Bayesian Learning of Bayesian Networks with Informative Priors

Nicos Angelopoulos NICOS@CS.YORK.AC.UK
James Cussens JC@CS.YORK.AC.UK
Department of Computer Science
University of York
Heslington, York, YO10 5DD, UK

Editor: Some editor

Abstract

This paper presents and evaluates an approach to Bayesian model averaging where the models are Bayesian nets (BNs). Prior distributions are defined using stochastic logic programs and the MCMC Metropolis-Hastings algorithm is used to (approximately) sample from the posterior. Experiments using data generated from known BNs have been conducted to evaluate the method. The experiments used 6 different BNs and varied: the structural prior, the parameter prior, the Metropolis-Hasting proposal and the data size. Each experiment was repeated three times with different random seeds to test the robustness of the MCMC-produced results. Our results show that with effective priors (i) robust results are produced and (ii) informative priors improve results significantly.

Keywords: Prior knowledge, Bayesian inference, Bayesian model averaging, Markov chain Monte Carlo, loss functions, stochastic logic programs.

1. Introduction

The Bayesian approach to machine learning is (conceptually) remarkably simple. It assumes that, before any analysis of data, a space of models (or *hypotheses*) is defined, and that one of these models is the true model of the process which led to the data being observed. Further it is assumed that a prior distribution is defined over this model space. The prior probability of any given model represents the modeller's belief that it is the true model based on information available prior to the data. Prior plus data define the posterior distribution over model space, and from a pure Bayesian viewpoint the posterior is the endpoint of statistical inference.

In practice, however, the posterior is rarely the final result since it is generally used to make rational decisions. For example, a posterior distribution over a space of classifiers can be used to classify new examples: the example is just classified to whichever class has minimal expected cost according to the posterior. If misclassification costs are equal this is just the most likely class for the example according to the posterior.

If the end-goal is to make rational decisions (for example, classification decisions) based on observed data, then there are strong arguments for a Bayesian approach (Howson and Urbach, 1989) which conclude that the posterior distribution is what is required to do this. Also the posterior (suitably presented and/or summarised) may permit insight into the

combined information provided by prior and data. Naturally, though, there are important questions which a Bayesian analysis does not address. Bayesian inference (at least in its 'vanilla' form) is inference conditional on the actually observed data, it is not interested in other data which might have been observed but were not. If, for example, we want bounds on the probability of observing a dataset which leads some learning algorithm to produce a result which is 'approximately correct' then non-Bayesian results on uniform convergence are needed (Vapnik and Chervonenkis, 1971).

Even when a Bayesian analysis is theoretically desirable, there are often large practical obstacles to its application. Anyone who has tried to shoe-horn real prior knowledge, with all its heterogeneity, into a usable prior distribution will have encountered a problem this paper aims to reduce. Formalising prior knowledge so that it is accessible to Bayesian inference requires a representation language which straddles the gap between the prior knowledge in a human's brain and that expressed by a probability distribution. In this paper we propose first-order logic with an additional probabilistic component (specifically, stochastic logic programs) as such a language. It is of course no great discovery that first-order logic forms a good basis for formalising human thoughts: it was specifically designed to do just that (Frege, 1879).

A second problem for the Bayesian approach is getting hold of the posterior. In many cases, particularly in parametric Bayesian inference, a *conjugate* prior distribution is used so that the computation and representation of the posterior are greatly simplified. In the current paper the focus is on non-parametric Bayesian inference: both prior and posterior are defined over a large space of models. Although in this paper, in contrast to some of our other work (Angelopoulos and Cussens, 2004a), each model space is finite, they are nonetheless generally too big to represent explicitly. This means that a simple tabular representation of prior and posterior is ruled out. Here we adopt a common approach to this problem: a Markov Chain Monte Carlo (MCMC) algorithm is used to construct an approximate sample from the posterior and the sample is then used as a proxy for the actual posterior.

Using MCMC, an approximate sample from the true posterior distribution is generated, and from a pure Bayesian point of view, the only learning result to evaluate is the accuracy of this approximation. Our MCMC runs have been repeated with different random seeds to test that similar estimates of posterior quantities (e.g. expected losses) are produced each time. We also extract single models ('best guesses' of the true model) from the MCMC sample to evaluate our results from the more usual non-Bayesian perspective.

The paper is structured as follows. Section 2 is a survey of existing methods of incorporating prior information into Bayesian net learning. Section 3 shows how to use stochastic logic programs to define prior distributions. Section 4 describes how our SLP priors are 'folded into' the Metropolis-Hastings algorithm to generate approximate samples from the posterior. Sections 5 and 6 describe the experiments we have conducted and the results of those experiments, respectively. The paper ends with conclusions and suggestions for future work in Section 7.

2. Existing approaches to using prior information when learning the structure of Bayesian nets

In this section, we provide an overview of existing approaches to including prior knowledge into BN learning. However, it should be noted that most BN learning approaches make no significant attempt to integrate prior knowledge into the learning process. As Friedman and Koller (2003) note " ... relatively little attention has been paid to the choice of structure prior, and a simple prior is often chosen largely for pragmatic reasons. The simplest and therefore most common choice is a uniform prior over structures (Heckerman, 1998)" Madigan et al. (1995) state that "All of the reported applications of BMA use a uniform prior distribution, thereby implying that all models are equally likely *a priori*." They go on to argue that:

> ... in the context of knowledge-based systems, or indeed in any context where the primary aim of the modeling effort is to predict the future, such prior distributions are often inappropriate; one of the primary advantages of the Bayesian approach is that it provides a practical framework for harnessing all available resources including prior expert knowledge.

We agree with this position and our own work is an attempt facilitate harnessing expert knowledge. Moreover, despite the frequent use of non-informative priors there remains some work which seeks to incorporate prior knowledge into BN learning. We turn now to consider the various ways in which this has been done.

2.1 Hard constraints

The next step up in complexity from a uniform prior is to completely rule out particular structures. This approach has been applied in both Bayesian and non-Bayesian approaches. In the Bayesian case, one can either declare a uniform prior on the surviving structures or go on to specify some non-uniform prior using one of the techniques given in Sections 2.2–2.4.

An early non-Bayesian example of using hard constraints is the algorithm presented by Srinivas et al. (1990) which allows (but does not require) an expert to effect constraints of four different kinds:

- Declaring that a variable must be a root node
- Declaring that a variable must be a leaf node
- Declaring that one variable must be the parent of another
- Making explicit conditional independence declarations

The expert knowledge also defines a partial order between variables (a 'priority ordering') which declares an ancestor relation between pairs of nodes. This knowledge guides the learning of a BN by a non-Bayesian algorithm.

In Bayesian approaches a hard constraint sets the prior probability of the relevant structures to zero. It follows from Bayes theorem that such structures will also have posterior

probability of zero no matter how much the data might support them. An interesting suggested application of zero prior probabilities applies when (unusually) we consider structures containing extra 'hidden' variables not present in the data:

> One difficulty in considering the possibility of hidden variables is that there is an unlimited number of them and an unlimited number of belief-networks that can contain them ... Another approach is to specify explicitly nonzero priors for only a limited number of belief-network structures that contain hidden variables. (Cooper and Herskovits, 1992)

By far the most common hard constraint is to impose a total ordering $\prec$ on variables such that y is a parent of x ($y \in \Pi_x$) only if $y \prec x$. (This is equivalent to stating that y is an ancestor of x only if $y \prec x$.) This is the approach taken by Buntine (1991) and in the K2 algorithm (Cooper and Herskovits, 1992), for example. Such an ordering reduces the model space considerably. For example, for 10 variables there are about 4.2×10^{18} possible BN structures, but with an ordering $\prec$ this reduces to $2^{C(10,2)} = 2^{(10\times 9)/2} = 3.5 \times 10^{13}$ (Cooper and Herskovits, 1992).

