

Adaptive MCMC-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

submitted 1 January 2003; revised 1 January 2003; accepted 1 January 2003

Abstract

Probabilistic Logic Programming (PLP) languages enable programmers to specify systems that combine logical models with statistical knowledge. The inference problem, to determine the probability of query answers in PLP, is intractable in general. In this paper, we present a technique for approximate inference of conditional probabilities for PLP queries. It is an adaptive Markov Chain Monte Carlo (MCMC) technique, where the proposal distribution is modified as the Markov Chain is explored. In particular, the distribution is modified to increase the likelihood that a proposed sample is consistent with evidence. In our context, each sample is uniquely characterized by the outcomes of a set of random variables. Inspired by reinforcement learning, our technique propagates positive rewards to random variable/outcome pairs used in a consistent sample. The cumulative rewards of each outcome of a random variable is used to derive a new “adapted distribution” for the variable. For a query with “Markovian evaluation structure”, we show that the adapted proposal distribution converges to the query’s conditional probability distribution. We empirically evaluate the effectiveness of the adaptive sampling method.

1 Introduction

Probabilistic Logic Programming (PLP) covers a class of Statistical Relational Learning frameworks (Getoor and Taskar 2007) aimed at combining logical and statistical reasoning. Examples of languages and systems combining logical and statistical inference include ICL (Poole 1997), SLP (Muggleton et al. 1996), PRISM (Sato and Kameya 1997), LPAD (Vennekens and Verbaeten 2003) and ProbLog (De Raedt et al. 2007). In addition to standard statistical models, these languages allow reasoning over many models where logical and statistical knowledge is intricately combined, and cannot be expressed as standard statistical models.

An example problem with such a model is reachability over finite probabilistic graphs, i.e., graphs in which the presence or absence of edges is determined by a set of independent probabilistic processes. Fig. 1(a) shows a probabilistic graph, where labels on the edges denote the probability with which that edge is present. The logical relationship between reachability of e from a , and the underlying edges in the graph cannot be expressed concisely in traditional probabilistic frameworks, but is easily specified in a probabilistic logic framework. A PRISM program encoding this problem is shown in Fig. 1(b).

As illustrated in Fig. 1(b), PRISM adds *probabilistic facts* of the form $\text{msw}(s, i, t)$ where s is a term representing a random process called a *switch*, i a term representing

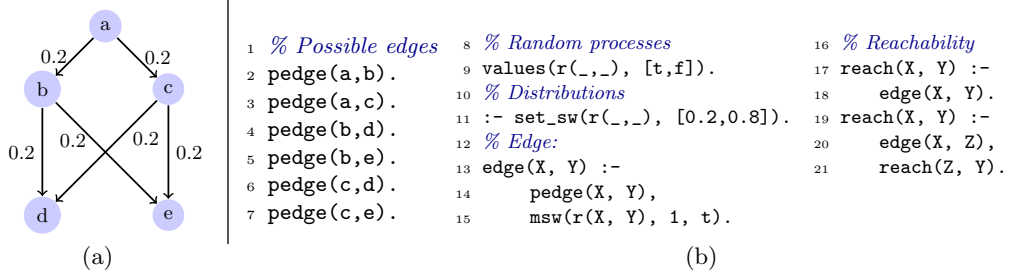


Fig. 1. Example: (a) Probabilistic Graph; (b) Reachability over probabilistic graphs in PRISM

its instance, and t its outcome. Instance i in an `msw` may be omitted when a switch has only one instance. The range of a switch is specified by “`values`” declarations; and its distribution by “`set_sw`” declarations. A *possible world* associates an outcome with each switch instance, and can be seen as a set of `msw` facts external to the program. In each possible world, the PRISM program, together with `msw` facts defining the world is a non-probabilistic program; the distribution over the possible worlds induces a distribution over the models of the PRISM program. Such a declarative *distribution semantics*, originally defined for ICL and PRISM, has been defined for other PLP languages such as LPAD and ProbLog as well.

The Problem. Inference of answer probabilities in PLPs is intractable in general. Of the several powerful sampling-based techniques developed for statistical reasoning, Markov Chain Monte Carlo (MCMC) techniques are especially suited for inference in PLPs, as shown by Cussens (2000) and Moldovan et al. (2013).

