Automatic Inference in BLOG

Nimar S. Arora and **Stuart Russell**
University of California, Berkeley
{nimar,russell}@ cs.berkeley.edu

Erik Sudderth
Brown University
sudderth@cs.brown.edu

Abstract

BLOG is a powerful language to express models with an unknown number of objects and identity uncertainty. Current inference engines for BLOG are either too slow or require users to write a model-specific proposal distribution. We describe here, ongoing work to design a new, fast, generic inference engine for BLOG called *blogc*. The new implementation uses Gibbs sampling for finite-valued variables and performs an analysis of the model to generate customized sampling code in *C*. We describe our algorithms and methods in the context of various commonly used models and demonstrate significant performance improvement.

1 Introduction

The BLOG language (Milch et al. 2005) is a first-order, open-universe probabilistic language which makes it very easy to write models with an unknown number of objects and identity uncertainty. A typical example is the Balls and Urn model in Figure 1. In this model, there are an unknown number of balls in an urn. The color of each ball can be either blue or green with equal probability. In each draw, one of the balls is sampled uniformly at random with replacement and its color is observed with some probability of observing the wrong color. Now, given the observed colors of some number of draws we need to infer the distribution of the number of balls.

The most general method for inference in BLOG models is 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 (Milch and Russell 2006). 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. BLOG also provides the users the ability to write model-specific proposal distributions but this limits the widespread usage of the language.

In this paper we describe a new implementation of BLOG, called *blogc*, which is based on the premise that efficient inference in BLOG should be possible without user assistance. blogc is a compiler for BLOG. Given a model, the observations, and the queries, blogc generates *C* code for MCMC inference in the given model. The generated inference code always does Gibbs sampling (Geman and Geman 1984) for all finite-valued variables. For example, the $TrueColor(b)$ of each ball (see Figure 1) can be Gibbs sampled. In some cases, structurally related variables are block sampled for faster convergence. Number variables are sampled with birth and death moves. Again, in the Balls and Urn example, birth and death moves propose new balls or delete existing ones. Finally, the compiler determines the children variables by analyzing the model. For example, the analysis would conclude that $ObsColor(d)$ is a child of $TrueColor(b)$ if and only if $BallDrawn(d) = b$.

```
type Ball;
type Draw;
type Color;

random Color TrueColor(Ball);
random Ball BallDrawn(Draw);
random Color ObsColor(Draw);

guaranteed Color Blue, Green;
guaranteed Draw Draw[10];

#Ball ~ Poisson[6.0];

TrueColor(b) ~ TabularCPD[[0.5, 0.5]];

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

ObsColor(d) {
if BallDrawn(d) != null then
~ TabularCPD[[0.8,0.2],[0.2,0.8]](TrueColor(BallDrawn(d)))
};

obs BallDrawn(Draw1) = Blue;
...
obs BallDrawn(Draw10) = Green;

query #{Ball b};
```

Figure 1: The Balls and Urn example in BLOG

The rest of the paper is organized as follows: In Section 2 we describe the overall sampling algorithms in blogc while in Section 3 we describe the model analysis and the design of the generated code which allow the sampling to run efficiently. Finally, in Section 4 we present results on a few models.

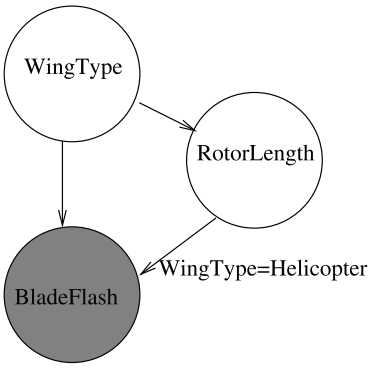

Figure 2: A Contingent Bayes Net for radar detection of blade flash.

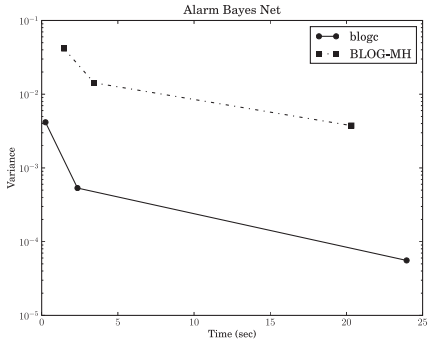

Figure 3: Results on the Alarm Bayes Net

2 Sampling Partial Worlds

Although Gibbs sampling is well understood for Bayes Net or closed-world models in general, there are subtle issues which arise in open-universe models. In open-universe models since inference is done over self-supporting partial instantiations, there may arise a situation where changing the value of a variable may make the instantiation no longer self-supporting. Consider the following simplified example:

