Gibbs sampling in open-universe stochastic languages

Nimar S Arora
Rodrigo de Salvo Braz
Erik Sudderth
Stuart J. Russell

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2010-34
http://www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-34.html

March 27, 2010

Acknowledgement

This work wouldn't have beeen possible without the considerable assistance provided by Brian Milch to make the models presented here work in BLOG-MH. Matthew Can provided a translation of the Alarm Bayes Net to BLOG. Finally, the first author wishes to thank his family for their boundless patience and support during this work.

Gibbs sampling in open-universe stochastic languages

Nimar S. Arora, Rodrigo de Salvo Braz, Erik Sudderth, Stuart Russell

March 27, 2010

Abstract

Languages for open-universe probability models (OUPMs) can represent situations with an unknown number of objects and identity uncertainty, which comprise a very important class of real-world applications. Current general-purpose inference methods for such languages are, however, much less efficient than those implemented for more restricted languages or for specific model classes. This paper goes some way to remedying the deficit by introducing, and proving correct, a general method for Gibbs sampling in partial worlds where model structure may vary across worlds. The method draws on and extends previous results on generic OUPM inference and on auxiliary-variable Gibbs sampling for non-parametric mixture models. It has been implemented for BLOG, a well-known OUPM language. Combined with compile-time optimizations, it yields very substantial speedups over existing methods on several test cases and substantially improves the practicality of OUPM languages generally.

1 Introduction

General-purpose probabilistic modelling languages aim to facilitate the development of complex models while providing effective, general inference methods so that the model-builder need not write model-specific inference code for each application from scratch. For example, BUGS (Spiegelhalter *et al.*, 1996) can represent directed graphical models over indexed sets of random variables and uses MCMC inference (in particular, Gibbs sampling where this is possible).

As the expressive power of modelling languages increases, the range of representable problems also increases. Thus, the class of first-order, open-universe probabilistic languages, which includes BLOG (Milch *et al.*, 2005b) and Church (Goodman *et al.*, 2008), handles cases in which the number of objects (in BUGS, the index set) is unknown and perhaps unbounded, and object identity is uncertain. It is still possible to write a complete inference algorithm for BLOG, based on MCMC over *partial* worlds; each such world is constructed from the minimal self-supporting set of variables relevant to the evidence and query variables.

Generality has a price, however: BLOG's Metropolis–Hastings inference engine samples each variable conditioned only on its parents, which is unacceptably slow for many commonly used models.

Our goal in this work is to remedy this situation, primarily by extending the range of situations in which full Gibbs samplng can be used within BLOG. Section 2 of this paper introduces the terminology of Contingent Bayesian Networks (CBNs), which we will use as the propositional "abstract machine" for open-universe stochastic languages. Section 3 describes previous work related to general-purpose sampling of CBNs and describes its limitations. Section 4 describes our novel Gibbs sampling algorithm for CBNs which addresses these limitations; its implementation for BLOG is described in Section 5. Finally, we present experimental results on various models in Section 6, demonstrating substantial speedups over existing methods.

2 Contingent Bayesian Networks

This section covers definitions which are repeated from (Milch *et al.*, 2005b) and (Milch *et al.*, 2005a) for clarity. Some of the definitions have been modified to increase generality.

A Contingent Bayesian Network (CBN) consists of a set of random variables $\mathcal{V}$ and for each variable $X \in \mathcal{V}$ a domain $dom(X)$ and a decision tree $\mathcal{T}_X$. The decision tree is a directed binary tree where each node is a predicate on a subset of $\mathcal{V}$. A leaf of $\mathcal{T}_X$ is a probability distribution parameterized by a subset of $\mathcal{V}$ and defined on $dom(X)$.

Example 1. *An aircraft of unknown WingType – Helicopter or FixedWingPlane – is detected on a radar. Helicopters have an unknown RotorLength and depending on this length they might produce a characteristic pattern called a* BladeFlash *(Tait, 2009) in the returned radar signal. A FixedWingPlane might also produce a BladeFlash. (see Figure 1)*

$\mathcal{T}_{WingType} = F_1$

$$\mathcal{T}_{RotorLength} = \begin{cases} F_2 & \textit{if WingType = Helicopter} \\ \textit{null} & \textit{otherwise} \end{cases}$$

$\mathcal{T}_{BladeFlash} =$

$$\begin{cases} F_3(\textit{RotorLength}) & \textit{if WingType = Helicopter} \\ F_4 & \textit{otherwise} \end{cases}$$

An instantiation σ is an assignment of values to a subset of $\mathcal{V}$. We write $vars(\sigma)$ for the set of variables to which σ assigns values, and σ_X for the value that σ assigns to a variable X. $\sigma^{X=a}$ is a modified instantiation which agrees with σ except for setting X to a. An instantiation σ is said to be finite if $vars(\sigma)$ is finite. An instantiation σ *supports* X if all the variables needed to evaluate $\mathcal{T}_X$ are present in σ. In Example 1, [WingType=FixedWing] supports BladeFlash, but [WingType=Helicopter] doesn't.