PLP queries for evaluating conditional probabilities are called as *conditional queries* and denoted as $\text{prob}(q \mid e)$, where q and e are ground atomic goals, called *query* and *evidence*, respectively. A conditional query $\text{prob}(q \mid e)$ denotes the suitably normalized distribution of q over all possible worlds where e holds. Existing PLP systems either provide efficient techniques that apply to a restricted class of q and e (e.g., *hindsight* in PRISM) or do not treat evidence specially, leading to poor performance especially when the likelihood of evidence is low. For instance, consider evaluating $\text{prob}(\text{reach}(a, d) \mid \text{reach}(a, e))$, over the probabilistic graph in Fig. 1(a). Techniques such as the one proposed by Moldovan et al. (2013) will generate a world and reject it if evidence does not hold in the world. Since the probability that e is reachable is about 0.08, a number of generated worlds will be inconsistent with the evidence, and hence unusable for computing the conditional probability.

We address the problem of efficiently estimating conditional probability by developing an *Adaptive* Markov Chain Monte Carlo (AMCMC) technique. Following adaptive MCMC techniques in statistical reasoning, we progressively modify the distribution from which samples are derived, so as to favor those samples that are consistent with evidence. The adaptive sampler reduces the number of generated samples needed to estimate the conditional probability to a given precision.

Approach Overview and Summary of Contributions. Our technical development starts with an MCMC technique where each state of the Markov chain is an *assignment* of values to a set of switch instances. An assignment at a state corresponds to a set of possible worlds such that the truth values of evidence and query are identical

in all the worlds in the set. Transitions are proposed on this chain by resampling one or more switch instances in the state and extending the resulting assignment to another state. A Metropolis-Hastings (Hastings 1970) sampler is used to accept or reject this proposal, yielding the next state in the chain. The procedure is described in Section 3. Since the states of the Markov chain are sets of possible worlds, this procedure is largely *independent of LP evaluation itself*. It can hence be used for approximate inference even in the presence of complex logical inference procedures such as tabling.

We introduce adaptation into this basic MCMC technique (see Section 4). For each switch instance/outcome pair, we maintain its *Q-value*: an estimate of the likelihood that an evaluation of the evidence goal using that switch instance/outcome will succeed. Q-values are computed by propagating rewards through the set of switch instance/outcome pairs used in the evidence goal’s evaluation, where the reward depends on the success or failure of the evaluation. The adapted distribution of a switch instance is proportional to its original distribution weighted by the Q-values of each outcome.

Although motivated by conditional queries, the technique we describe is more generally applicable, even to unconditional queries. For a class of queries with “Markovian evaluation structure”, the adapted distribution of a random variable coincides with its marginal. For such queries, we obtain an alternative adaptation procedure that can be used to obtain an adaptive independent sampler.

We describe the results of our preliminary experiments to evaluate the MCMC procedure as well as the adaptation procedure in Section 5. The rest of the paper begins with a brief overview of MCMC in Section 2. A more detailed description of related work and concluding remarks appears Section 6.

2 Preliminaries: Markov Chain Monte Carlo Techniques

A sequence of random variables $X^{(i)}, i \geq 0$ taking on values $x^{(i)}$ is called a *Markov chain* if $P(x^{(i)} | x^{(i-1)}, x^{(i-2)}, \dots, x^{(1)}) = P(x^{(i)} | x^{(i-1)})$ (Andrieu et al. 2003). The values of the random variables are chosen from a fixed set called the state space of the Markov chain. When the state space is finite, the one step transition probabilities between various states are generally given as a matrix known as the *transition kernel*.

For certain Markov chains, irrespective of the initial distribution on $X^{(0)}$, the distribution on the values $X^{(n)}$ converges as n increases, to its *limiting* or *stationary distribution*. More formally, a stationary distribution π with respect to a Markov chain with transition kernel A satisfies the condition $\pi = \pi A$.

Given a hard-to-sample *target* distribution, MCMC techniques solve the problem by constructing a Markov chain whose stationary distribution is the target distribution and drawing samples from it. **Metropolis-Hastings** (MH) is a popular MCMC-based sampling technique. Given a target distribution π and an irreducible, aperiodic Markov chain with transition kernel A , the MH sampler proposes a transition from state x to y according to $A(x, y)$, but then accepts or rejects this proposal according to the acceptance probability $\min\{1, \frac{\pi(y)A(y, x)}{\pi(x)A(x, y)}\}$ (Hastings 1970).

