1) ).
% Evidence: string is a palindrome.
evidence(N) :- genlist(N, L), palindrome(L).
% Query: string has K 'a's
query(N, K) :- genlist(N, L), count_as(L, K).
% Check if a given list is a palindrome
palindrome(L) :- phrase(palindrome, L).
palindrome --> [].
palindrome --> [_X].
palindrome --> [X], palindrome, [X].
% Query condition:
count_as([], 0).
count_as([X|Xs], K) :-
    K > 0, (X=a -> L is K-1; L=K),
    count_as(Xs, L).
% Domains:
values(flip, [a,b]).
% Distribution parameters:
set_sw(flip, [0.5, 0.5]).
```

(b) Palindrome PLP

Fig. 1: Examples of PLPs

The conditional probability is well-defined according to PRISM's distribution semantics [17]. However, *the PRISM itself will be unable to correctly compute the conditional query's probability*, since the conditional query, as encoded above, will violate the PRISM system's assumptions of independence among random variables used in an explanation. It should be noted that the above conditional probability may be efficiently inferred by transforming the "generate-and-test" program to one where the tests are folded into the generation phase. However, such transformations are dependent on the encoding of the query and evidence predicates, and are hard to generalize. Moreover, while the probability of goal `evidence(N)` can be computed in linear time (by exploiting sharing in explanation graphs), the size of the explanation graph for goal `query(N)` may be exponential in N when the subgoals in the explanations are placed in the order in which they are encountered.

Approximate inference based on rejection sampling performs poorly, rejecting a vast number of generated samples, since the likelihood of a string being a palindrome decreases exponentially in N. Alternatives such as Metropolis-Hastings-based Markov Chain Monte Carlo (MCMC) techniques [9, e.g.] do not behave much better due to the fact that the chains exhibit poor convergence (mixing), since most transitions lead to strings inconsistent with evidence. Gibbs-sampling-based MCMC [8] cannot be readily applied since the dependencies between random variables are hidden in the program and not explicit in the model.



(a) Before Constraint Propagation (b) After Constraint Propagation

Fig. 2: Symbolic Derivation for evidence "e" in BN Example

Our Approach. We identify two basic problems that contribute to the difficulty of inference in PLPs. First is that the random variable dependencies are not explicit in the program but may vary based on the program's control and data flow. The second is that evidence (and query) specifications may be complex rendering it difficult to predict whether a variable's valuation will be consistent with the evidence (or lead to query's success). We use *constraint propagation* both to uncover the hidden dependencies and to predict consistency with evidence. We explicitly construct *symbolic* derivations that abstract actual valuations of random variables and use a graphical structure to represent the derivations. We then provide inference algorithms, both approximate and exact, that compute the probability of a (possibly conditional) query based on this graphical structure.

Summary of Contributions. This paper describes a novel technique that addresses the problem of scalability of inference in PLPs.

1. The paper introduces a structure called an Ordered Symbolic Derivation Diagram to represent succinctly the set of possible derivations for a PLP query or evidence (Section 2).
2. The paper presents a likelihood-weighted sampling method based on OSDDs that can be used for approximate inference (Section 3).
3. The paper also presents an exact inference algorithm that operates directly on OSDDs. While this algorithm has relatively narrow applicability, it provides a powerful way to infer over large problem sizes without enumerating random variable valuations (Section 3).

We present experimental results which show the effectiveness of OSDD-based inference methods, as well as their cost (Section 4). Related work is discussed in detail in Section 5. The paper concludes with a discussion on the other uses of OSDDs for inference in PLPs.

2 Symbolic Derivations and Diagrams

In this paper, we use PRISM's syntax and distribution semantics, *but without the independence and mutual exclusion requirements on the explanations of a* *goal.* Thus we consider PRISM programs with their *intended* model-theoretic semantics, rather than that computed by the PRISM system.