As an alternative to supplying a total ordering on variables, the BIFROST algorithm (Højsgaard and Thiesson, 1995) asks the user to partition the variables into 'blocks' and supply a total ordering only on the blocks.

> Hence model selection in BIFROST demands that the user specifies a block recursive model by which the types of potential associations between variables are defined. BIFROST will then on the basis of a complete data set determine on which of the potential associations that actually holds. (Højsgaard and Thiesson, 1995)

In addition the user may also give information on the existence of links.

> It is possible to reduce the set of possible selectable models by specifying prior knowledge of some partial associations which definitely exist and some which definitely do not exist. (Højsgaard and Thiesson, 1995)

Using a greedy non-Bayesian approach, BIFROST induces a block-recursive model (aka a chain graph) not a Bayesian network. Since BIFROST only considers *decomposable* models, the resulting block-recursive model can always be translated into a Bayesian network.

Another common constraint is to limit the number of parents any variable may have (the indegree) (Cooper and Herskovits, 1992; Friedman and Koller, 2003). Such a constraint can be justified as more than just a convenience:

> There are few applications in which very large families are called for, and there is rarely enough data to support robust parameter estimation for such families. From a more formal perspective, networks with very large families tend to have low score. (Friedman and Koller, 2003)

The 'low score' referred to here is a measure of fit to data, so BNs with large families are likely to be largely ruled out by the data. Although a tight bound on indegree can reduce the set of possible models considerably, for n variables and an indegree k, the number of possible structures is still $2^{O(kn \log n)}$ (Friedman and Koller, 2003).

To finish this section on hard constraints, note that hard constraints are the most significant form of prior knowledge for the following reason supplied by Langseth and Nielsen (2003):

> The use of structural priors when learning BNs has received only little attention in the learning community. The most obvious reason is that in most cases the effect of the prior is dominated by the likelihood term, even for relatively small databases. One exception, however, is when some of the network structures are given zero probability a priori, in which case the data cannot change that belief.

2.2 Using priors on arcs

2.2.1 Using individual arc probabilities

The most natural non-uniform prior on Bayesian networks uses (i) a total ordering on variables and (ii) assumptions of independence so that the probability of a graph is simply the product of probabilities of its components. Call a prior which meets these two conditions *modular*. This approach originates with Cooper and Herskovits (1992) where $P(B_S)$, the probability of structure B_S is the product of the probabilities of parent sets (π_i^S) for each variable x_i:

> Assume that $P(B_S)$ can be calculated as $P(B_S) = \prod_{i=1,n} P(\pi_i^S \rightarrow x_i)$. Thus, for all distinct pairs of variables x_i and x_j, our belief about X_i having some set of parents is independent of our belief about x_j having some set of parents. (Cooper and Herskovits, 1992, p.18)

A second independence assumption can be used to decompose $P(\pi_i^S \rightarrow x_i)$:

> The probability $P(\pi_i^S \rightarrow x_i)$ could be assessed directly or derived using additional methods. For example, one method would be to assume that the presence of an arc in $\pi_i^S \rightarrow x_i$ is marginally independent of the presence of the other arcs there; if the marginal probability of each arc in $\pi_i^S \rightarrow x_i$ is specified, we can compute $P(\pi_i^S \rightarrow x_i)$. (Cooper and Herskovits, 1992, p.18)

Another early use of this prior is given by Buntine (1991), who gives an explicit formula for the probability of a structure Π in terms of individual arc probabilities. Assume that the expert has given us an ordering $\prec$ and also for any y and x, where $y \prec x$, has given us $P(y \rightarrow x | \prec, E)$. E represents information from the expert. For any structure Π consistent with $\prec$, the first independence assumption gives us:

$$Pr(\Pi | \prec, E) = \prod_{x \in \mathcal{X}} Pr(\Pi_x | \prec, E)$$

where Π_x is the family structure for node x, and the second independence assumption gives us:

$$Pr(\Pi_x | \prec, E) = \left(\prod_{y \in \Pi_x} Pr(y \rightarrow x | \prec, E) \right) \left(\prod_{y \notin \Pi_x} (1 - Pr(y \rightarrow x | \prec, E)) \right)$$

The possibility of computing exact posterior quantities using modular priors make them an attractive option. When the true ordering is not known modular priors can still be exploited by summing/maximising over orders (Koivisto and Sood, 2004) or sampling orders using MCMC (Friedman and Koller, 2003).

Note that the total order $\prec$ is crucial for the simple product form of this prior. Without it we have to consider combinations of parent-child links which give rise to directed cycles and hence illegal Bayesian networks. To see this consider the approach of Valdueza Castelo and Siebes (1998). In this approach the user is not required to supply a total ordering $\prec$. For any two distinct nodes A and B, the user is allowed (but not required) to define $P(A \rightarrow B)$, $P(A \leftarrow B)$ and $P(A \dots B)$. $P(A \dots B)$ denotes the probability that there is no arc. Clearly we must have

$$P(A \rightarrow B) + P(A \leftarrow B) + P(A \dots B) = 1$$

Valdueza Castelo and Siebes (1998) assume that the user's beliefs are coherent so that probabilities given by the user do not contradict this equation. If, for any two distinct nodes A and B, the user provide fewer than two probabilities, then the unspecified probabilities are set automatically by uniformly distributing the remaining probability mass. For example, if the user declares only that $P(A \rightarrow B) = 3/4$, then the system sets $P(A \leftarrow B) = P(A \dots B) = 1/8$.

Given a BN structure B_S, let $(v_i \leftrightarrow v_j)_{B_S}$ be either $v_i \rightarrow v_j$ or $v_i \leftarrow v_j$ or $v_i \dots v_j$ according to which of these three possibilities is specified by B_S. If we were to set

$$P(B_S|\xi) = \prod_{\substack{v_i, v_j \in V \\ i \neq j}} p((v_i \leftrightarrow v_j)_{B_S}|\xi)$$

then we would not have defined a probability distribution over BN structures because $\sum_{B_S} P(B_S|\xi) < 1$. The problem is that this simple product distribution assigns positive probability to directed graphs *with cycles*, so some of the mass for the BNs is lost.

The solution is simple. The product distribution is altered so that graphs with directed cycles are set to probability zero, and to compensate a normalising constant is introduced. Valdueza Castelo and Siebes (1998) consider two options for normalisation. The distribution over BNs can either be:

$$P(B_S|\xi) = c \prod_{\substack{v_i, v_j \in V \\ i \neq j}} p((v_i \leftrightarrow v_j)_{B_S}|\xi) \tag{1}$$

or

$$P(B_S|\xi) = c + \prod_{\substack{v_i, v_j \in V \\ i \neq j}} p((v_i \leftrightarrow v_j)_{B_S}|\xi) \tag{2}$$

Although Valdueza Castelo and Siebes (1998) do not describe it as such, Equation 1 defines a log-linear (also exponential-family or MAXENT) distribution over BNs, where the links are the 'features' of the BN and c is the dreaded partition function Z. In general, c will be hard to calculate, but as Valdueza Castelo and Siebes (1998) note we generally do not need to, since it is often enough to have probabilities defined only up to a normalising constant.

The approach of Valdueza Castelo and Siebes (1998) turns out to be essentially the same as one of those suggested by Madigan and Raftery (1994). In their experiments, Madigan and Raftery (1994) initially use a uniform prior, but this is criticised as follows:

> In the examples considered above, the prior model probabilities $pr(M)$ were assumed equal (Cooper and Herskovits, 1992, also assume that models are equally likely *a priori*). In general this can be unrealistic and may also be expensive and we will want to penalise the search strategy as it moves further away from the model(s) provided by the expert(s)/data analyst(s). Ideally one would elicit prior probabilities for all possible qualitative structures from the expert but this will be feasible only in trivial cases.