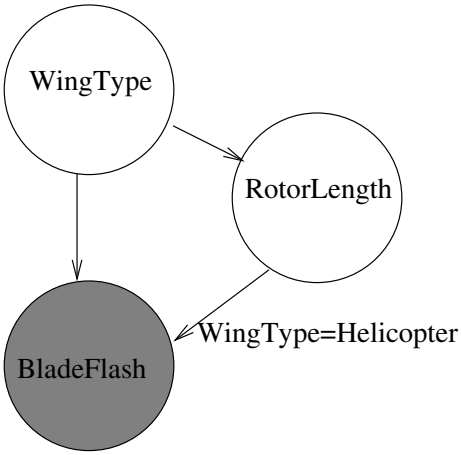

Figure 1: The CBN of Example 1

We write $\sigma_{\mathcal{T}_X}$ for the minimal subset of σ needed to evaluate $\mathcal{T}_X$. The resulting distribution of X is referred to as $p_X(. \mid \sigma_{\mathcal{T}_X})$. The parents of X in σ are $vars(\sigma_{\mathcal{T}_X})$ and the children of X are $\Lambda(\sigma, X) = \{Y | Y \in vars(\sigma), X \in vars(\sigma_{\mathcal{T}_Y})\}$ The subset of $vars(\sigma_{\mathcal{T}_X})$ which were used to evaluate the nodes of $\mathcal{T}_X$ (rather than the leaf) are the *switching parents* of X in σ. The intuition behind switching parents is that a change in the value of these can switch the distribution of X and its set of parents. A *switching variable* in σ is a variable which is a switching parent for some variable in σ. From Example 1 again, the instantiation [WingType=Helicopter, RotorLength=6, BladeFlash=true] has WingType as a switching parent of both RotorLength and BladeFlash.

An instantiation σ is *self-supporting* if it supports all variables in σ. Assuming that the CBN is well-defined (Milch *et al.*, 2005a) we can define the probability of a self-supporting instantiation:

$$p(\sigma) = \prod_{X \in vars(\sigma)} p_X(\sigma_X | \sigma_{\mathcal{T}_X}) \tag{1}$$

An instantiation σ is *feasible* if $p(\sigma) > 0$.

3 Related Work

It has been shown previously in (Milch & Russell, 2006) that the MCMC state space for inference in CBNs can consist of minimal partial instantiations that include the evidence, E, and the query variables, Q. This idea has been exploited to build the inference engine for BLOG. Also, one of the inference methods in non-parametric mixture models uses a somewhat similar idea by instantiating all the mixture components that are needed to support the evidence plus a

few more (Neal, 2000). In this section, we will briefly describe these existing methods as the basis of our new algorithm.

3.1 Parent-Conditional Sampling

In the absence of a model-specific proposal distribution, BLOG relies on a *parent-conditional proposal*, q. This algorithm picks a variable, X, at random from all the non-evidence variables in the current instantiation σ, $V(\sigma) = vars(\sigma) - E$, and proposes a new instantiation σ' with the value of X drawn from $p_X(.|\sigma_{\mathcal{T}_X})$. If X was a switching variable in σ then we may need to instantiate new variables and uninstantiate unneeded ones to make σ' minimal and self-supporting over Q and E. All new variables are instantiated with values drawn from their parent-conditional distribution. We say that any σ' constructed by this procedure is *reachable* from σ via X, or $\sigma \overset{X}{\rightsquigarrow} \sigma'$. The following are easily seen to be true of reachability.

Proposition 1. *A minimal self-supporting feasible instantiation σ' is reachable from an instantiation σ via X if and only if $X \in vars(\sigma) \cap vars(\sigma')$ and σ and σ' agree on all other variables in $vars(\sigma) \cap vars(\sigma')$.*

Proposition 2. *If $\sigma \overset{X}{\rightsquigarrow} \sigma'$ then $\nexists Y$ s.t. $\sigma \overset{Y}{\rightsquigarrow} \sigma'$ and $Y \neq X$.*

The nature of the proposal distribution makes it quite simple to compute the acceptance ratio for the Metropolis–Hastings (MH) method. In this method, the acceptance ratio for a transition is given by:

$$\alpha(\sigma \rightarrow \sigma') = min\left(1, \frac{p(\sigma')q(\sigma' \rightarrow \sigma)}{p(\sigma)q(\sigma \rightarrow \sigma')}\right) \tag{2}$$

Now, for σ' which is reachable from σ via X the only way of proposing the transition is to select X from $V(\sigma)$ and propose the value σ'_X for it and then to propose the values for all the new variables in σ'. Thus, $q(\sigma \rightarrow \sigma') =$

$$\frac{p_X({\sigma'}_X|\sigma_{\mathcal{T}_X})}{|V(\sigma)|} \prod_{Y \in vars(\sigma') - vars(\sigma)} p_Y({\sigma'}_Y|{\sigma'}_{\mathcal{T}_Y}) \tag{3}$$