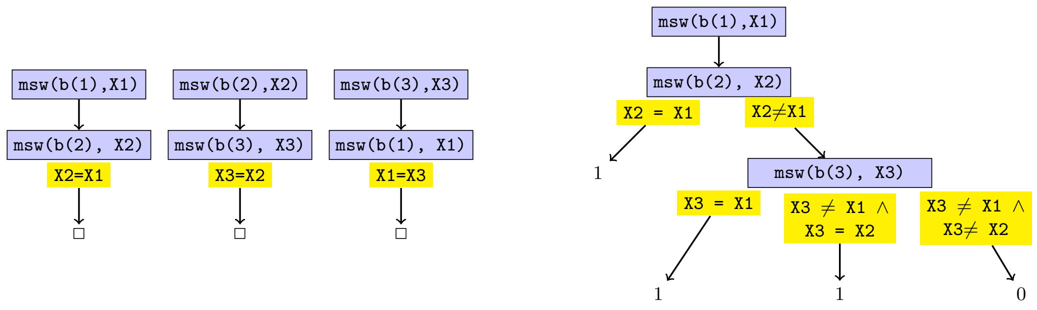
As stated in the Introduction, the dependencies between random variables implicit in the control and data flow of a PLP, and the impossibility of completely representing the set of all random variable valuations that are consistent with evidence contribute to the difficulty of inference in PLPs. To address these two problems, we devise an inference technique based on *constraint propagation*, to uncover hidden dependencies and predict evidence consistency. In the first step, we build derivations *symbolically*, instantiating random variables only when necessary. A symbolic derivation is a sequence of `msw` goals and constraints, as illustrated in Fig. 2(a) by the derivation for evidence "`e`" from example of Fig. 1(a).

In the second step, we propagate the constraints in a symbolic derivation, resulting in possible restrictions on the domains of variables. For instance, in the Bayesian Network (BN) example of Fig. 1(a), since the evidence demands $Y = Z$, the domains of b_1 and b_2 get restricted to `[2,3]`. Such constraint propagation is done using light-weight techniques such as node-consistency and arc-consistency algorithms. We add inferred domain restrictions (if any) to the derivation. We also place the constraints where their satisfaction can be effectively tested. Fig. 2(b) shows the symbolic derivation for "`e`" in the BN example after constraint propagation. The constraints denote a sufficient condition for any concrete instance of the symbolic derivation to represent a successful derivation. Symbolic derivations, parameterized by the consistency algorithms used in their construction, can be readily formalized; see [14]. Symbolic derivations can be subsequently used in a number of ways, two of which are described below.

Generalization. For many standard statistical models (e.g. PGMs) our technique will construct at most one symbolic derivation. In general, however, PLPs may have more than one symbolic derivation, as illustrated by the Birthday Collision example in Fig. 5. This example encodes the problem of determining the (unconditional) probability that two persons in a population of a given size share the same birthday. The query `same_birthday(3)`, which fixes a population of size 3, has 6 symbolic derivations, 3 of which are shown in Fig. 3(a). In such cases, we combine the set of symbolic derivations into a tree structure, called the Ordered Symbolic Derivation Diagram (OSDD), illustrated in Fig. 3(b). An OSDD is analogous to a Constraint Decision Diagram (CDD) [3]: each node defines a variable, and the outgoing edges are guarded by constraints on that variable. An OSDD is constructed based on a total order over variables, as in an Ordered Binary Decision Diagram [2]. Each path in an OSDD is a symbolic derivation. In fact, every symbolic derivation is a rudimentary OSDD (with 0-branches removed).

3 Inference Based on Symbolic Derivation Diagrams

We illustrate the process of generating likelihood-weighted samples [7,18] for goal "`e`" from its symbolic derivation. We start with likelihood weight of 1. When



(a) Selected Symbolic Derivations (b) Symbolic Derivation Diagram

Fig. 3: Symbolic Derivations for query "`same_birthday(3)`" in Birthday Collision Example

visiting `msw(a,X)`, we notice no domain restrictions on `X`, and hence bind `X` to a random sample generated from `a`'s distribution. Assume the sample we drew is `X=1`. We then visit `msw(b1(1), Y)` to sample `Y`. However, since `Y` has a domain restriction $Y \in \{2, 3\}$, we generate a sample for `Y` such that $Y \in \{2, 3\}$. This is done by picking from $\{2, 3\}$ uniformly, and multiplying the likelihood weight with the probability of the picked value. Assume we pick `Y=2`; then the current likelihood is set to 0.3, the probability of 2 in `b1(1)`'s distribution. Finally, we visit `msw(b2(1),Z)`, whose two constraints restrict `Z` to $\{2\}$. Selecting `Z=2`, we multiply the likelihood weight of the current derivation with 0.5. Thus we generate a sample (`X=1`, `Y=2`, `Z=2`) consistent with evidence `e` with a likelihood weight of 0.15.