They go on to discuss an alternative to the uniform prior:

> For models with fewer than 15 to 20 nodes, prior model probabilities may be approximated by eliciting prior probabilities for the presence of every possible link and assuming that the links are mutually independent, as follows. Let $\mathcal{E} = \mathcal{E}_P \cup \mathcal{E}_A$ denote the set of all possible links for the nodes of model M , where $\mathcal{E}_P$ denotes the set of links which are present in model M and $\mathcal{E}_A$ denotes the absent links. For every link $e \in \mathcal{E}$ we elicit $\mathrm{pr}(e)$, the prior probability that link e is included in M. The prior model probability is then approximated by
>
> $$\mathrm{pr}(M) \propto \prod_{e \in \mathcal{E}_P} \mathrm{pr}(e) \prod_{e \in \mathcal{E}_A} (1 - \mathrm{pr}(e))$$
>
> Prior link probabilities from multiple experts are treated as independent sources of information and are simply multiplied together to give pooled prior model probabilities. Clearly, the contribution from each expert/data analyst could be weighted. (Madigan and Raftery, 1994)

The simplicity with which this prior incorporates information from multiple sources is a strong point in its favour, however the independence assumption seems unrealistic. Richardson and Domingos (2003) have recently used a similar method.

2.2.2 A distance based approach

Eliciting very many arc probabilities is unrealistic, so Madigan and Raftery (1994) go on to consider a coarser approach:

> For applications involving a larger number of nodes or where the elicitation of link probabilities is not possible, we could assume that the "evidence" in favour of each link included by the expert(s)/data analyst(s) in the elicited qualitative structure(s) is "substantial" or "strong" but not "very strong" or "decisive" (Jeffreys, 1961). For example, we could assume that the evidence in favour of an included link lies at the center of Occam's window corresponding to a prior link probability for all $e \in \mathcal{E}_P$ of
>
> $$\mathrm{pr}(e) = \frac{1}{1 + \exp\left(\frac{O_L + O_R}{2}\right)}$$

Similarly, the prior link probabilities for $e \in \mathcal{E}_A$ are given by

$$\mathrm{pr}(e) = \frac{\exp\left(\frac{O_L+O_R}{2}\right)}{1+\exp\left(\frac{O_L+O_R}{2}\right)}$$

(Madigan and Raftery, 1994)

This approach is most easily explained if we assume the domain expert has provided us with a single model which s/he proposes as his/her 'best guess'. Basically, each arc in this elicited model receives the same prior probability, and this probability will be higher than that for arcs not in the elicited model. The parameters O_L and O_R are set by the user, and affect the search algorithms proposed in Madigan and Raftery (1994). Their effect on the prior is to determine the prior bias in favour of arcs included in the model provided by the user.

A similar prior is used by Heckerman et al. (1995a) who present a Bayesian approach where "a user specifies his prior knowledge about the problem by constructing a Bayesian network, called a *prior network*, and by assessing his confidence in this network." They argue that "...structures that closely resemble the prior network will tend to have higher prior probabilities." If a given structure B_S differs from the prior network by δ arcs, then

$$P(B_S) = c\kappa^{\delta}$$

where κ $(0 < \kappa < 1)$ is a user-selected penalty factor and c is a normalising constant.

An alternative to penalising structures too distant from the expert's is simply to fix on some particular prior network irrespective of expert opinion. For example, Friedman and Koller (2003) consider (but do not use) a prior which penalises dense networks. In this case, implicitly, the arcless BN is the prior network. Suppose β is the probability for an edge to be present, then a structure with m edges will have prior probability proportional to $\beta^m(1-\beta)^{C(n,2)-m}$.

2.3 Priors over variable orderings

We have seen that if the user is prepared to supply a total order $\prec$ then setting an arc-based prior is simplified (as is learning in general). If the user is unwilling to do this, he/she might at least provide a prior over possible orderings. Although Madigan and Raftery (1994) suggest that "In the directed case it may be possible to construct a prior distribution on the space of orderings ...", this possibility is only really explored by Friedman and Koller (2003). However, this work concentrates on a MCMC method for sampling from the posterior distribution over orderings, rather than exploring methods for defining priors over that space. Consequently Friedman and Koller (2003) restrict their work to using a uniform distribution over orderings. Such a distribution is not *hypothesis equivalent* (Heckerman et al., 1995a): different structures in the same Markov equivalence class have different priors. Concerning this problem Friedman and Koller (2003) state that:

> In general, while this discrepancy in priors is unfortunate, it is important to see it in proportion. The standard priors over network structures are often used not

> because they are particularly well-motivated, but rather because they are simple and easy to work with. In fact, the ubiquitous uniform prior over structures is far from uniform over PDAGs (Markov equivalence classes)—PDAGs consistent with more structures have a higher induced prior probability.

This raises the question of the importance of *hypothesis equivalence* when setting priors. When the goal of BN learning is to induce a standard probabilistic model, then BNs in the same Markov equivalence class (i.e. defining the same set of probability distributions) are interchangeable and so it might seem inconsistent if they do not all have the same prior. However, if we consider working in a model space of Markov equivalence classes (whether represented by PDAGs or not) there is no inconsistency: the prior probability of a Markov equivalence class of BNs can be defined to be the sum of the priors of the BNs in that class. There seems no reason why these BN priors need be equal. Of course, it may be more convenient to define a prior directly on the Markov equivalence class: this is a knowledge engineering issue. In the case where BNs model more than probability distributions, for example where the arcs also have a causal interpretation then, of course, there is no argument for hypothesis equivalence.

2.4 Defining priors using global features

An application where the proper incorporation of prior structural knowledge is particularly important is the use of Bayesian networks to represent *pedigrees* (Sheehan and Sorensen, 2003). Pedigrees are directed graphs representing family relationships between individual people (or other animals). Given DNA data on a group of individuals it is often desired to uncover the pedigree relating them: settling paternity cases is a common motivation. Given DNA data x and a pedigree g, a likelihood $P(x|g)$ can be determined using the Mendelian laws of genetic inheritance. A Bayesian approach is possible by defining a prior distribution $P(g)$ in addition. If the number of possible pedigrees is not too great it is possible to apply an exact Bayesian approach to this, where the posterior probability of each pedigree is calculated. This is the approach taken in the (freely available) familias system: `http://www.math.chalmers.se/~mostad/familias/` (Egeland et al., 2000). There is a variety of BN representations of pedigrees in the literature. Here we will have each random variable in the BN represent the *genotype* of some individual. (A person's genotype is the entire set of genes under consideration for that person.)

For pedigrees, the terms, 'parent' and 'child' are not metaphors; a link $A \rightarrow B$ in a pedigree states that the person A is a known parent of the person B. Nodes in a pedigree have zero, one or two parents, corresponding to the number of known parents of the individual corresponding to that node. If the pedigree is represented as a Bayesian network this becomes a BN learning task. A Bayesian approach requires that a prior over each possible pedigree is specified. In the `familias` system (Egeland et al., 2000) each pedigree g has a prior proportional to

$$M_I^{b_I(g)} M_P^{b_P(g)} M_G^{b_G(g)} \tag{3}$$

where: M_I, M_P and M_G are non-negative parameters set by the user, $b_I(g)$ quantifies the amount of *inbreeding* present in g, $b_P(g)$ quantifies the level of *promiscuity* in g and $b_G(g)$ is the number of *generations* in g. Hard constraints can also be included by specifying

any known family relationships. Once pedigrees not satisfying the hard constraints are excluded, a uniform distribution over the survivors can be imposed by setting $M_I = M_P = M_G = 1$. However, doing so tends to give unrealistically high prior probabilities to pedigrees displaying high levels of inbreeding and promiscuity.

2.5 Setting parameter priors

Up to this point, only probabilities on Bayesian net structures—not parameters—have been considered. But it is not possible to consider only structure, since a BN structure on its own does not define a likelihood for the data. It follows that for each BN structure, a prior density over possible parameters for that structure is required. Methods for so doing will not be explored here, since our focus is on structural priors and the issues have been well covered elsewhere (Heckerman et al., 1995b). In short, there are compelling reasons to use Dirichlet priors, not least because, with complete data, it is possible to integrate the parameters out to get a closed form for the marginal likelihood. Prior Dirichlet parameters should be chosen so that BNs in the same Markov equivalence class have the same marginal likelihood. Using the freely available `deal` system (which is an R package) `http://www.math.aau.dk/novo/deal/` (Bøttcher and Dethlefsen, 2003) is a good way to understand how to set the Dirichlet priors.