From Equation (1) and Equation (3), it follows that the product terms corresponding to $vars(\sigma') - vars(\sigma)$ cancel out in $\frac{p(\sigma')}{q(\sigma \rightarrow \sigma')}$. Similarly, terms in $vars(\sigma) - vars(\sigma')$ cancel out in $\frac{q(\sigma' \rightarrow \sigma)}{p(\sigma)}$. Further, it is easy to see that for variables Y s.t. $Y \in vars(\sigma) \cap vars(\sigma') - \Lambda(\sigma, X) \cap \Lambda(\sigma', X)$, $\sigma_{\mathcal{T}_Y} = {\sigma'}_{\mathcal{T}_Y}$. Hence, $p_Y(. \mid \sigma_{\mathcal{T}_Y}) = p_Y(. \mid {\sigma'}_{\mathcal{T}_Y})$ and the terms for all such variables Y (including X) cancel out. Finally, the acceptance ratio reduces to:

$$min\left(1, \frac{|V(\sigma)|}{|V(\sigma')|} \prod_{Y \in \Lambda(\sigma,X) \cap \Lambda(\sigma',X)} \frac{p_Y({\sigma'}_Y \mid {\sigma'}_{\mathcal{T}_Y})}{p_Y(\sigma_Y \mid \sigma_{\mathcal{T}_Y})}\right) \tag{4}$$

The complete algorithm is given in Figure 2.

1. Create an initial minimal self-supporting feasible instantiation σ consistent with the evidence and including the query variables.
2. Initialize statistics of the query.
3. Repeat the following for the desired number of samples.
   (a) Pick a variable $X \in V(\sigma)$ uniformly at random.
   (b) Pick a random σ' s.t. $\sigma \overset{X}{\rightsquigarrow} \sigma'$.
   (c) Compute the acceptance ratio, α (Equation (4)).
   (d) With probability α, set $\sigma \leftarrow \sigma'$. Otherwise, let σ be unchanged.
   (e) Update query statistics using σ.
4. Report query statistics.

Figure 2: General-purpose inference in CBNs using parent-conditional sampling

3.2 Gibbs Sampling

Equation (4) helps explain the main problem with parent-conditional sampling. If the proposed value for the sampled variable X doesn't assign high probability to the children of X then the move will be rejected. Gibbs sampling (Geman & Geman, 1984) is a way to solve this issue for Bayes Nets by directly sampling X from its *full conditional* distribution, $p_X(. \mid \sigma)$ (as opposed to its parent-conditional prior $p_X(. \mid \sigma_{\mathcal{T}_X})$). This method involves computing the Gibbs weight for each value $a \in dom(X)$:

$$w(a) = p_X(a \mid \sigma_{\mathcal{T}_X}) \prod_{Y \in \Lambda(\sigma, X)} p_Y(\sigma_Y \mid \sigma_{\mathcal{T}_Y}^{X=a}) \tag{5}$$

A new value is sampled for X proportional to its weight. The MH move corresponding to this proposal is guaranteed to have an acceptance ratio of 1. This method works for variables with finite, countable, or even uncountable domains as long as the weights can be easily normalized by integration. In languages like BUGS, Gibbs sampling has proven quite successful. However, one problem with applying this technique to CBNs is that if X is a switching variable then $\sigma^{X=a}$ for some $a \in dom(X)$ may no longer support a child Y of X, and the Gibbs weights can't be computed. One solution that has been proposed in the context of Dirichlet Process Mixture Models (DPMM) by (Neal, 2000) is to enhance σ with auxiliary variables so that $\sigma^{X=a}$ is self-supporting for all $a \in dom(X)$. This enhanced σ, which is now no longer minimal, is used to construct the Gibbs weights, and then after making the move any extra variables are discarded. An important point about the auxiliary variables is that they are always sampled in σ with the current value of X. In other words, if $\sigma_X = a$ and if σ was enhanced with a variable Z which was needed to support $\sigma^{X=b}$ for some $b \in dom(X) - a$, then we would sample Z from $p_Z(. \mid \sigma_{\mathcal{T}_Z}^{X=a})$. This can lead to poor mixing or

$dom(X) = \{0, 1, 2\}$
$X \sim Categorical(.1, .6, .3)$

$\forall i \in \mathcal{N}\ dom(Y_i) = 0, 1$
$Y_i \sim \begin{cases} Bernoulli(\frac{1}{1+X}) & if\ X + i\ mod\ 2\ \equiv\ 0 \\ Bernoulli(\frac{1}{1+X+Y_{i+1}}) & otherwise \end{cases}$

Evidence: $Y_1 = true$. Query: X.

Figure 3: A CBN which requires infinitely many auxiliary variables for standard Gibbs sampling approaches.

no mixing at all if $p_Z(. \mid \sigma_{\mathcal{T}_Z}^{X=a})$ and $p_Z(. \mid \sigma_{\mathcal{T}_Z}^{X=b})$ have non-intersecting support. (In the DPMM case, this wasn't a concern since $\mathcal{T}_Z$ never referred to X and the p_Z in both instantiations was identical.)

For example, consider the model of Example 1 and a minimal instantiation, σ = [WingType = FixedWingPlane, BladeFlash = True]. If we were to apply the auxiliary variable method to do MCMC sampling in this model, we would need to instantiate RotorLength given WingType = FixedWingPlane and then construct Gibbs weights for WingType = FixedWingPlane and Helicopter. However, the only value of RotorLength that can be sampled given WingType = FixedWingPlane is null and this value has 0 probability with WingType = Helicopter. Thus, the chain will not mix at all.

In fact, there are cases when the auxiliary variable method is not even defined because we may need an unbounded number of auxiliary variables. Consider, for example, the rather artificial but instructive CBN in Figure 3, and an instantiation $\sigma = [X = 0, Y_1 = 1, Y_2 = 1]$. If we were to enhance σ to make it self-supporting for all values of X we would certainly need to add Y_3 since Y_2 depends on Y_3 when $X = 1$. But Y_3 depends on Y_4 when $X = 0$ and so we need to add Y_4, and so on. Ultimately, we would need to instantiate all Y_i's for $i \geq 1$.