Adaptive MCMC. Given a target distribution, MCMC algorithms construct a Markov chain whose stationary distribution matches the target. Several Markov chains may have matching stationary distribution, with different rates of convergence. Adaptive MCMC

algorithms tune a given transition kernel, effectively switching between different Markov chains as samples are drawn (Roberts and Rosenthal 2007).

The *total variation distance* between two distributions P and Q is defined as $\|P - Q\| = \frac{1}{2} \sum_x |P(x) - Q(x)|$ (Levin et al. 2009). An adaptive MCMC algorithm preserves the stationary distribution under the *ergodicity* conditions (Roberts and Rosenthal 2007) given below. In the following, $A(x, \cdot)$ denotes the x -th row of matrix A , and $\pi(\cdot)$ denotes the *row vector* π .

Ergodicity conditions. Given a family of transition kernels $\{P_{\Gamma_1}, P_{\Gamma_2}, \dots\}$, with π as the common stationary distribution, adaptive MCMC algorithms choose kernel P_{Γ_i} at time i . The update rule of P_{Γ_i} is specified by the adaptive algorithm. Then ergodicity is preserved if all transition kernels have *simultaneous uniform ergodicity*, namely,

$$\forall \varepsilon > 0, \exists N \text{ such that } \|P_{\gamma}^N(x, \cdot) - \pi(\cdot)\| \leq \varepsilon \text{ for all } x \text{ and } \gamma$$

and the following *diminishing adaptation* condition is satisfied.

$$\lim_{n \rightarrow \infty} \sup_x \|P_{\Gamma_{n+1}}(x, \cdot) - P_{\Gamma_n}(x, \cdot)\| = 0 \text{ in probability}$$

3 MCMC for Probabilistic Logic Programs

The ability to treat a PRISM program as non-probabilistic in each world also helps us in designing sample-based query evaluation. Given a PRISM program P and a ground goal q , we lazily construct a set of worlds by sampling, such that q succeeds or fails in all worlds in the set. The set of worlds are represented by *assignments* described below.

Assignments. An *assignment* is denoted by partial function σ such that $\sigma(s, i)$ is the value of instance i of switch s . Note that σ represents a set of worlds; the set of worlds corresponding to σ is denoted by $worlds(\sigma)$.

Let σ be an assignment. Then $\sigma[(s, i) \rightarrow v]$ is an assignment that is identical to σ at every point except at (s, i) where it is v . We define a partial order “ $\succeq$ ” over assignments: $\sigma' \succeq \sigma$ if $\sigma'(s, i) = \sigma(s, i)$ whenever $\sigma(s, i) \neq \perp$. We also say that σ' *extends* σ if $\sigma' \succeq \sigma$. Two assignments σ and σ' are *mutually exclusive*, denoted by $\sigma \parallel \sigma'$, if there is some switch instance (s, i) such that both σ and σ' are defined at (s, i) , but $\sigma(s, i) \neq \sigma'(s, i)$. Two assignments are *compatible* (denoted by “ $\nparallel$ ”) if they are *not* mutually exclusive.

Given a switch s , we denote by $random(s)$ a value randomly drawn from the domain of s using the probability distribution defined for s . For looking up in an assignment or extending an assignment, we use function *pick_value* defined as follows:

$$pick_value(\sigma, s, i) = \begin{cases} \langle v, \sigma \rangle & \text{if } \sigma(s, i) = v \neq \perp \\ \langle v, \sigma[(s, i) \mapsto v] \rangle & \text{if } \sigma(s, i) = \perp \text{ and } v = random(s) \end{cases}$$

Note that *pick_value* is non-decreasing in the sense that if $\langle v, \sigma' \rangle = pick_value(\sigma, s, i)$, then $\sigma' \succeq \sigma$.