3. Defining informative priors using stochastic logic programs

We use *stochastic logic programs* (SLPs) to define informative priors on BN structures. Basically, an SLP is a logic program with added probabilities. These probabilities are used to change the normal Prolog depth-first backtracking search to a probabilistic depth-first search, which in this paper will use backtracking. Stochastic logic programs were introduced by Muggleton (1996). Maximum likelihood parameter estimation for SLPs was done by Cussens (2001) and the first use of SLPs to define prior distributions was by Angelopoulos and Cussens (2001). Earlier work on SLPs, such as (Cussens, 2001), did not permit backtracking as we do here; backtracking for SLPs was first introduced by Cussens (2000).

In this section, the aim is to give a concise account of the basic method. Throughout this section, deliberately simplistic examples of SLPs and fragments of SLPs will be used for explanatory purposes.

3.1 Defining the model space with a logic program

Before considering how to define an informative prior over a space of models using an SLP, we will consider the easier—and related—problem of defining a model space with a logic program. Suppose that we wish to consider a space of possible BNs (denoted by Ω) containing a mere 5 Bayesian nets, as listed in Fig 1. To use a logic program to represent this space the first decision is to choose a representation of each member of Ω as a *first-order term*. (In all applications that have been studied so far, it has been natural to use *ground* first-order terms to represent statistical models, although there is no actual requirement for groundedness.) One option is to represent BNs as a list of families. Each family specifies a child together with a list of its parents. Next we choose a monadic predicate symbol

to denote the set of models so represented. Suppose we used $k/1$ for this purpose. This produces the logic program in Fig 1. $k/1$ is such that $k(bn)$ is true iff bn represents an BN in Ω. Note that the functor $\leftarrow$ is an *operator* so that $b \leftarrow [a, c]$ is syntactic sugar for $' \leftarrow' (b, [a, c,])$.

$$\Omega = \left\{ \begin{array}{ccccc} A & \rightarrow & B & \rightarrow & C \\ A & \rightarrow & B & \leftarrow & C \\ A & \rightarrow & B & & C \\ A & \leftarrow & B & \leftarrow & C \\ A & & B & & C \end{array} \right\}$$

```
:- op( 100, xfx, '<-' ).

k([a<-[],b<-[a],c<-[b]]).
k([a<-[],b<-[a,c],c<-[]]).
k([a<-[],b<-[a],c<-[]]).
k([a<-[b],b<-[c],c<-[]]).
k([a<-[],b<-[],c<-[]]).
```

Figure 1: A very small model space and the logic program defining this space.

Naturally, much bigger model spaces can be defined using this technique. The logic program in Fig 2 uses a *binary* predicate $k/2$ to define the set of all BNs consistent with some given ordering of the BN random variables. The first argument contains the ordering and the second the BN. The program in Fig 2 represents BNs as a list of edges, rather than a list of families—this is just more convenient for this program.[1]

```
:- op( 100, xfx, '<-' ).

k([],[]).
k([RV|RVs], BN) :-                      %it's a Bayes net if ..
        k(RVs, TailBN),                 %this is and ..
        edges(RVs, RV, TailBN, BN).     %we connect RV thus giving BN

edges([], _RV, BN, BN).                 % Finished.
edges([H|T], RV, TailBN, BN) :-         % Connect or otherwise, each
   connected(H, RV, TailBN, MidBN),     % potential parent.
   edges(T, RV, MidBN, BN).

connected(Pa, RV, TailBN, [Pa<-RV|TailBN]). % connect
connected(_NoPa, RV, BN, BN).               % don't connect
```

Figure 2: A logic program defining a model space containing all BNs consistent with some given variable ordering.

Fig 3 shows what happens when we use a normal Prolog interpreter to enumerate all BNs consistent with the lexicographic ordering for random variables `a`, `b` and `c`. Note that $k/2$ defines not one model space but an infinite number: one for each possible ordering

1. Some variables in Fig 2 begin with an underscore. This is just a Prolog convention for variables that appear only once in a clause.

provided by an instantiation of the first argument. It turns out to be very convenient to have such parameterised definitions of model spaces.

```
| ?- k([a,b,c],BN).
BN = [c<-a,b<-a,c<-b] ? ;
BN = [b<-a,c<-b] ? ;
BN = [c<-a,c<-b] ? ;
BN = [c<-b] ? ;
BN = [c<-a,b<-a] ? ;
BN = [b<-a] ? ;
BN = [c<-a] ? ;
BN = [] ? ;
no
```

Figure 3: Enumerating 3 node BNs consistent with lexicographic ordering, using the predicate $k/2$ defined in Fig 2. The semi-colons are user input.

Next, consider the logic program in Fig 4. This is similar to the program in Fig 2 except that BNs do not have to respect the ordering of the variables provided in the first argument. Because of this it is necessary to check that any graphs produced are acyclic. The code for doing this in Fig 4 is chosen for simplicity and is not the most efficient way of checking for cycles. The cut (!) in Fig 4 is a 'green cut': removing it to make the logic program pure would only affect the program's efficiency. The symbol `\+` is Prolog notation for negation, which in logic programs is negation-as-failure not standard first-order negation.

Logic programs (being a subset of first-order logic) provide a convenient high-level and declarative language with which to precisely define complex model spaces. For example, if we wish to restrict attention to only those BNs which respect some conditional independence relation it is enough to describe the condition in first-order logic and add it to the definition of what constitutes a model.

3.2 Representing proofs with proof trees

Consider the behaviour exhibited in Fig 3. An initial query `k([a,b,c],BN)` is posed to the Prolog interpreter: this amounts to the question: *does there exist a BN such that `k([a,b,c],BN)` is true?*. The interpreter (1) finds a proof (not shown in Fig 3) that `k([a,b,c],BN)` is true, (2) gives an affirmative answer and (3) provides a *witness* `BN=[c<-a,b<-a,c<-b]` corresponding to the found proof. In Fig 3 the user has asked for further proofs by entering '`;`' and the interpreter finds 8 proofs in total, providing a witness BN in each case. As will be explained in Section 3.3, SLPs define a distribution over first-order terms such as `BN=[c<-a,b<-a,c<-b]` by defining a distribution over the *proofs* of queries such as `k([a,b,c],BN)`.

Since proofs play such a central rôle in SLP-defined distributions, we need an appropriate representation of them. It will prove useful to represent each proof as a *proof tree*. Proof trees are fully described, for example, by Nilsson and Małuszyński (1995). Basically, a proof

```
k([],[]).
k([RV|RVs],BN) :-              % it's a Bayes net if ..
 k(RVs,TailBN),                % this is and ..
 edges(RVs,RV,TailBN,BN),      % we connect RV thus giving ..
 no_cycle(BN).                 % a BN

edges([],_RV,BN,BN).           % finished
edges([H|T],RV,TailBN,BN) :-  edges(T,RV,[H<-RV|TailBN],BN).
edges([H|T],RV,TailBN,BN) :-  edges(T,RV,[RV<-H|TailBN],BN).
edges([_H|T],RV,TailBN,BN) :- edges(T,RV,TailBN,BN).

no_cycle([]).
no_cycle(BN) :-
        select(_X<-Y,BN,Rest), % delete a parent ..
        \+ select(Y<-_Z,BN,_), % which is an orphan
        !, no_cycle(Rest).

select(H,[H|T],T).
select(X,[H|T],[H|T2]) :-  select(X,T,T2).
```

Figure 4: A logic program defining a model space containing all BNs for a given set of variables.

tree is the logic programming version of a parse (or derivation) tree for grammars. Proof trees are constructed by bolting together *elementary trees*. Each clause in the logic program has a corresponding elementary tree. Figs 5 and 6 show two clauses from Fig 4 together with their corresponding elementary trees. (We have abbreviated some of the predicate names in Fig 6 for reasons of space.)