4 Gibbs Sampling in Contingent Bayesian Networks

We will present here a general-purpose Gibbs sampling algorithm for switching variables with finite domains. The proposal for a switching variable, X, will proceeed in three steps. First, the instantiation, σ, will be reduced to a subset of variables, $core(\sigma, X)$, such that this subset is guaranteed to exist in any minimal self-supporting instantiation constructed from $\sigma^{X=a}$ for all $a \in dom(X)$. Second, we will construct minimal self-supporting instantiations, σ_i $i = 1 \ldots$, for each value in $dom(X) - \{\sigma_X\}$ such that these instantiation agree with σ on $core(\sigma, X)$ and they all differ on X. The rest of the variables in these σ_is are sampled from their parent-conditional priors. For notational simplicity, we will write σ_0 for σ. Finally, we will assign weights to all σ_i $i = 0 \ldots$ and make a

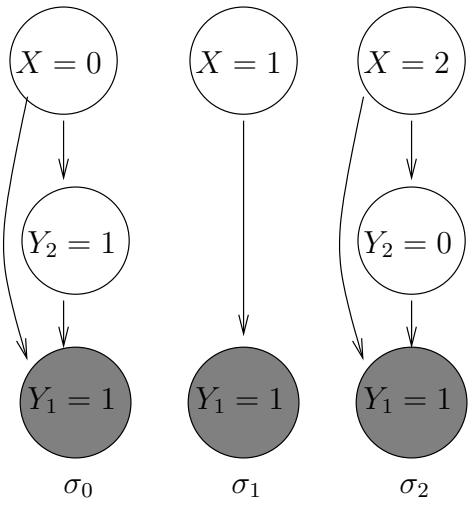

Figure 4: The three partial instantiations considered for Gibbs sampling of X given Y_1 as evidence. (σ_0 is the current instantiation)

transition proportional to these weights. We will now explain the motivation for this algorithm before providing formal details and proving it correct.

It may seem counter-intuitive to first reduce the instatiation and then extend it. After all, the previous two algorithms we described in Section 3, parent-conditional sampling and Gibbs with auxiliary variables, first extended the current instantiation and then reduced it. The motivation for our approach is that the variables whose existence depends on the value of X should be sampled in the world with the appropriate value of X. Consider again, for example, the model in Figure 3 and three partial instantiations as shown in Figure 4 for each of the values of X. Now, starting from σ_0 (which has $X = 0$) we could have decided to keep the value of Y_2 when constructing σ_2 (which has $X = 2$). However, the distribution of Y_2 given $X = 2$ is quite different from that when $X = 0$ and reusing the value could lead to construction of low probability instantiations. The other reason for resampling non-core variables like Y_2 is to simplify the detailed balance equations that we will discuss later. In brief, due to the nature of our algorithm the distribution of σ_2 doesn't depend on whether we start from σ_0 or σ_1. Hence when demonstrating detailed balance between pairs of instantiations we needn't reason about other instantiations which might be involved in the transition. This last observation relies on the fact that $core(\sigma_0, X) = core(\sigma_1, X)$. We will first prove this in general.

Definition 1. *For an instantiation σ and variables $X, Y, Z \in vars(\sigma)$, if $\mathcal{T}_Z$ refers to X and Y and the first reference of X is before the first reference to Y we call the Y to Z edge as* contingent *on X.*

Definition 2. *$core(\sigma, X)$ is defined as the subset of variables in $vars(\sigma)$ which*

have a path (possibly of length zero) consisting of parent-child edges not including edges contingent on X to some variable in $Q \cup E$. X is excluded from $core(\sigma, X)$.

Note that the ancestors of X in σ are always in $core(\sigma, X)$ since the edges from X to its children are not contingent on X.

Definition 3. *For an instantiation σ and variable $X \in vars(\sigma)$, $\Upsilon(\sigma, X) \stackrel{\triangle}{=} \Lambda(\sigma, X) \cap core(\sigma, X)$.*

Proposition 3. *For any two minimal self-supporting instantiations, σ and σ', and variable X common to them, if σ and σ' agree on $core(\sigma, X)$ then $core(\sigma, X) = core(\sigma', X)$ and $\Upsilon(\sigma, X) = \Upsilon(\sigma', X)$.*

Proof. Let $Y \in core(\sigma, X)$ then either $Y \in Q \cup E$ or there exists a path of edges not contingent on X from Y to $Q \cup E$. Clearly, if $Y \in Q \cup E$ then $Y \in core(\sigma', X)$. Otherwise, let Z be the first child in such a path. Since X is not referred before Y in $\mathcal{T}_Z$ thus X is not referred before any W which is referred before Y in $\mathcal{T}_Z$. Such a variable W must also be in $core(\sigma, X)$ since W has the same path to $Q \cup E$ via Z as Y. But σ and σ' agree on $core(\sigma, X)$ and hence on W. Since σ and σ' agree on all the variables referred before Y in $\mathcal{T}_Z$ it follows that the evaulation of $\mathcal{T}_Z$ upto Y is identical in σ and σ'. Hence, the Y to Z edge is not contingent on X in σ'. By induction, the path from Y to $Q \cup E$ in σ' is not contingent on X which implies that $Y \in core(\sigma', X)$.