Sampling Evaluators. Our MCMC algorithm is parameterized with respect to a probabilistic query evaluation procedure called the *Sampling Evaluator*. Given an assignment σ and ground goal q , the sampling evaluator (probabilistically) generates an answer to q (*success/failure*), denoted by $ans(q)$, an assignment σ' , and a sequence ρ of switch/instance/outcome triples $(s_1, i_1, v_1), (s_2, i_2, v_2), \dots, (s_k, i_k, v_k)$, $k \geq 0$ such that the following conditions hold:

- SE1.** Consider a sequence of assignments $\sigma_0, \sigma_1, \dots, \sigma_k$ such that $\sigma_0 = \emptyset$, and $\sigma_j = \sigma_{j-1}[(s_j, i_j) \mapsto v_j]$, $0 < j \leq k$. Then, $\sigma_k = \sigma'$, and σ' is compatible with σ , i.e. $\sigma' \Vdash \sigma$.
- SE2.** If $\text{ans}(q) = \text{success}$ (similarly, *failure*), then q is *true* (or *false*, resp.) in all worlds $w \in \text{worlds}(\sigma')$.

Moreover, let Σ denote the set of all σ' generated by the sampling evaluator. Then,

- SE3.** If w is a world s.t. q is true (similarly, false) in w , then $\exists \sigma' \in \Sigma$ such that $w \in \text{worlds}(\sigma')$ and $\text{ans}(q) = \text{success}$ (or *failure*, respectively).
- SE4.** Every distinct $\sigma_a, \sigma_b \in \Sigma$ are mutually exclusive: i.e. either $\sigma_a = \sigma_b$, or $\sigma_a \parallel \sigma_b$.

Properties **SE2** and **SE3** correspond to soundness and completeness, respectively. Property **SE4** ensures that a sampling evaluator can be used to define states in an MCMC algorithm. We can use Prolog-style evaluation, performed until the first derivation is found (if one exists) to construct a sampling evaluator satisfying the above requirements including **SE4**. The key idea is to implement *pick_value* to draw samples for switch instances and commit to the chosen values by representing assignments in the dynamic database. This evaluator is described in detail in (Nampally and Ramakrishnan 2014).

Initial State. When evaluating probabilities of unconditional queries, we generate an assignment corresponding to the initial state by invoking a sampling evaluator with an empty assignment. For conditional queries, a randomly constructed explanation for evidence is used to generate the initial state. We do this via a backtracking search for a derivation of evidence, and collect all the switches and outcomes used in that derivation into an initial assignment. The initial assignment is randomly constructed by randomizing the order in which clauses and switch values are selected during the backtracking search. We refer to this procedure as *InitialSample*(P, e) in the MCMC algorithm shown in Fig. 2.

Transitions. Consider a state in the Markov Chain corresponding to assignment σ_j . We generate a successor state by (1) generating an alternative assignment σ' by assigning different outcomes for some switch instances in σ_j , and (2) invoking *SamplingEvaluator* with σ' to evaluate e and q to obtain the proposal for the next state, $\hat{\sigma}_{j+1}$. The switch instances to be resampled can be selected in many ways. We use one of the following:

1. Single Switch: We select a single (s, i) such that $\sigma_j(s, i) \neq \perp$ uniformly, and generate $\sigma' = \sigma_j[(s, i) \mapsto \perp]$, effectively forgetting (s, i) .

2. Multi-Switch: This resampling mode is parameterized with a probability P . We generate σ' from σ_j by forgetting with probability P each (s, i) for which σ_j is defined. In Fig. 2, the resampling procedure is referred to as *Resample*.

```

1: function MCMC
2:   Input:  $P$ : Program,  $q$ : Query,  $e$ : Evidence,
3:    $N$ : Steps to simulate
4:   Output:  $p = \text{prob}(q \mid e)$ 
5:   // Initialize
6:    $\sigma_0 := \text{InitialSample}(P, e)$ 
7:    $(r_q, \sigma, \rho) := \text{SamplingEvaluator}(P, q, \sigma_0)$ 
8:    $N_q := 0$ 
9:   // Generate a chain of length  $N$ 
10:  for  $N$  times do
11:     $\sigma' = \text{Resample}(\sigma)$ 
12:     $(r'_e, \sigma_e, \rho) = \text{SamplingEvaluator}(P, e, \sigma')$ 
13:    if  $r'_e = \text{success}$  then
14:       $(r'_q, \sigma_q, \rho) = \text{SamplingEvaluator}(P, q, \sigma_e)$ 
15:      if  $\text{accept}(\sigma, \sigma_q)$  then
16:         $\sigma := \sigma_q$ 
17:         $r_q := r'_q$ 
18:      if  $r_q = \text{success}$  then  $N_q := N_q + 1$ 
19:  return  $N_q/N$ 

```

Fig. 2. MCMC Algorithm for Inferring Conditional Probabilities

Metropolis-Hastings. To draw samples from the target distribution $\text{prob}(q \mid e)$, we construct an MH sampler as follows. If the proposed state is inconsistent with evidence, it is rejected deterministically. If it is consistent, it is accepted/rejected based on *acceptance probability*. For single switch resampling strategy, the acceptance probability to go to state σ_2 from σ_1 is $\min\{1, \frac{|\sigma_1|}{|\sigma_2|}\}$. For multi-switch resampling strategy, the acceptance probability is 1, as derived in (Nampally and Ramakrishnan 2014).