Figure 5: A unit clause with its elementary tree representation.

```
k([RV|RVs],BN) :-
 k(RVs,TBN),
 e(RVs,RV,TBN,BN),
 nc(BN).
```

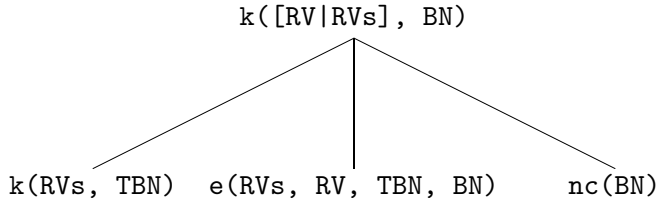

Figure 6: A non-unit clause with its elementary tree representation.

Atomic *goals* (i.e. things we want to prove such as `k([a,b,c],BN)`) also have an elementary tree representation: each goal corresponds to a tree where the goal is the only node, such as in Fig 7. Elementary trees are joined together to make *derivation trees* by connecting the root of one elementary tree onto the leaf of another and declaring an equality constraint between the atomic formulae which label these two nodes. Only leaves labelled with atomic formulae, not those labelled ■, can have trees joined to them.

Figure 7: A single-node elementary tree corresponding to the goal `k([a,b,c],BN)`.

For example, Fig 8 shows the derivation tree produced by joining the elementary trees of Figs 6 and 7. Note that variables are renamed ('standardised apart') to prevent accidental equalities. Derivation trees with a goal at the root node represent states in the search for a proof of that goal. The tree in Fig 8 represents the state that is reached after choosing the rule in Fig 6 as the first rule in the search for a proof for `k([a, b, c], BN)`.

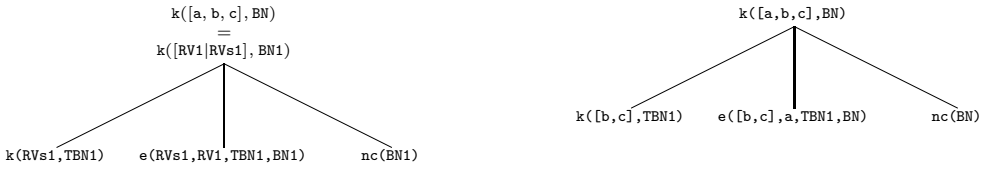

Figure 8: LHS: Derivation tree produced by combining the elementary trees of Figs 6 and 7. Note the variable renaming. RHS: Simplified version of this tree.

A derivation tree all of whose leaves are labelled by ■ is said to be *complete*. Complete derivation trees are also called *proof trees*. Proof trees cannot be further extended, so they represent the termination of a proof search. If the set of equations in a derivation tree have a solution, then the tree is *consistent*, otherwise it is *inconsistent*. Only consistent proof trees represent successful proofs; inconsistent ones represent 'failed proofs'.

One strategy to finding proofs is to build a complete proof tree, and only then check whether the equations it contains are soluble. Clearly this can be highly inefficient: it is much better to check for solubility as the tree is built, stopping the construction of the tree if the current set of equations is insoluble. The Prolog approach to doing this is to simplify the set of equations after each extension of the derivation tree, thus facilitating the detection of an insoluble set of equations. The RHS of Fig 8 shows how simplification works. Note that the extension of the initial goal tree in Fig 7 to the derivation tree in Fig 8 amounts to replacing one atomic goal with the three atomic goals at the leaves of Fig 8. Each of these atomic goals has to be proved in the same way as the original goal: by having a consistent proof tree built 'underneath' it. However, because the variables `BN` and `TBN1` are shared between the three goals there is a dependency between possible proof trees for each of the three goals.

Consider next negated goals such as `\+ select(Y<-_Z,BN,_)` in Fig 4. A consistent proof tree with a negated goal `\+ Goal` exists if and only if it can be established that there is *no* consistent proof tree with `Goal` as its root. In this case the proof tree has only two

nodes: the root `\+ select(Y<-_Z,BN,_)` and a single leaf ■. Note that no variables in a negated goal get instantiated.

It may be useful for some readers to compare proof trees to the more standard account of proof construction for logic programs in terms of SLD-resolution. In SLD-resolution to prove a formula such as $k([a,b,c],BN)$, the negation $\neg k([a,b,c],BN)$ is asserted (the initial goal) and an attempt to prove a contradiction is made. Standard SLD-resolution selects the leftmost atom from the current goal, unifies it with the head of a clause and replaces the selected atom with the body of that clause. It at any point the empty goal is produced then a contradiction has been established. The connection with proof trees is simple: replacing the selected atom with a clause body is like joining an elementary tree; the current goal is just the set of non-■ leaves; the selected atom of the current goal will be the deepest, leftmost leaf; a complete tree (all leaves ■) corresponds to producing the empty goal.

Prolog execution (at least conceptually) consists of interleaving steps of tree extension and equation simplification. As previously mentioned, in standard Prolog, the tree is always extended by attaching an elementary tree to the deepest, leftmost non-■ leaf. Call the atomic formula at this leaf the *selected atom*. Generally, there will be several elementary trees available. This creates what is known as a *choice-point* which is simply an ordered set of possible elementary trees. In standard Prolog the ordering of elementary trees is fixed by the lexical ordering of the corresponding clause in the Prolog source file.

Upon the creation of a choice-point the first tree in the ordering is used and then removed from the choice-point. If at any stage an inconsistent derivation tree is produced, then the process backtracks to the most recent non-empty choice point and uses the first elementary tree in the choice-point (this tree is then removed from the choice-point). The process of backtracking to non-empty choice points and removing trees from choice-points means it is possible to backtrack all the way back to the first choice-point created.

The trees in Figs 7 and 8 show the first two stages in the construction of a proof tree for the goal `k([a, b, c], BN)`. For reasons of space we will not show all the subsequent stages, but one intermediate stage is shown in Fig 9 and the final stage is the consistent proof tree shown in Fig 10. This last tree constitutes a proof that `k([a,b,c], [c<-a,b<-a,c<-b])` is true. We say that the consistent proof tree has *yielded* the atom `k([a,b,c], [c<-a,b<-a,c<-b])`. Yielded atoms are initial goals with the instantiation generated by the tree applied. The SLPs considered in this paper only every produce ground yielded atoms.

3.3 Defining priors using stochastic logic programs

Having seen how logic programs are used to define model spaces, we now turn to how *stochastic* logic programs (SLPs) can be used to define probability distributions over these model spaces. The basic idea is very simple: parameters are added to a logic program so that when a choice-point is created an elementary tree is chosen according to a probability distribution determined by the parameters, rather than deterministically as in standard Prolog execution. This determines a distribution over consistent proof trees for any given initial goal and thus a distribution over instantiations of any variables in that initial goal.