Now, say $core(\sigma, X) \subset core(\sigma', X)$. For any element in $core(\sigma', X) - core(\sigma, X)$ there must be a path of edges not contingent upon X in σ' to $Q \cup E$ via some variables in $core(\sigma, X) \cup \{X\}$ (trivially, since $Q \cup E \subseteq core(\sigma, X) \cup \{X\}$). Let Y and Z be one such parent-child pair in σ' s.t. $Y \in core(\sigma', X) - core(\sigma, X)$ and $Z \in core(\sigma, X) \cup \{X\}$. Now, all the variables referred in $\mathcal{T}_Z$ upto the first reference of X (if any) would also be in $core(\sigma, X)$ since they have a edge to Z which is not contingent on X. Since σ and σ' agree on $core(\sigma, X)$, the evaluation of $\mathcal{T}_Z$ would follow an identical path in σ and σ' upto the first reference of X. But since Y is not referred to after X while evaluating $\mathcal{T}_Z$ in σ' it follows that $Y \in core(\sigma', X)$.

Let, $Y \in \Upsilon(\sigma, X)$, i.e. Y is a child of X in σ and $Y \in core(\sigma, X)$. From the above result $Y \in core(\sigma', X)$ and we will next show that Y is a child of X in σ'. Consider the evaluation path of $\mathcal{T}_Y$ in σ. All the variables that are referred before X are also in $core(\sigma, X)$ by definition. Since these variables will have the same value in σ', it follows that the evaluation of $\mathcal{T}_Y$ in σ' will lead to X being referred. In other words, X is a parent of Y in σ' which implies that $\Upsilon(\sigma, X) \subseteq \Upsilon(\sigma', X)$. By symmetry, $\Upsilon(\sigma', X) \subseteq \Upsilon(\sigma, X)$ □

Proposition 4. *For any two minimal self-supporting instantiations, σ and σ', there is at most one variable X common to them s.t. σ and σ' agree on $core(\sigma, X)$ and differ on X.*

Proof. Assume to the contrary that there exist two such variables X and Y. Now, since σ and σ' agree on $core(\sigma, X)$ but differ on Y it follows that $Y \notin core(\sigma, X)$. Hence Y can't be in $Q \cup E$. But since σ is a minimal instantiation,

Y must have a path to $Q \cup E$. Now consider the shortest path of Y to $Q \cup E$. Some edge, W-Z in this path must be contingent on X. Hence we can construct a path from X to $Q \cup E$ via Z which can't be contingent on Y (otherwise, Y would have a shorter path to $Q \cup E$). This implies that $X \in core(\sigma, Y)$ but then σ and σ' agree on X, a contradiction. □

We will now describe the weights for each of the partial instantiation σ_i that are constructed for each value in $dom(X)$.

$$w(\sigma_i) = \frac{p_X(\sigma_{iX}|\sigma_{i\mathcal{T}_X})}{|V(\sigma_i)|} \prod_{Y \in \Upsilon(\sigma,X)} p_Y(\sigma_{iY}|\sigma_{i\mathcal{T}_Y}) \tag{6}$$

Note that, up to a multiplicative constant, this equation reduces to Equation 5 if X is not a switching variable.

It only remains to show that detailed balance holds between any two minimal instantiations σ_0 and σ_1. It follows from propositions 3 and 4 that there is at most one shared variable X such that a transition is possible between σ_0 and σ_1 by sampling X. Thus the only way for a transition to occur is in σ_0 to first select X for sampling with probability $\frac{1}{|V(\sigma_0)|}$. Next, the new variables in σ_1, $\psi(\sigma_0, X, \sigma_1) = vars(\sigma_1) - core(\sigma_0, X) - \{X\}$ must be sampled with probability $\prod_{Y \in \psi(\sigma_0,X,\sigma_1)} p_Y(\sigma_{1Y}|\sigma_{1\mathcal{T}_Y})$. Finally, we must select σ_1 out of all the other random instantiations with probability, $\frac{w(\sigma_1)}{w(\sigma_0)+\ldots+w(\sigma_{n-1})}$. Since the last quantity is a random variable which depends on the choices made in constructing $\sigma_2, \ldots, \sigma_{n-1}$ we can take its expectation giving, $q(\sigma_0 \rightarrow \sigma_1)$:

$$\frac{1}{|V(\sigma_0)|} \prod_{Y \in \psi(\sigma_0,X,\sigma_1)} p_Y(\sigma_{1Y}|\sigma_{1\mathcal{T}_Y}) E\left[\frac{w(\sigma_1)}{\sum_{i=0}^{n-1} w(\sigma_i)}\right] \tag{7}$$

We can construct a similar expression for the reverse move probability and note that the numerator in the expectation is a constant and the rest of the expectation doesn't depend on which of σ_0 or σ_1 we start out with. Thus $\frac{q(\sigma_0 \rightarrow \sigma_1)}{q(\sigma_1 \rightarrow \sigma_0)}$ is:

$$\frac{|V(\sigma_1)|}{|V(\sigma_0)|} \frac{w(\sigma_1)}{w(\sigma_0)} \frac{\prod_{Y \in \psi(\sigma_0,X,\sigma_1)} p_Y(\sigma_{1Y}|\sigma_{1\mathcal{T}_Y})}{\prod_{Y \in \psi(\sigma_1,X,\sigma_0)} p_Y(\sigma_{0Y}|\sigma_{0\mathcal{T}_Y})}$$