4 Adaptive MCMC for Probabilistic Logic Programs

The rate at which samples are rejected deterministically based on the evidence (due to failure of condition in line 11 of Fig. 2) is called the *rejection rate*. We now present a technique to progressively adapt the proposal distribution based on the samples generated so far, in order to reduce the rejection rate.

The adaptation algorithm we present here is inspired by Q-learning, a reinforcement learning technique (Sutton and Barto 1998). For each switch s , instance i and outcome v used by the sampling evaluator, the Q-value $Q(s, i, v)$ is a real number in $[0, 1]$. Intuitively $Q(s, i, v)$ represents the probability of generating a consistent sample, when the sampling evaluator chooses v as the outcome of the i -th instance of switch s .

Initially, all Q-values are set to the same constant, essentially assuming that are equally likely to yield consistent samples. At each iteration of MCMC, adaptation is done after evidence is evaluated, by passing rewards to each switch/instance/outcome triple in ρ (computed in line 10 of Fig 2). We begin this processing with *reward* = 0 if $r'_e = \text{failure}$, denoting an inconsistent sample, and *reward* = 1 otherwise. We work backwards through the sequence ρ so that the last switch/instance/outcome is given a reward of 0/1, which it then modifies and passes to the switch/instance/outcome preceding it in ρ . The Q-value of each random process/instance/outcome is computed as the average of the all rewards received by it. The algorithm for maintaining Q-values is given in Fig. 3.

The MCMC algorithm in Fig. 2 is modified for adaptive sampling as follows. First of all, function ADAPT is invoked after line 16. Secondly, *pick_value* function used in the sampling evaluator draws values for a switch instance (s, i) based on the normalized product of the original distribution and the Q-values of (s, i) . Finally, the acceptance probability computation is modified to take the adapted distributions into account. Consider computing the acceptance probability to transition from state σ to σ' . We can partition the assignment σ into three non-overlapping functions: σ_1 for those (s, i) 's defined by σ but not by σ' ; σ_2 for those defined by both σ and σ' but assigned different values; and finally, σ_3 for those defined by both σ and σ' and assigned same values. We can similarly partition σ' into σ'_1 , σ'_2 and σ'_3 .

```

1: function ADAPT
2:   Input:  $\rho, r$ : Reward
3:   Global:  $Q$ : Q-values,  $c$ : counts,
4:            $t$ : total Q-values.
5:    $j := \text{length}(\rho)$  ▷ Initialize
6:   while  $j > 0$  do
7:     let  $(s_j, i_j, v_j) = \rho[j]$ 
8:      $t(s_j, i_j, v_j) := (t(s_j, i_j, v_j) + r)$ 
9:      $c(s_j, i_j, v_j) := c(s_j, i_j, v_j) + 1$ 
10:     $Q(s_j, i_j, v_j) := t(s_j, i_j, v_j) \div c(s_j, i_j, v_j)$ 
11:     $j := j - 1$ 
12:     $r := \sum_{v \in \text{values}(s_j)} P(s_j, i_j, v) * Q(s_j, i_j, v)$ 

```

Fig. 3. Adaptation of Q-values

For single-switch resampling strategy, the acceptance probability is given by

$$\min \left(1, \frac{P(\sigma'_1)P(\sigma'_2)P'(\sigma_1)P'(\sigma_2)1/|\sigma'|}{P(\sigma_1)P(\sigma_2)P'(\sigma'_1)P'(\sigma'_2)1/|\sigma|} \right)$$

where P is the original probability and P' is the adapted probability. For multi-switch strategy, the acceptance probability is given by

$$\min \left(1, \frac{P(\sigma'_1)P(\sigma'_2)P'(\sigma_1)P'(\sigma_2)}{P(\sigma_1)P(\sigma_2)P'(\sigma'_1)P'(\sigma'_2)} \right)$$

The derivations of these probabilities and the proof that the algorithm preserves ergodicity with respect to the target distribution are in (Nampally and Ramakrishnan 2014).