In summary, likelihood-weighted samples are drawn by (a) independently sampling random variables whose valuations are unconstrained; (b) uniformly sampling variables whose valuations have domain constraints; and (c) computing the probability of the sample as the product of probabilities of values picked in step (b). This procedure can be readily formalized; see [14].

Exact Inference. For certain class of programs and queries, symbolic derivations can be directly used for *exact* inference. Fig. 4 shows the symbolic derivation of evidence `evidence(6)` from the Palindrome example (Fig. 1(b)). Note that only the constraints in the symbolic derivation determine whether a concrete instance succeeds. Thus, if the distribution of `flip` is uniform, the three constraints are each satisfied independently, resulting in 0.125 as the probability. Such exact computation of probabilities is formalized in terms of *measurability*; a variable X with domain constraint η is said to be measurable if the size of X's domain (consistent with η) is independent of the valuation of other variables. When a symbolic derivation diagram consists only of uniformly distributed measurable variables, then the associated probability can be computed exactly. Such exact computation is readily formalized as well; see [14].

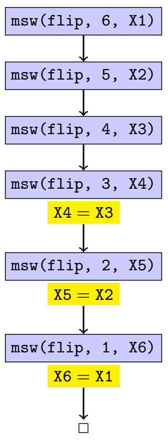


Fig. 4: Symbolic Derivation for evidence "`evidence(6)`" in Palindrome Example

```
% Two from a population of size N
% share a birthday.
same_birthday(N) :-
    person(N, P1),
    % P1's birthday is D
    msw(b(P1), D),
    person(N, P2),
    P1 \= P2,
    % and so is P2's.
    msw(b(P2), D).

person(N, P) :-
% bind P, backtracking through 1..N
    basics:for(P, 1, N).

% Distribution parameters:
set_sw(b(_), uniform(1,365)).
```

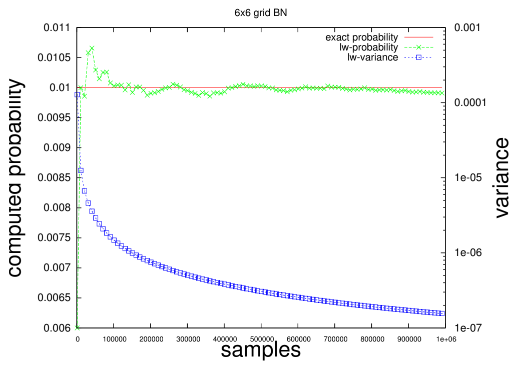
Fig. 5: Birthday Collision PLP

4 Experimental Evaluation

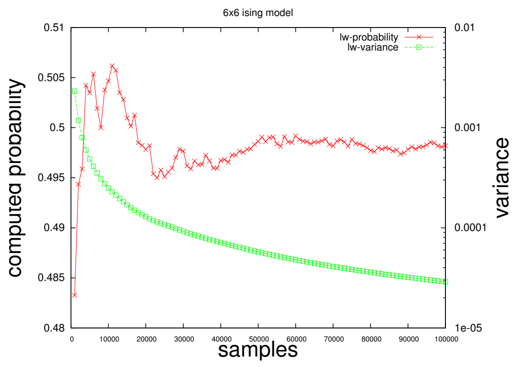
We present the results of experiments using a prototype implementation of a likelihood-weighted sampler based on symbolic derivations. The prototype uses XSB Prolog to build symbolic derivations, propagate constraints and construct OSDDs; and a few modules written in C for maintaining the sampler's state and dealing with random variable distributions. We used the following examples in the experiments.