Substituting for $w(\sigma_1)$ and $w(\sigma_0)$:

$$\frac{q(\sigma_0 \rightarrow \sigma_1)}{q(\sigma_1 \rightarrow \sigma_0)} = \frac{p_X(\sigma_{1X}|\sigma_{1\mathcal{T}_X})}{p_X(\sigma_{0X}|\sigma_{1\mathcal{T}_X})} \prod_{Y \in \Upsilon(\sigma,X)} \frac{p_Y(\sigma_{1Y}|\sigma_{1\mathcal{T}_Y})}{p_Y(\sigma_{0Y}|\sigma_{0\mathcal{T}_Y})} \cdot \frac{\prod_{Y \in \psi(\sigma_0,X,\sigma_1)} p_Y(\sigma_{1Y}|\sigma_{1\mathcal{T}_Y})}{\prod_{Y \in \psi(\sigma_1,X,\sigma_0)} p_Y(\sigma_{0Y}|\sigma_{0\mathcal{T}_Y})}$$

Observe that the only terms missing from $\frac{p(\sigma_1)}{p(\sigma_0)}$ above are those for variables in $core(\sigma, X) - \Upsilon(\sigma, X)$. However, if $Y \in core(\sigma, X)$ then $\sigma_Y = \sigma'_Y$ and further if $Y \notin \lambda(\sigma, X)$ all the parents of Y are also in $core(\sigma, X)$ and hence have the same values in σ and σ'. Thus these variables have identical values and distributions in σ_0 and σ_1 and their terms cancel out. Finally,

$$\frac{q(\sigma_0 \rightarrow \sigma_1)}{q(\sigma_1 \rightarrow \sigma_0)} = \frac{p(\sigma_1)}{p(\sigma_0)}$$

5 BLOG Compiler

We have implemented our algorithm in a new implementation of the BLOG language, which we will refer to as *blogc*[1] . The broad outline of our implementation is similar to Milch's public-domain version except in two significant aspects.

First, for variables with finite domain (or unknown, but finite domain) we always use Gibbs sampling. By statically analyzing the structure of the model we can determine which variables are switching variables, which ones need to be resampled for each transition, etc. Based on the analysis, appropriate code is generated that does the actual sampling and reporting.

Consider, as an example, the BLOG model in Figure 5. This model describes the prior distribution of two types of aircraft – fixed-wing planes and helicopters. These planes may produce an arbitrary number of blips on the radar (The fact that plane a produces a blip b is represented by setting $Source(b) = a$). Further, helicopters due to the interaction of their rotor with the radar beam can produce blade-flashes in the radar blip. In this model, the variable $RotorLength(a)$ for all aircraft a can easily be Gibbs sampled. If $WingType(a) = Helicopter$ then $RotorLength(a)$ can be either Short or Long, otherwise it can only be null (as per BLOG semantics for a missing else clause). While compiling the model we can detect that the children variables of $WingType(a)$ in any instantiation are all the $BladeFlash(b)$ variables such that $Source(b) = a$. In order to speed up the computation of the Gibbs weights at runtime, we maintain a list, for each object a of type $Aircraft$, of all objects b of type $Blip$ such that $Source(b) = a$.

The variable $WingType(a)$ is more interesting. It can only have two possible values, but, since it is a switching variable, care has to be taken when sampling it. In particular, the variable $RotorLength(a)$ has to be uninstantiated. This is because all the children edges from $RotorLength(a)$ are contingent on the value of $WingType(a)$. Note that $Source(b)$ for all objects b of type $Blip$ is also a switching variable. However, in this case the decision to uninstantiate a variable $WingType(a)$ s.t. $Source(b) = a$ depends on whether there exists another object b_1 s.t. $Source(b_1) = a$.

The second major difference in our implementation is the handling of number variables. Instead of directly sampling the number variables, our implementation proposes birth and death moves. In the radar example, for each object w

[1] blogc is available for download from: http://code.google.com/p/blogc/

of type $WingType$, we generate an $Aircraft$ object that has no blips assigned to it. The death move kills off such objects with no blips. In order to get faster mixing, we allow some extra flexibility in the birth and death move during the burn-in samples. During these samples, birth and death moves ignore the probability of children variables. To understand the motivation, assume for a moment that the expected number of blips for a given aircraft was 1 million. Now, a birth move which proposes an aircraft with 0 blips would be almost certainly rejected. By allowing such birth moves during the burn-in we give the inference engine an opportunity to later attach blips to the aircraft.

6 Experimental Results

We would like to have compared blogc with the generic Metropolis-Hastings inference engine provided with BLOG, which we will refer to as BLOG-MH, in terms of the number of samples. However, due to differences in the definition of a *sample* such a direct comparison is not feasible. (A sample in BLOG-MH is generated after sampling the value of a randomly selected variable while blogc collects a sample after sampling all instantiated variables once.) Instead, we compare convergence speed w.r.t. time. In the following three models each inference engine is run for a varying number of samples, where a sample is as defined by that inference engine. For each number of samples, inference is repeated 20 times with a different random seed and the variance of a query variable is plotted against the average elapsed time (in seconds).