There are number of different ways of doing this, three of which were discussed by Cussens (2000). In this paper, we opt for an approach named *backtrackable sampling*, which

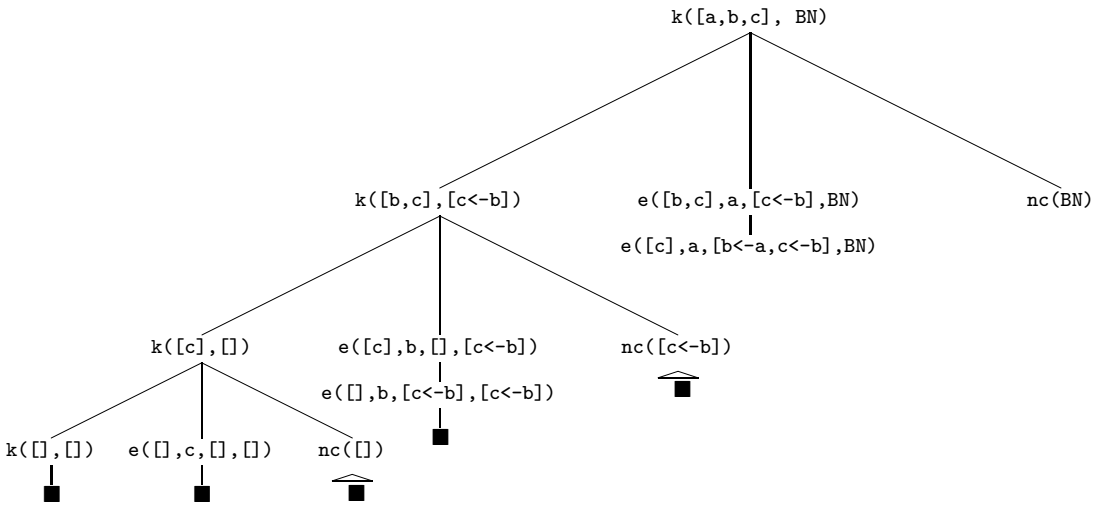

Figure 9: The lexically first recursive $e/4$ clause is used, `a` is chosen to be a parent of `b`. The selected atom is now `e([c],a,[b<-a,c<-b],BN)`. A choice point is created since there are alternative elementary trees (= clauses) for this atom.

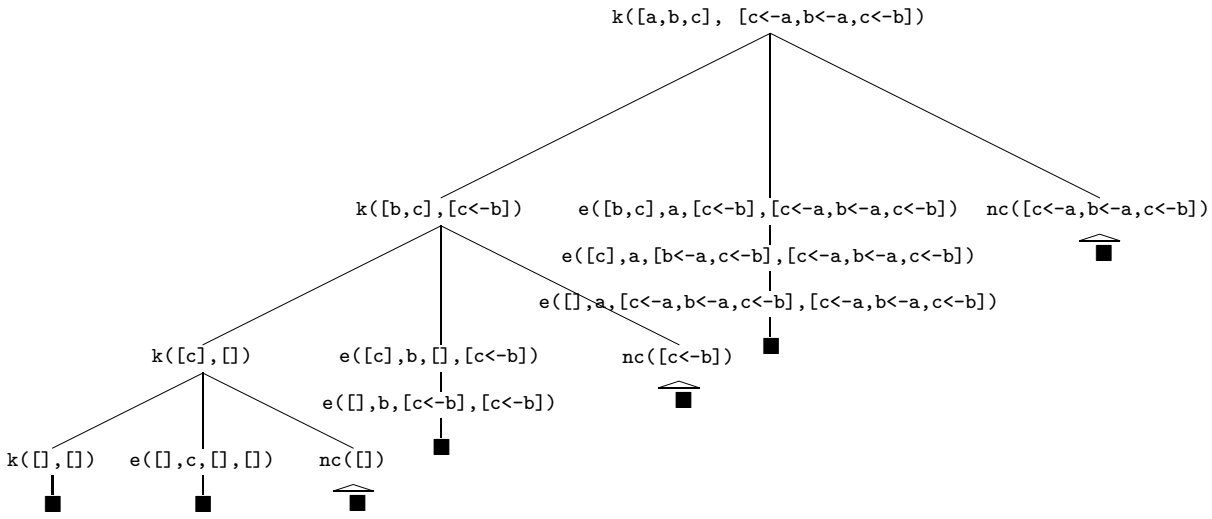

Figure 10: At this point `nc([c<-a,b<-a,c<-b])` is proved to be true, but the relevant proof sub-tree has been omitted. This is a consistent proof tree proving that `k([a,b,c], [c<-a,b<-a,c<-b])` is true.

is similar in spirit to that given in the original paper by Muggleton (1996). Backtrackable sampling was introduced by Cussens (2000) where it is contrasted with using SLPs to define log-linear distributions. It is important to realise that backtrackable sampling *does not* define a log-linear model, in contrast with the sampling approach analysed by Cussens (2001) which did not permit backtracking.

Backtrackable sampling permits us to use a very generalised form of SLP. Not all clauses need to be parameterised, so the SLPs are *impure*, and any parameters need only to be non-negative, so that the SLPs are *unnormalised*. We will begin our presentation with *unextended* SLPs leading to the simple formal definition in Definition 1.

Definition 1 *An* unextended, impure, unnormalised stochastic logic program *is a logic program where some clauses are annotated with non-negative numbers. A clause C annotated with p is represented by $p :: C$. For each predicate in such an SLP, either all the clauses making up its definition are parameterised or none are. Predicates of the former sort are called* probabilistic predicates*.*

It remains to state how these clause parameters determine a distribution over consistent proof trees for any given initial goal. The distribution is determined by a sampling mechanism for consistent proof trees which we now define.

Definition 2 Backtrackable sampling *for unextended, impure, unnormalised stochastic logic programs produces a distribution over consistent proof trees for an initial goal as follows. Backtrackable sampling is identical to the standard Prolog construction of a proof tree except that elementary trees are chosen* probabilistically *from non-empty choice points when the predicate symbol of the selected atom is probabilistic. If $T_1, T_2, \ldots T_n$ is such a choice point and $p_1 :: C_1, p_2 :: C_2, \ldots p_n :: C_n$ is the corresponding set of annotated clauses, then tree T_i is selected with probability $p_i / \sum_{j=1}^{n} p_j$. (Note that these probabilities are independent of the ordering of the elementary trees.)*

Definition 3 *A distribution over consistent proof trees P_{tree} (for a given initial goal G_0) defines a distribution P_{atom} over yielded atoms $G_0\theta$ as follows. For any atom $G_0\theta$, $P_{\text{atom}}(G_0\theta) = \sum_{\{t:t \text{ yields } G_0\theta\}} P_{\text{tree}}(t)$*

It is because selection probabilities are defined by the quotient $p_i / \sum_{j=1}^{n} p_j$ that there is no need for the parameters for a particular predicate to be *normalised*, i.e. to sum to one. However, in our examples we will use normalised SLPs since there is no reason not to. Note that the set of possible trees at a choice point is reduced if backtracking back to that choice point occurs. This is because the 'guilty' elementary tree (= clause) is no longer a possible choice. This means that the denominator in $p_i / \sum_{j=1}^{n} p_j$ will be reduced. If only one tree is left at a choice point then it is chosen with probability one.

For a given SLP and initial goal, it is possible to characterise backtrackable sampling for SLPs as a Markov chain. Each state of the Markov chain is a derivation tree together with the current choice point for every node in the tree. We will call these states *augmented derivation trees*. Augmented derivation trees are nothing more than representations of the internal state of a Prolog program as it executes. The initial state (with probability one) is the one-node derivation tree whose single node is the initial goal with a choice

point containing all elementary trees whose root node matches the predicate symbol of the initial goal. If the selected atom of the current augmented derivation tree (= state) is non-probabilistic then with probability 1 it is extended with the next available elementary tree, if any, according to the standard Prolog lexical ordering. If the selected atom is probabilistic, then for the unextended SLPs we are currently considering, the selected elementary tree is chosen as specified in Definition 2. If an inconsistent tree is produced or if we run out of elementary trees for a particular choice-point then with probability 1 the chain backtracks to the most recent non-empty choice-point as previously described. We can use the usual trick of making 'final' states, which here are consistent proof trees, into *absorbing* states. Once we hit a consistent proof tree the chain remains stuck there for ever. For non-trivial SLPs the state space of this Markov chain is very large, but since both it and the relevant transition probabilities are defined in a compact manner it is still a usable representation.