- **Grid BN** is a Bayesian Network with Boolean random variables arranged in a 6×6 grid (with dependencies going left-to-right and top-to down). This simple structure was used to evaluate the effectiveness of our technique when the evidence probability is extremely low ($\tilde{}10^{-12}$).
- **Ising Model** is a well-known undirected graphical model. We used a 6×6 grid of Boolean random variables with factors on edges. The PRISM program independently generates values of terminal nodes of all edges, and ties them together by expressing equality constraints between shared variables of edges.
- **Palindrome**, which is shown in Fig. 1(b), with evidence limited to strings of length 20, and query checking for a string with 4 "a"s.
- **Birthday Collision**, shown in Fig. 5 (page 6), with population size of 6, i.e. query `same_birthday(6)`.

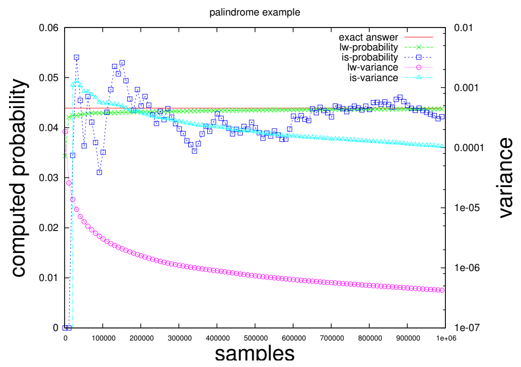
The first three examples involved conditional queries with low-likelihood evidence. The birthday collision example had an unconditional query. It should be noted that only the first example, **Grid BN**, can be evaluated in the PRISM system; the other examples have queries that violate PRISM's mutual exclusion and independence assumptions and hence cannot be directly evaluated in that



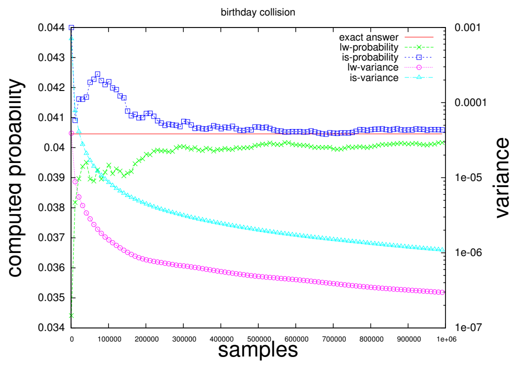
(a) BN



(b) Ising Model



(c) Palindrome



(d) Birthday

Fig. 6: Experimental Results

system. Our inference procedure, however, removes PRISM's assumptions and correctly evaluates the query probabilities for all the above examples.

The results of the experiments are shown in Fig. 6. Each subfigure plots the estimated probability and variance of the estimate(on log scale), for two samplers: the LW method described in this paper, and a simple independent sampler (with rejection sampling for conditional queries). Note that the LW sampler's results show significantly lower variance in all the examples. For the Grid BN and Ising Model, the evidence probability was low enough that a rejection sampler was unable to draw a consistent sample. The LW sampler, however, was able to converge to a reasonable estimate of low variance in about 500,000 samples. Both examples generated a single symbolic derivation. We directly sampled from this instead of materializing a OSDD structure. For the Grid BN, node consistency was sufficient to derive domain restrictions. For the Ising model, we found that standard LW sampling (picking a restricted value uniformly and assigning a likelihood weight) generated a number of samples with extremely low weights. Instead the probabilites of the set of allowed values were normalized to create a new proposal distribution. This resulted in generating samples with higher likelihood weights.

For the Palindrome example, we get a single symbolic derivation and the LW sampler quickly converges to the actual probability, while the independent sampler fails to converge even after a million samples. However, node- and arc-consistency can discover no further domain restrictions; forward checking at sampling time generated all the restrictions for LW sampler. The unusual pattern of variance for independent sampler in the initial iterations is due to it not being able to generate consistent samples and hence not having an estimate for the answer probability. The birthday collision example produces a number of symbolic derivations which were incorporated in an explicit OSDD. Domain restrictions are discovered only via forward checking, and that too for only one variable. The results show smaller difference between independent sampling and LW sampling for this example, compared to the others. One interesting observation from this example was that independent sampling using the OSDD structure was significantly faster (up to 2×) than using the program directly. This is because the program's non-deterministic evaluation has been replaced by a deterministic traversal through the OSDD.