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 2).*

Now, assume that we have observed that $BladeFlash$ is $True$ and we are trying to query the $WingType$. Assume further that our current state is the minimal self-supporting instantiation – $[BladeFlash = True, WingType = FixedWingPlane]$. Clearly, we can't immediately apply Gibbs sampling for $WingType$ since we would have to compute the probability of $BladeFlash$ given $WingType = Helicopter$, which in turn requires the value of $RotorLength$, a variable missing from our current instantiation. One solution is to use the auxiliary variable method as used for Gibbs sampling in Dirichlet Process

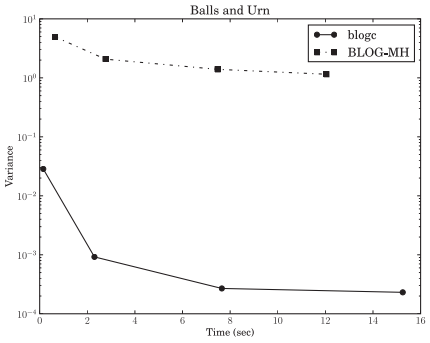

Figure 4: Results on the Balls and Urn model.

```
type Cluster;
type Galaxy;

random Real ClusVelocity(Cluster);
random Cluster OrigCluster(Galaxy);
random Real Velocity(Galaxy);

#Cluster ~ Poisson [10.0];
guaranteed Galaxy Galaxy[82];

ClusVelocity(c) ~ UniformReal[5000.0, 50000.0];

OrigCluster(g) ~ UniformChoice({Cluster c});

Velocity(g)
 ~ UnivarGaussian[10000.0](ClusVelocity(OrigCluster(g)));

obs Velocity(Galaxy1) = 9172;
...
obs Velocity(Galaxy82) = 34279;

query ClusVelocity(OrigCluster(Galaxy1));
...
query ClusVelocity(OrigCluster(Galaxy82));
query #{Cluster c};
```

Figure 5: Example of galaxy velocities with a Gaussian distribution around the cluster velocity.

Mixture Models (Neal 2000) and add the auxiliary variable $RotorLength$ first. However, if we were to sample $RotorLength$ with the current value of $WingType$, i.e. $FixedWingPlane$, then we would only get $null$ which would prevent the chain from mixing since helicopters always have non-null $RotorLength$.

The solution that we have adopted in such cases is to do Gibbs sampling over partial instantiations, $\sigma_0 \ldots \sigma_{n-1}$, constructed for each value of the sampled variable, $WingType$. Variables needed to make a σ_i self-supporting are sampled in σ_i. For example, in the partial instantiation $[WingType = Helicopter, BladeFlash = True]$ the variable $RotorLength$ is sampled with its distribution given $WingType = Helicopter$ to make the instantiation self-supporting. In general, Gibbs sampling over partial worlds may also require re-instantiating some of the variables. Full details of the algorithm and the proof of correctness for arbitrary Contingent Bayes Nets (CBN) are described in (Arora et al. 2010). In blogc, for each variable we look for child

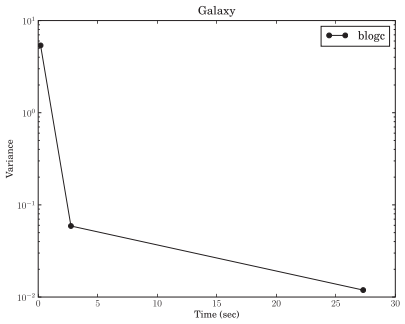

Figure 6: Results on the Galaxy dataset

variables whose existence in the minimal partial instantiation is contingent upon the value of the variable. Such child variables are automatically re-sampled when Gibbs sampling the variable. In the radar example, $RotorLength$ was a child of $WingType$ whose existence in a minimal self-supporting world was contingent upon the value of $WingType$. Hence $RotorLength$ is always resampled when Gibbs sampling $WingType$.

In open-universe models, the ability to change the number of objects in the instantiation is clearly very critical. In blogc for each non-guaranteed type whose objects are not all observed, a birth-death move adds or deletes one object of that type through a Metropolis Hastings move. The added object has no variable pointing to it. For example, if we are adding a new Ball in the the Balls and Urn example then none of the BallDrawn variables point to this ball. Ofcourse, this doesn't imply that the new object has no children. In the example under consideration, all BallDrawn variables are children of all the Ball objects. This birth move can have low acceptance probability in the early samples for some models. For example, proposing a second Ball would bring down the probability of the BallDrawn variables by a factor of 2^{-10}. In order to facilitate faster mixing, we allow the birth-death moves to not check child probabilities during the burn-in samples. We are working on a new birth-death move which simultaneously proposes new objects and modifies other variables to point to it.