If *any* sequence of probabilistic choices leads to a consistent proof tree *without backtracking*, then the SLP is said to be *failure-free*. Our first example SLP, in Fig 11 is clearly failure-free; it is a simple tabular representation of a discrete probability distribution for the model space defined by its underlying logic program (which is given in Fig 1).

```
0.40 :: k([a<-[],b<-[a],c<-[b]]).
0.15 :: k([a<-[],b<-[a,c],c<-[]]).
0.15 :: k([a<-[],b<-[a],c<-[]]).
0.15 :: k([a<-[b],b<-[c],c<-[]]).
0.15 :: k([a<-[],b<-[],c<-[]]).
```

Figure 11: A failure-free SLP defining a prior distribution over a very small space of BNs.

Our next example SLP is produced by replacing the definition of `connected/4` in Fig 2 with:

```
0.6 :: connected(Pa, RV, TailBN, [Pa<-RV|TailBN]).     % connect
0.4 :: connected(_NoPa, RV, BN, BN).                   % don't connect
```

The SLP so produced is not formally failure-free, since a naïve underlying Prolog interpreter might, for example, select the first *edges*/4 clause when the first argument of the selected atom is a non-empty list. This would produce an inconsistent tree (a unification failure) and backtracking would be invoked to try (successfully) the other *edges*/4 clause. A smarter system would use indexing to avoid this. This is just a question of the efficiency of sampling, the same distribution over consistent proof trees is defined in both cases.

Failure-free SLP priors are efficient to sample from (since each clause choice constitutes progress towards a successful proof) and have a simple mathematical characterisation. They are essentially equivalent to stochastic context-free grammars with predicates playing the rôle of grammar non-terminals. For the unextended SLPs we have seen so far, the sequence of probabilistic choices made in building a consistent proof tree for a failure-free SLP is simply a joint instantiation of a sequence of *independent* random variables—each individual random variable having a multinomial distribution specifying which clause to choose for a probabilistic predicate.

We next consider a more complex SLP which is produced by replacing the definition of `edges/4` in Fig 4 with:

```
edges([],_RV,BN,BN).                    % finished
edges([H|T],RV,BNIn,BNOut) :-
        p_edges([H|T],RV,BNIn,BNOut).

0.3 :: p_edges([H|T],RV,TailBN,BN) :-  % RV parent of H
 edges(T,RV,[H<-RV|TailBN],BN).
0.2 :: p_edges([H|T],RV,TailBN,BN) :-  % RV child of H
 edges(T,RV,[RV<-H|TailBN],BN).
0.5 :: p_edges([_H|T],RV,TailBN,BN) :- % no direct connection
 edges(T,RV,TailBN,BN).
```

We have shown how for the logic program of Fig 4 the standard Prolog approach would build a consistent proof tree for the goal `k([a,b,c], BN)` in Figs 8–10. We now consider how backtrackable sampling works for its probabilistic version.

Note that our SLP has two predicates: *edges*/4 and *p_edges*/4 where its corresponding logic program in Fig 4 had only one: *edges*/4. This is to cleanly separate the entirely deterministic question of whether there remain nodes to consider connecting to the 'current' node `RV` from the probabilistic issue of what sort of connection to make. To cut down on the size of the derivation trees we are about to show we will informally pretend, when presenting these trees, that there is only one 'edges' predicate in our SLP, which, moreover, we will abbreviate to *e*/4.

Suppose that, as chance would have it, the lexically first recursive *p_edges*/4 clause had been probabilistically chosen for the first two choice points, just as if normal deterministic Prolog execution were being used. The derivation tree would then be the one shown in Fig 9. The selected atom at this point is `edges([c],a,[b<-a,c<-b],BN)`—abbreviated to `e([c],a,[b<-a,c<-b],BN)`—and a choice point is created since there is a choice of 3 *p_edges*/4 clauses. Suppose that the elementary tree corresponding to the *second* *p_edges*/4 clause is probabilistically chosen. This produces the derivation tree shown in Fig 12. The procedure would then (deterministically) use the base *edges*/4 clause to produce the tree in Fig 13.

The tree in Fig 13 is not inconsistent itself, but it has no consistent complete extension because `nc([a<-c,b<-a,c<-b])` is not true and thus cannot be proved. We will not detail the relevant proof steps since there is no probabilistic element. Suffice to say that once the unprovability of `nc([a<-c,b<-a,c<-b])` has been established, our sampling mechanism will backtrack to the most recent choice point which is the tree in Fig 9, but with the 'guilty' second recursive clause now removed from the choice point. Suppose that this time, the third *p_edges*/4 clause is probabilistically chosen, then the derivation tree in Fig 14 is the result. This tree will then be deterministically extended to the consistent proof tree in Fig 15 without any further backtracking.

3.4 Extended SLPs

It is not difficult to see that the state transition probabilities of the Markov chain associated with an unextended SLP depend only on the predicate symbol of the selected atom

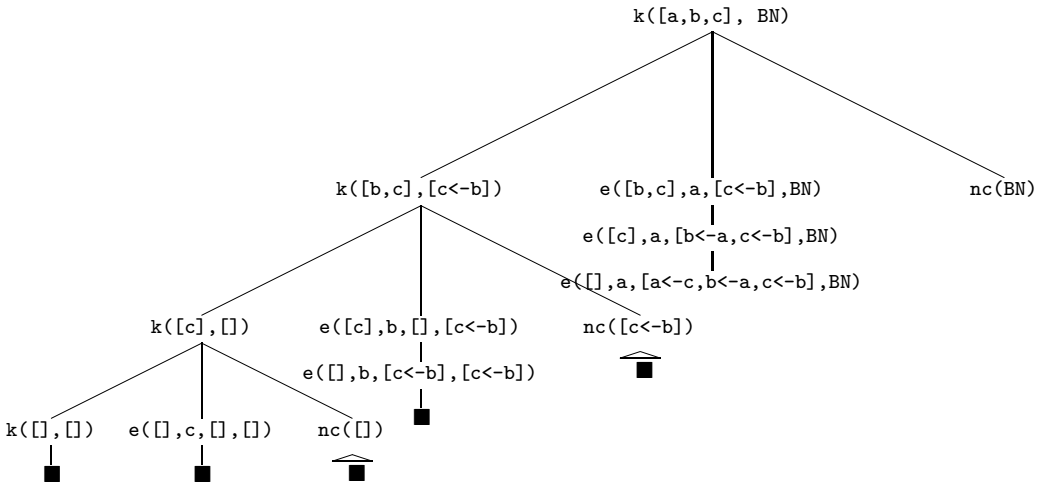

Figure 12: Probabilistically choose c to be the parent of a

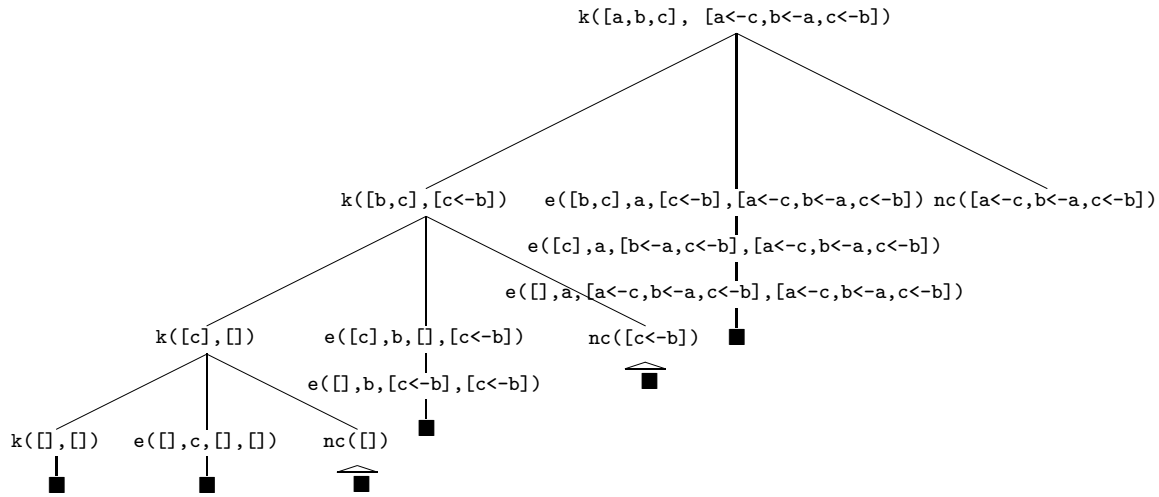

Figure 13: An incomplete derivation tree which cannot be extended to a *consistent* proof tree, since `nc([a<-c,b<-a,c<-b])` is not true. The backtrackable sampling mechanism will thus eventually backtrack to the most recent choice point, which is Fig 9.