First, we evaluate on the Alarm network of (Beinlich *et al.*, 1989) available from the Bayes Network Repository[2] (Friedman *et al.*, 1997). This is a Bayes Net with 37 discrete random variables of which we observe 9. The results are summarized in Figure 6. The important thing to note is that the variance achieved by blogc in less than 1 second is much better than that achieved by BLOG-MH in over 20 seconds.

Next, we consider the model in Figure 7 which is the urns-and-balls example of (Milch *et al.*, 2005b) with a slight twist. Balls have a weight instead of a discrete color. Figure 8 shows that blogc converges significantly faster.

Our final result is on the radar example of Figure 5. For this model we experimented running blogc without the logic which detects that $RotorLen(a)$ must be uninstantiated when sampling $WingType(a)$. This mode is labeled as blogc-noblock in Figure 9. As before, blogc is significantly faster than BLOG-MH. It is also 3 to 7 times faster than blogc-noblock. The fact that blogc-noblock converges at all is due to the birth move which creates new aircraft for each wingtype and samples their $RotorLen$ variable. Later, the move which resamples $Source(b)$ for each blip has the opportunity to select this new aircraft. These two moves thus compensate for the fact that the move which attempts to sample $WingType(a)$ is always rejected.

In follow-on work, we plan to demonstrate inference performance comparable to model-specific inference code for a number of widely used statistical models.

[2] http://compbio.cs.huji.ac.il/Repository/

```
type AircraftType;
type Length;
type Aircraft;
type Blip;

origin AircraftType WingType(Aircraft);
random Length RotorLength(Aircraft);
origin Aircraft Source(Blip);
random Boolean BladeFlash(Blip);

guaranteed AircraftType Helicopter, FixedWingPlane;
guaranteed Length Short, Long;

#Aircraft(WingType = w)
  if w = Helicopter then
    ~Poisson [1.0]
  else
    ~Poisson [4.0];

#Blip ~Poisson[2.0];

#Blip(Source = a) ~ Poisson[1.0];

RotorLength(a) {
  if WingType(a) = Helicopter then
    ~TabularCPD [[0.4, 0.6]]
};

BladeFlash(b) {
  if Source(b) = null then
    ~Bernoulli [.01]
  elseif WingType(Source(b)) = Helicopter then
    ~TabularCPD[[.9,.1],[.6,.4]]
                (RotorLength(Source(b)))
  else
    ~Bernoulli [.1]
};

obs {Blip b} = {b1, b2, b3, b4, b5, b6};

obs BladeFlash(b1) = true;
obs BladeFlash(b2) = false;
obs BladeFlash(b3) = false;
obs BladeFlash(b4) = false;
obs BladeFlash(b5) = false;
obs BladeFlash(b6) = false;

query WingType(Source(b1));
query WingType(Source(b2));
query WingType(Source(b3));
query WingType(Source(b4));
query WingType(Source(b5));
query WingType(Source(b6));
```


Figure 5: Example of helicopters and fixed-wing planes being detected by a radar

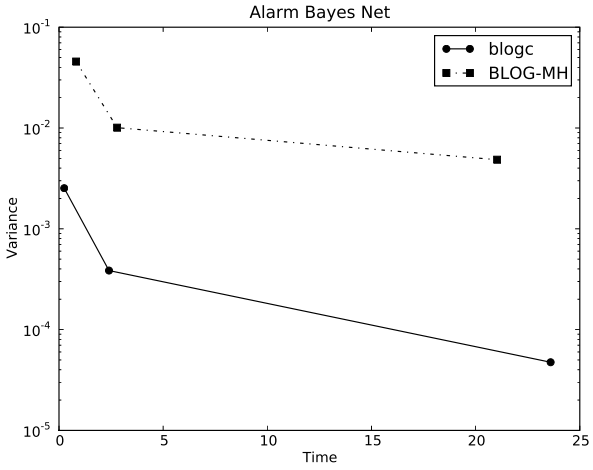

Figure 6: Results on the Alarm Bayes Net

7 Conclusions

We have demonstrated a significant improvement in inference performance for models written in the BLOG language. Our Gibbs sampling algorithm for CBNs and our compiler techniques for generating efficient inference code are generally applicable to all open-universe stochastic languages.

Acknowledgements

This work wouldn't have beeen possible without the considerable assistance provided by Brian Milch to make the models presented here work in BLOG-MH. Matthew Can provided a translation of the Alarm Bayes Net to BLOG. Finally, the first author wishes to thank his family for their boundless patience and support during this work.

References

Beinlich, I., Suermondt, G., Chavez, R., & Cooper, G. 1989. The ALARM monitoring system: A case study with two probabilistic inference techniques for belief networks. *In: Proc. 2'nd European Conf. on AI and Medicine*. Springer-Verlag, Berlin.

Friedman, N., Goldszmidt, M., Heckerman, D., & Russell, S. 1997. Challenge: Where is the impact of Bayesian networks in learning? *In: IJCAI*.