3 Code Generation

In order to improve the running time of the sampler, blogc generates customized *C* code for each model. The overall design of the generated code is as follows.

- Each BLOG type gets its own C type.
- BLOG functions are stored as attributes of the C type corresponding to the types of the parameters of the BLOG function. For example, the function TrueColor is stored as an attribute of the Ball type.
- Each type has a list of objects of other types which point to it. We will refer to this as a *referring list*. For instance, the Ball type keeps a referring list of Draw objects which point to it through the BallDrawn variable. A type may have multiple referring lists.

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

origin AircraftType WingType(Aircraft);
random Real Location(Aircraft);
random Length RotorLength(Aircraft);
origin Aircraft Source(Blip);
random Real BlipLocation(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) {
if WingType(a) = Helicopter then
~ Poisson[1.0]
else
~ Poisson[2.0]
};

Location(a) ~ UniformReal [100.0, 1000.0];

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

BlipLocation(b) {
if Source(b) != null then
~ UnivarGaussian[10.0] (Location(Source(b)))
else
~ UniformReal [50.0, 1050.0]
};

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]
};
```

Figure 7: Example of aircrafts generating blips on a radar.

- Guaranteed objects or fully observed objects are stored as an array while other objects are stored as a linked list which can be altered by birth-death moves.
- A customized sampler is generated for each BLOG function and a customized birth-death sampler is generated for each BLOG type.
- A main driver function parses the command line arguments (for the number of samples etc.), initializes the observed variables and then repeatedly invokes the sampler for the unobserved variables and birth-death moves for types with unknown number of objects.

One of the biggest costs in an MCMC sampler is maintaining and traversing the parent-child dependencies between variables. In blogc, model analysis is critical to compactly and efficiently maintaining such dependencies. We describe here how we handle three important types of child variables.

```
obs {Blip b} = {b1, b2, b3, b4, b5, b6, b7};
obs BladeFlash(b1) = true;
obs BlipLocation(b1) = 500.1;
obs BladeFlash(b2) = false;
obs BlipLocation(b2) = 200.3;
obs BladeFlash(b3) = false;
obs BlipLocation(b3) = 197.5;
obs BladeFlash(b4) = false;
obs BlipLocation(b4) = 600.0;
obs BladeFlash(b5) = true;
obs BlipLocation(b5) = 601.3;
obs BladeFlash(b6) = false;
obs BlipLocation(b6) = 598.7;
obs BladeFlash(b7) = false;
obs BlipLocation(b7) = 300.3;
query WingType(Source(b1));
query Location(Source(b1));
query WingType(Source(b2));
query Location(Source(b2));
query WingType(Source(b3));
query Location(Source(b3));
query WingType(Source(b4));
query Location(Source(b4));
query WingType(Source(b5));
query Location(Source(b5));
query WingType(Source(b6));
query Location(Source(b6));
query WingType(Source(b7));
query Location(Source(b7));