Beyond MCMC. It should be noted that the adapted distribution may not coincide with the conditional distribution $\text{prob}(q \mid e)$. This is not a problem for MCMC, since the adapted distribution is used as the proposal. However, the same adaptation scheme cannot be used, in general, with other sampling strategies such as independent sampling. Nevertheless, for a class of program/query pairs whose sampling evaluations is “Markovian”, each switch instance’s adapted distribution converges to its marginal distribution. For such programs, a modified adaptation can be used for independent sampling as well.

Consider the class of programs and queries for which the sequence ρ of random process/instance outcomes $(s_1, i_1, v_i), \dots, (s_k, i_k, v_k)$ is such that the probability $P(\text{ans}(e) \mid (s_j, i_j, v_j))$ is independent of the triples $(s_l, i_l, v_l), l < j$. These program/query pairs are said to have a *Markovian Evaluation Structure*. We can redefine the Q-value to be the last propagated reward, ensuring that the rewards received by any switch instances will monotonically decrease due to adaptation. This allows us to adapt independent sampling as well as MCMC for such programs and queries.

5 Experimental Results

The MCMC algorithm was implemented in the XSB logic programming system (Swift et al. 2012). The sampling evaluator and the main control loop (Fig. 2) were implemented in Prolog. Lower level primitives managing the maintenance of assignments, resampling, computation of acceptance/rejection were implemented in C and invoked from Prolog. We evaluated the performance of this implementation on four synthetic examples: BN, Hamming, Grammar, and Reach. The experiments were run on a machine with 2.4GHz Opteron 280 processor and 4G RAM.

BN. This example consists of Bayesian networks whose Boolean-valued variables are arranged in the form of a 6×6 grid with each node having its left and top neighbors (if any) as parents. Evidence sets the outcome of 6 variables; and we query the outcome of one of the remaining variables. Fig 4(a) shows the conditional probability estimated by our algorithm plotted as a function of sample size. The time overhead for performing adaptation for this problem was small. This example clearly illustrates the benefit of adaptation.

Hamming. The Hamming code example is a PRISM program that generates a set of (4,3) Hamming codes. The evidence is a set of bits in the code with fixed values, and the query is the value of a non-evidence bit (Moldovan et al. 2013). The data bits in the code