Geman, S., & Geman, D. 1984. Stochastic Relaxation, Gibbs Distributions,

```
type Ball;
type Draw;

random Real TrueWeight(Ball);
random Ball BallDrawn(Draw);
random Real ObsWeight(Draw);

guaranteed Draw Draw[10];

#Ball ~ Poisson[6.0];

TrueWeight(b) ~ UniformReal [0.0, 100.0];

BallDrawn(d) ~ UniformChoice({Ball b});

ObsWeight(d) {
  if BallDrawn(d) != null then
    ~UnivarGaussian[1](TrueWeight(BallDrawn(d)))
};

obs ObsWeight(Draw1) = 61.8;
obs ObsWeight(Draw2) = 64.4;
obs ObsWeight(Draw3) = 17.7;
obs ObsWeight(Draw4) = 81.8;
obs ObsWeight(Draw5) = 40.9;
obs ObsWeight(Draw6) = 81.9;
obs ObsWeight(Draw7) = 82.3;
obs ObsWeight(Draw8) = 82.9;
obs ObsWeight(Draw9) = 82.6;
obs ObsWeight(Draw10) = 60.8;

query TrueWeight(BallDrawn(Draw1));
query TrueWeight(BallDrawn(Draw2));
query TrueWeight(BallDrawn(Draw3));
query TrueWeight(BallDrawn(Draw4));
query TrueWeight(BallDrawn(Draw5));
query TrueWeight(BallDrawn(Draw6));
query TrueWeight(BallDrawn(Draw7));
query TrueWeight(BallDrawn(Draw8));
query TrueWeight(BallDrawn(Draw9));
query TrueWeight(BallDrawn(Draw10));
```

Figure 7: Example of selecting balls with replacement from an urn and measuring their weight

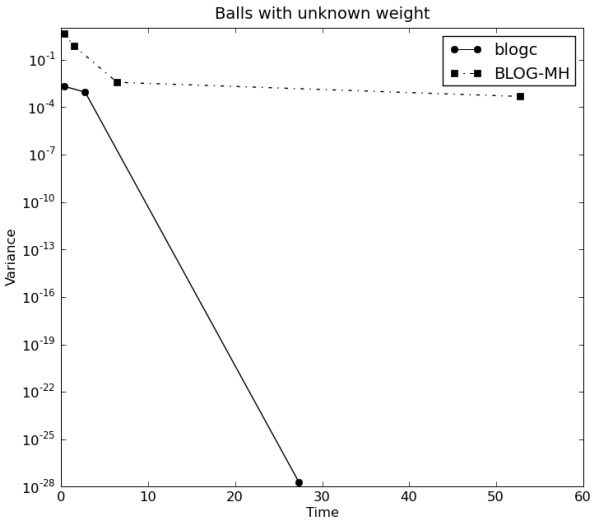

Figure 8: Balls with unknown weights

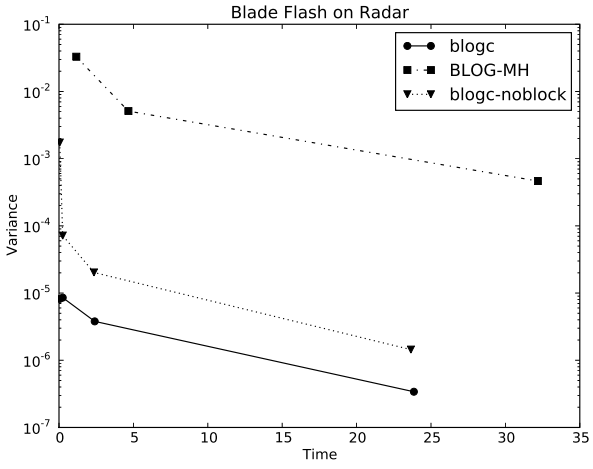

Figure 9: Results on the radar model

and the Bayesian Restoration of Images. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, **6**(6), 721–741.

Goodman, Noah, Mansinghka, Vikash, Roy, Daniel, Bonawitz, Keith, & Tenenbaum, Joshua. 2008. Church: a language for generative models. *In: UAI*.

Milch, Brian, & Russell, Stuart. 2006. General-Purpose MCMC Inference over Relational Structures. *Pages 349–358 of: Proceedings of the Proceedings of the Twenty-Second Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-06)*. Arlington, Virginia: AUAI Press.

Milch, Brian, Marthi, Bhaskara, Sontag, David, Russell, Stuart, Ong, Daniel L., & Kolobov, Andrey. 2005a. Approximate Inference for Infinite Contingent Bayesian Networks. *Pages 238–245 of: In Proc. 10th AISTATS*.

Milch, Brian, Marthi, Bhaskara, Russell, Stuart J., Sontag, David, Ong, Daniel L., & Kolobov, Andrey. 2005b. BLOG: Probabilistic Models with Unknown Objects. *Pages 1352–1359 of: IJCAI*.

Neal, Radford M. 2000. Markov Chain Sampling Methods for Dirichlet Process Mixture Models. *Journal of Computational and Graphical Statistics*, **9**(2), 249–265.

Spiegelhalter, David, Thomas, Andrew, Best, Nicky, & Gilks, Wally. 1996. *BUGS: Bayesian Inference using Gibbs Sampling, Version 0.50*. Tech. rept.

Tait, Peter. 2009. *Introduction to Radar Target Recognition*. The Institution of Engineering and Technology, United Kingdom.