Overheads. For all the examples, the time to construct the symbolic derivations, and propagate constraints was negligible (ranging from 4ms for Grid BN to 7ms for Birthday Collision, with XSB 3.5.0 on a 2.5GHz Intel Core 2 Duo machine). The overheads for sampling however were more pronounced. While an independent sampler picks values from the given distributions, the likelihood-weighting sampler needs to construct restricted domains to draw samples from. Consequently, our LW sampler takes up to 4× per sample as an independent sampler.

Comparison with PITA and ProbLog. We evaluated the exact inference procedures of PITA and ProbLog on the same examples. We used a timeout of 15 minutes for both systems. The exact inference algorithm of ProbLog using *sentential decision diagrams* was able to handle Grid BN instances of size up to 9×9. However, ProbLog's inference does not scale beyond small problem sizes for the remaining three examples. In contrast, exact inference algorithm of PITA scaled much better. PITA could successfully compute the conditional probabilities for Grid BN (up to size 10×10), Ising model (up to 13×13) and Palindrome with $n = 18$. For the Ising model example, PITA's inference completes but with numerical errors due to the low probability of evidence. Finally, PITA's inference completed for the Birthday example with population size 2, but ran out of memory for larger population sizes.

5 Related Work

Probabilistic Constraint Logic Programming [11] extends PLP with constraint logic programming (CLP). It allows the specification of models with imprecise probabilities. Whereas a world in PLP denotes a specific assignment of values to random variables, a world in PCLP can define constraints on random variables, rather than specific values. Lower and upper bounds are given on the probability of a query by summing the probabilities of worlds where query follows and worlds where query is possibly true respectively. While the way in which "proof constraints" of a PCLP query are obtained is similar to the way in which symbolic derivations are obtained (i.e., through constraint based evaluation), the inference techniques employed are completely different with PCLP employing satisfiability modulo theory (SMT) solvers.

cProbLog extends ProbLog with first-order constraints [6]. This gives the ability to express complex evidence in a succinct form. The semantics and inference are based on ProbLog. In contrast, our work makes the underlying constraints in a query explicit and uses the OSDDs to drive inference.

CLP($\mathcal{BN}$) [4] extends logic programming with constraints which encode conditional probability tables. A CLP($\mathcal{BN}$) program defines a joint distribution on the ground skolem terms. Operationally, queries are answered by constructing the relevant BN and performing BN inference.

There has been a significant interest in the area of lifted inference as exemplified by the work of [15,1,12]. The main idea of lifted inference is to treat indistinguishable *instances* random variables as one unit and perform inference at the population level. In contrast, exact inference using OSDDs treats indistinguishable *values* of random variables as one unit, thereby computing probabilities without grounding the random variables. Consequently, the method in this paper is orthogonal to traditional lifted inference and can be used when inversion and counting elimination are inapplicable (e.g. Birthday Collision example in Fig. 5).

The use of sampling methods for inference in PLPs has been widespread. The evidence has generally been handled by heuristics to reduce the number of rejected samples [5,13]. However we provide a systematic approach to deal with constraints imposed by evidence. When our constraint processing algorithm is powerful enough, the sampler can generate consistent samples without any rejections.

Adaptive sequential rejection sampling [10] is an algorithm that adapts its proposal distributions to avoid generating samples which are likely to be rejected. However, it requires a decomposition of the target distribution, which may not be available in PLPs. Further, in our work the distribution from which samples are generated is not adapted. It is an interesting direction of research to combine adaptivity with the proposed sampling algorithm.