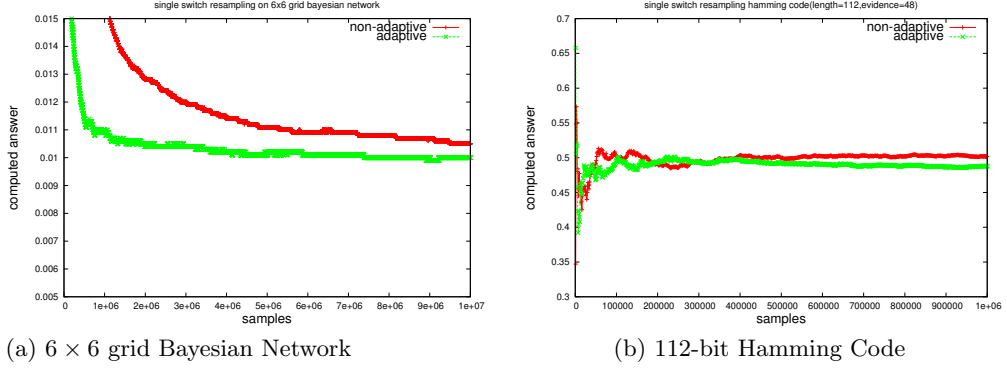


Fig. 4. Computed probability vs. sample size for two examples

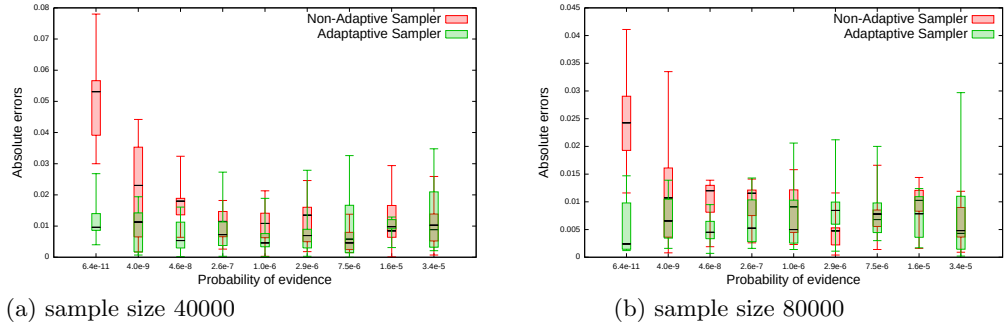


Fig. 5. Effectiveness of adaptation with varying probability of evidence and sample sizes

were independent random variables, while the parity bits were computed from the data bits' values. The answers computed by adaptive and non-adaptive samplers are given in Fig 4(b). In this example, the convergence of the adaptive MCMC is only a little better than that of the non-adaptive algorithm.

Reach. The final set of examples are reachability queries in probabilistic acyclic graphs, of the form shown in Fig. 1. For the graph shown in Introduction, while computing $\text{prob}(\text{reach}(\mathbf{a}, \mathbf{d}) \mid \text{reach}(\mathbf{a}, \mathbf{e}))$, the non-adaptive sampler rejects 8% of the samples, while the adaptive one rejects 1.5%. Similar rejection rates were observed for larger randomly generated graphs as well. However, since the rejection rate of the non-adaptive sampler is low, there is no significant difference between the convergence of adaptive and non-adaptive samplers.

Effect of probability of evidence. We compared the answers computed by the adaptive and non-adaptive samplers as the probability of evidence is varied. We used the 6×6 grid BN described earlier and varied the probability of evidence from 6.4×10^{-11} to 3.4×10^{-5} and used sample sizes of 40k and 80k. The absolute values of errors made by both the samplers are plotted in Fig 5. Observe from the figure that adaptive sampling exhibits significantly lower errors than non-adaptive sampling when the probability of evidence is low. This difference disappears when the evidence probability is high.

6 Discussion

An MCMC-based algorithm for approximate inference of stochastic logic programs (SLP)s (Muggleton et al. 1996) was presented by Cussens (2000). The technique uses derivations as states of the chain, and proposes transitions by selecting alternative branches using repeated Bernoulli trials. In contrast, our sampling technique is largely independent of the query evaluation process itself. More recently, Moldovan et al. (2013) describe an MCMC technique for inference in ProLog that treats explanations as states. However, special handling is needed since explanations may not be mutually exclusive in general, resulting in memory and time overheads when traversing the chain. In contrast, we use Prolog-style evaluation to assure that the samples are pair-wise mutually exclusive. A more detailed comparison of these techniques appears in (Nampally and Ramakrishnan 2014).

Adaptation techniques have been used to focus other sampling algorithms as well. For instance, Mansinghka et al. (2009) presents an adaptive sequential rejection sampling algorithm. This algorithm requires knowledge of the factors in the distribution from which samples are drawn. Since PRISM programs represent logical as well as statistical knowledge, explicit knowledge may not even be available in our case. Consequently, our work does not rely on an explicit knowledge of factors.

Recent works have used sampling techniques for parameter estimation in PLP. Sato (2011) presents an MCMC-based algorithm for computing posterior distribution on the parameters of a PRISM program. At its core, that work builds a Markov chain over the space of parameters, assuming that answer probabilities can be computed efficiently. In contrast, our work deals with inference when answer probabilities are intractable. Cussens (2011) describes an application of Approximate Bayesian Computation (ABC) Sequential Monte Carlo (SMC) (Toni et al. 2009) technique to the problem of parameter estimation in PRISM programs. A scheme similar to SMC is used to maintain a weighted set of parameters (particles), and these particles are updated using a “perturbation kernel”. The ABC scheme generates synthetic data using independent sampling and uses the notion of distance between synthetic data and observed data to weight the particles. An interesting open question is whether MCMC-based computation of answer probabilities can be effectively used by the ABC scheme.