query #{Aircraft a};
```

Figure 8: Sample data and query for the model in Figure 7

A variable F is a *static child* of G if F is a child of G in all possible worlds. For example, $ObsColor(d)$ is always a child of $BallDrawn(d)$. Such dependencies don't need to be explicitly maintained. The sampler for a variable contains code referring to the static children directly by name. Another example is that all the $BallDrawn(d)$ variables are static children of all the Ball objects and this dependency comes into play during a birth-death move for the Ball objects.

A variable F is a *dynamic child* of G if there exists another variable H and a value O such that F is a child of G in all worlds where $H = O$. For example, $ObsColor(d)$ is a child of $TrueColor(b)$ in all worlds

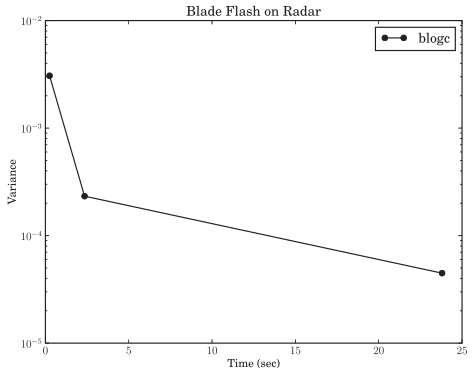

Figure 9: Results on the Radar model.

where $BallDrawn(d) = b$. Such dependencies are maintained by the referring lists. The sampler for G consults the referring list to determine all the appropriate children. For example, the sampler for $TrueColor(b)$ scans all the Draw objects in the BallDrawn referring list for b to deduce the $ObsColor(d)$ children.

A variable F is a *contingent child* of G if F is a static child of G and all the children of F refer to G before their first reference of F. For example, RotorLength is a contingent child of WingType. We use the knowledge of this dependency to resample F in the sampler for G.

4 Results

To demonstrate the improvement in using Gibbs sampling we compared the performance of the generic Metropolis-Hastings sampler provided with BLOG (BLOG-MH) with blogc on a Bayes Net. We used the Alarm network of (Beinlich et al. 1989) available from the Bayes Network Repository[1] (Friedman et al. 1997). This is a Bayes Net with 37 discrete random variables of which we observe 9. For one of the queried variables we measured the variance w.r.t. sampling time. The results are summarized in Figure 3. 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.

For the Balls and Urn model we compared the variance of the posterior average number of balls. Figure 4 demonstrates the relative convergence speed. Again, blogc converges to a reasonable value in under a second.

Probabilistic mixture models are often used for the visualization and analysis of scientific data. In the widely studied *galaxy* dataset, the number of modes in the distribution of galaxy velocities has implications for hypotheses about the universe's origins (Roeder 1990). Previous Bayesian analyses of this data have required complex, hand-designed sampling algorithms to search over mixtures of varying order (Escobar and West 1995; Green and Richardson 2001). Using the *blogc* compiler, however, this data can be automatically analyzed based on the high-level model specification in Fig. 5.

Figure 6 plots the variance in the posterior average number of clusters versus time.

For the final example, we consider a typical BLOG model in which objects generate other objects. The BLOG model is given in Figure 7. In this model, objects of type $AircraftType$ ($Helicopter$ or $FixedWingPlane$) generate objects of type $Aircraft$. The originating $AircraftType$ object for each $Aircraft$ object, a, is recorded in the origin function $WingType(a)$. The BLOG language also allows one to specify the distribution of the number of generated objects. In this model, the prior expects one helicopter and four fixed-wing planes on average. The $Aircraft$ objects can in turn generate $Blip$ objects through the $Source$ origin function, where helicopters generate one blip on average and aircrafts generate two. There is also the possibility of 2 false blips on average. The blips may also contain a BladeFlash depending on the type of the aircraft as

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

in Example 1. Sample data for the radar model is shown in Figure 8. In this example there are seven blips. In two of the blips, b_1 and b_5, a blade flash was also observed. However, since blip b_5 is located near two other blips it is less likely to be a helicopter than blip b_1. Figure 9 plots the variance in the posterior average number of aircraft.

BLOG-MH was not able to do inference in the last two models because it couldn't construct a feasible world even after running for over an hour. blogc, on the other hand, doesn't require that the initial world be feasible. It relies on the sampler to quickly get it to a feasible world. All the results in this section were computed by running the samplers 20 times for varying number of samples. 10% of all the samples were discarded as burn-in.

5 Conclusions

With a combination of better inference algorithms and model analysis, blogc has made a significant improvement in the state of the art for automatic inference in first-order probabilistic languages.

References

Arora, N. S.; de Salvo Braz, R.; Sudderth, E.; and Russell, S. J. 2010. Gibbs sampling in open-universe stochastic languages. Technical Report UCB/EECS-2010-34, EECS Department, University of California, Berkeley.

Beinlich, I.; Suermondt, G.; Chavez, R.; and 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.

Escobar, M. D., and West, M. 1995. Bayesian density estimation and inference using mixtures. *Journal of the American Statistical Association* 90(430):577–588.

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

Geman, S., and Geman, D. 1984. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Trans. on Pattern Analysis and Machine Intelligence* 6(6):721–741.

Green, P. J., and Richardson, S. 2001. Modelling heterogeneity with and without the Dirichlet process. *Scandinavian Journal of Statistics* 28:355–375.

Milch, B., and Russell, S. 2006. General-purpose mcmc inference over relational structures. In *Proceedings of the Proceedings of the Twenty-Second Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-06)*, 349–358. Arlington, Virginia: AUAI Press.

Milch, B.; Marthi, B.; Russell, S. J.; Sontag, D.; Ong, D. L.; and Kolobov, A. 2005. Blog: Probabilistic models with unknown objects. In *IJCAI*, 1352–1359.

Neal, R. M. 2000. Markov chain sampling methods for dirichlet process mixture models. *Journal of Computational and Graphical Statistics* 9(2):249–265.

Roeder, K. 1990. Density estimation with confidence sets exemplified by superclusters and voids in the galaxies. *Journal of the American Statistical Association* 85(411):617–624.

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