6 Discussion

We presented a technique for inference in PLPs based on constructing a symbolic structure called OSDD using constraint propagation. The technique effectively performs inference without enumeration for a number of programs. The technique also uncovers the dependencies between random variables, which can then be exploited by more powerful inference techniques (e.g. Gibbs-sampling-based MCMC) that were inapplicable otherwise. However, for programs where symbolic derivations match one-to-one with concrete derivations, the technique offers no benefit. An important topic of future work is to statically analyze a program to determine when (and when not to) use this technique. OSSDs are constructed by exploiting the presence of explicit random variables due to `msw`'s in PRISM. Application to other (equally expressive) PLP languages remains to be explored. Finally, OSDDs introduce a style of inference where indistinguishable valuations of random variables are treated together; combining this with lifted inference that groups indistinguishable random variables together will improve the scalability of inference in PLPs.

References

1. Rodrigo De Salvo Braz, Eyal Amir, and Dan Roth. Lifted first-order probabilistic inference. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pages 1319–1325, 2005.
2. Randal E Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys (CSUR)*, 24(3):293–318, 1992.
3. Kenil CK Cheng and Roland HC Yap. Constrained decision diagrams. In *Proceedings of the National Conference on Artificial Intelligence*, volume 20, page 366. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2005.
4. Vítor Santos Costa, David Page, Maleeha Qazi, and James Cussens. CLP (BN): Constraint logic programming for probabilistic knowledge. In *Proceedings of the Nineteenth conference on Uncertainty in Artificial Intelligence*, pages 517–524. Morgan Kaufmann Publishers Inc., 2002.
5. James Cussens. Stochastic logic programs: Sampling, inference and applications. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, pages 115–122. Morgan Kaufmann Publishers Inc., 2000.
6. Daan Fierens, Guy Van den Broeck, Maurice Bruynooghe, and Luc De Raedt. Constraints for probabilistic logic programming. In *Proceedings of the NIPS Probabilistic Programming Workshop*, pages 1–4, 2012.
7. Robert M. Fung and Kuo-Chu Chang. Weighing and integrating evidence for stochastic simulation in Bayesian Networks. In *Proceedings of the Fifth Annual Conference on Uncertainty in Artificial Intelligence*, UAI '89, pages 209–220, Amsterdam, The Netherlands, 1990. North-Holland Publishing Co.
8. Stuart Geman and Donald Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (6):721–741, 1984.
9. W Keith Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.
10. Vikash K Mansinghka, Daniel M Roy, Eric Jonas, and Joshua B Tenenbaum. Exact and approximate sampling by systematic stochastic search. In *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics*, pages 400–407, 2009.
11. Steffen Michels, Arjen Hommersom, Peter JF Lucas, Marina Velikova, and Pieter Koopman. Inference for a new probabilistic constraint logic. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 2540–2546. AAAI Press, 2013.

12. Brian Milch, Luke S Zettlemoyer, Kristian Kersting, Michael Haimes, and Leslie Pack Kaelbling. Lifted probabilistic inference with counting formulas. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, pages 1062–1068, 2008.
13. Bogdan Moldovan, Ingo Thon, Jesse Davis, and Luc De Raedt. MCMC estimation of conditional probabilities in probabilistic programming languages. In *Symbolic and Quantitative Approaches to Reasoning with Uncertainty*, pages 436–448. Springer, 2013.
14. Arun Nampally and C. R. Ramakrishnan. Constraint-based inference in probabilistic logic programs. Technical report, Computer Science Department, Stony Brook University, `http://www.cs.stonybrook.edu/~cram/Papers/NR_LW15/lw15.pdf`, 2015.
15. David Poole. First-order probabilistic inference. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, volume 3, pages 985–991, 2003.
16. Taisuke Sato and Yoshitaka Kameya. PRISM: a language for symbolic-statistical modeling. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, volume 97, pages 1330–1339, 1997.
17. Taisuke Sato and Yoshitaka Kameya. Parameter learning of logic programs for symbolic-statistical modeling. *Journal of Artificial Intelligence Research*, pages 391–454, 2001.
18. Ross D. Shachter and Mark A. Peot. Simulation approaches to general probabilistic inference on belief networks. In *Proceedings of the Fifth Annual Conference on Uncertainty in Artificial Intelligence*, UAI '89, pages 221–234, Amsterdam, The Netherlands, 1990. North-Holland Publishing Co.