This paper focused on a generic MCMC method and adaptation, and did not consider the effect of resampling strategies. The order in which random processes are sampled may affect the convergence and hence the quality of inference. For instance, Decayed MCMC (Marthi et al. 2002) samples processes based on a temporal order. As future work, we plan extend our sampler to use an order based on programmer annotation; whether such annotations can be inferred from the program is an open problem. Finally, while sampling-based inference may be generally deployed, exact inference may still be feasible for queries with short derivations. Hence, an interesting direction of future work is to develop a hybrid inference technique that can combine exact and approximate inference based on programmer annotation. Such an inference technique can be seen as an analogue of the Rao-Blackwellized Particle Filtering method developed for Dynamic Bayesian Networks (Doucet et al. 2000).

Acknowledgements: This work was supported in part by NSF Grants CCF-1018459, CCF-0831298, and ONR Grant N00014-07-1-0928.

References

- ANDRIEU, C., DE FREITAS, N., DOUCET, A., AND JORDAN, M. 2003. An introduction to MCMC for machine learning. *Machine learning* 50, 1, 5–43.
- CUSSENS, J. 2000. Stochastic logic programs: Sampling, inference and applications. In *Uncertainty in Artificial Intelligence (UAI)*. Morgan Kaufmann, 115–122.
- CUSSENS, J. 2011. Approximate Bayesian computation for the parameters of PRISM programs. In *Inductive Logic Programming*. Springer, 38–46.
- DE RAEDT, L., KIMMIG, A., AND TOIVONEN, H. 2007. Problog: A probabilistic Prolog and its application in link discovery. In *IJCAI*. 2462–2467.
- DOUCET, A., FREITAS, N. D., MURPHY, K. P., AND RUSSELL, S. J. 2000. Rao-Blackwellised particle filtering for dynamic Bayesian networks. In *Uncertainty in Artificial Intelligence (UAI)*. 176–183.
- GETOOR, L. AND TASKAR, B. 2007. *Introduction to Statistical Relational Learning*. MIT press.
- HASTINGS, W. 1970. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika* 57, 1, 97–109.
- LEVIN, D. A., PERES, Y., AND WILMER, E. L. 2009. *Markov chains and mixing times*. Amer. Mathematical Society.
- MANSINGHA, V. K., ROY, D. M., JONAS, E., AND TENENBAUM, J. B. 2009. Exact and approximate sampling by systematic stochastic search. In *International Conference on Artificial Intelligence and Statistics*. 400–407.
- MARTHI, B., PASULA, H., RUSSELL, S., AND PERES, Y. 2002. Decayed MCMC filtering. In *Uncertainty in Artificial Intelligence (UAI)*. 319–326.
- MOLDOVAN, B., THON, I., DAVIS, J., AND RAEDT, L. D. 2013. MCMC estimation of conditional probabilities in probabilistic programming languages. In *European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU)*. 436–448.
- MUGGLETON, S. ET AL. 1996. Stochastic logic programs. *Advances in inductive logic programming* 32, 254–264.
- NAMPALLY, A. AND RAMAKRISHNAN, C. R. 2014. Adaptive MCMC-based inference in probabilistic logic programs. *CoRR abs/1403.6036*.
- POOLE, D. 1997. The independent choice logic for modelling multiple agents under uncertainty. *Artificial Intelligence* 94, 1, 7–56.
- ROBERTS, G. O. AND ROSENTHAL, J. S. 2007. Coupling and ergodicity of adaptive Markov chain Monte Carlo algorithms. *Journal of applied probability* 44, 2, 458–475.
- SATO, T. 2011. A general MCMC method for Bayesian inference in logic-based probabilistic modeling. In *IJCAI*. AAAI Press, 1472–1477.
- SATO, T. AND KAMEYA, Y. 1997. PRISM: a language for symbolic-statistical modeling. In *IJCAI*. 1330–1335.
- SUTTON, R. S. AND BARTO, A. G. 1998. *Introduction to reinforcement learning*. MIT Press.
- SWIFT, T., WARREN, D. S., ET AL. 2012. The XSB logic programming system, Version 3.3. Tech. rep., Computer Science, SUNY, Stony Brook. <http://xsb.sourceforge.net>.
- TONI, T., WELCH, D., STRELKOWA, N., IPSEN, A., AND STUMPF, M. P. 2009. Approximate Bayesian computation scheme for parameter inference and model selection in dynamical systems. *Journal of the Royal Society Interface* 6, 31, 187–202.
- VENNEKENS, J. AND VERBAETEN, S. 2003. A general view on probabilistic logic programming. In *Proceedings of BNAIC-03*.