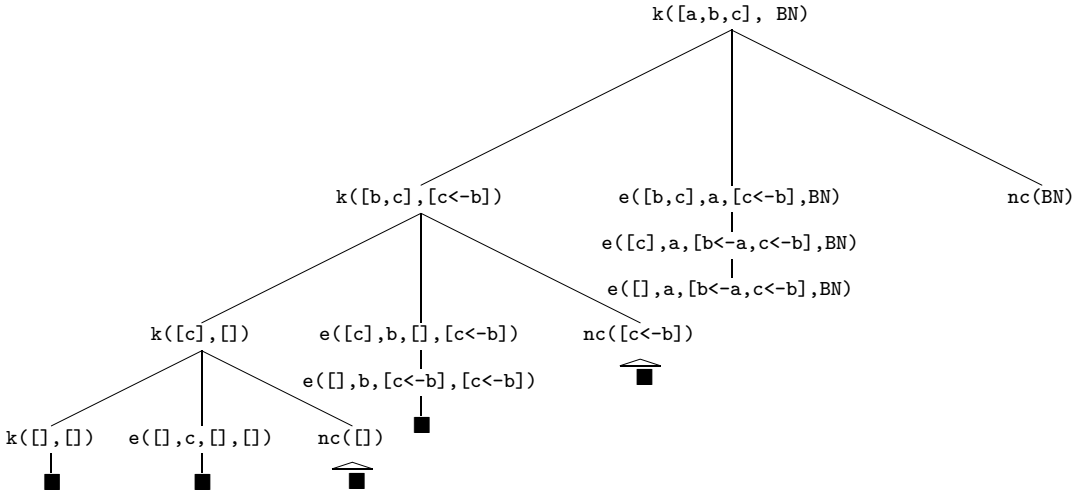

Figure 14: Probabilistically choose c not to connect to a

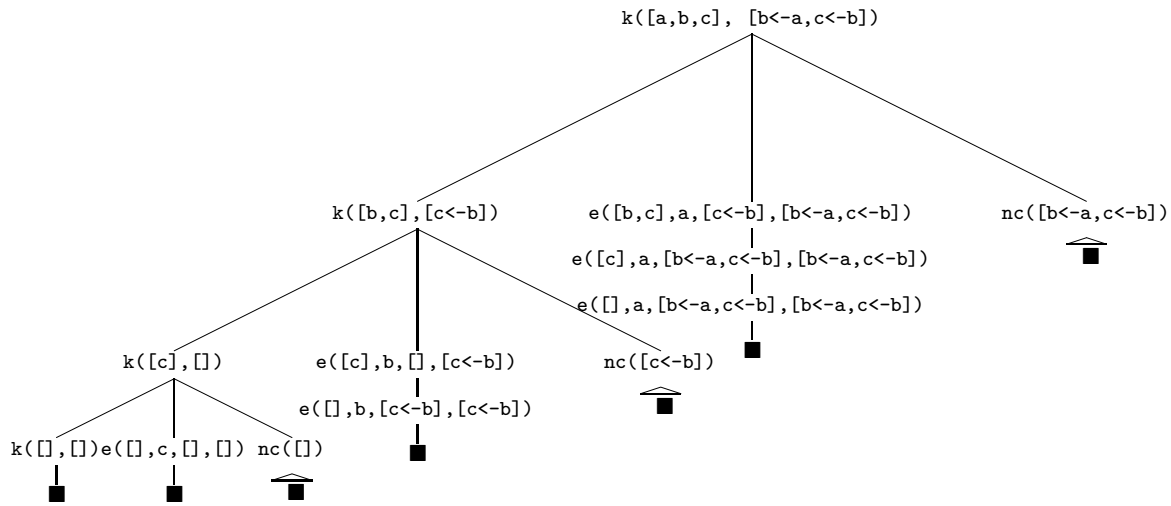

Figure 15: A consistent proof tree proving that `k([a,b,c], [b<-a,c<-b])`

and which clauses, if any, have been removed from the choice-point corresponding to this selected atom. Sometimes this is not fine-grained enough. *Extended SLPs* (Angelopoulos and Cussens, 2004b), in contrast, permit a much greater range of transition probabilities to be used. We introduce extended SLPs by way of a simple example in Fig 16.

```
:- pvars( umember(_El,List), [L-(length( List, L ),L>0)] ).

1/L :: [L] :: umember( El, [El|T] ).
1 - (1/L) :: [L] :: umember( El, [_|T] ) :-  [L-1] :: umember( El, T ).
```

Figure 16: An extended SLP for selecting uniformly from a list.

The basic idea behind `umember/2` in Fig 16 is that the probability with which each element (first argument) is picked from a list (second argument) should be uniform. To achieve this, when `umember(El,List)` is the selected atom the `pvars/2` directive is called and `length( List, L ),L>0` is used to compute the list length `L`. The value `L` is then used to dynamically compute the probabilities for the two `umember/2` clauses via the expressions `1/L` and `1 - (1/L)`. It is easy to see (by induction) that this procedure defines a uniform distribution over members of the list.

In general, the first argument for directive `pvars/2` is an atom corresponding to a probabilistic predicate while the second one is a list of pairs. Each member in the list connects a variable (e.g. `L`) to its *guard* (e.g. `(length( List, L ),L>0)`). The syntax for defining probabilistic clauses is extended to, `Expr::List::Clause`. `Expr` is an arithmetic expression containing some of the variables from `List`. `List` is a list of variables, each of which corresponds to one item in the second argument of the relevant `pvars/2` directive. `Clause` is a standard clause. Note that the recursive call in our example is `[L - 1]::umember( El, T )`. Since the length of tail `T` can be computed from that of `[H|T]`, recursive calls can avoid calling `length/2`. In the example, this is achieved by attaching the evaluable expression `L - 1` to the recursive call. In general a goal corresponding to a probabilistic call can be either of the form `Head` or of the form `Exprs::Head`. With `Exprs` being a list of evaluable expressions and variables. When the call is of the form `Head` or when a variable is found in `Exprs`, the relevant guards from `pvars/2` are called. Otherwise, the expressions in `Exprs` are evaluated.

The basic rule for using extended SLPs is that the SLP must be written in such a way that selected atoms contain sufficient information to enable computation of the desired probabilities. In this paper we will not attempt to characterise which probability distributions can be represented by extended SLPs, leaving this for future work. So far, we have only used extended SLPs for simple utility predicates, similar to `umember/2`.

4. MCMC for posterior distributions over large finite spaces

This section does not contain a tutorial on Markov chains or on MCMC. The bare minimum required to understand our particular approach and to fix notation is provided. Good introductions to finite Markov chains are provided by Feller (1950) and Häggström (2002).

For an introduction to MCMC from a machine learning perspective see Andrieu et al. (2003) and for a statistical perspective see Robert and Casella (2004).

4.1 MCMC basics

Let $\mathcal{X}$ be a finite set, and let π be some probability distribution over $\mathcal{X}$. Suppose we wish to sample from π but this is difficult to do directly. The Markov chain Monte Carlo (MCMC) solution to this problem is to construct a Markov chain to indirectly produce an approximate sample from π. Restricting to the homogenous case as we will do throughout, a Markov chain on $\mathcal{X}$ is defined by an initial distribution μ_0 on $\mathcal{X}$ and transition probabilities $P(x, y)$ for all $x, y \in \mathcal{X}$. Elements of $\mathcal{X}$ can be viewed as *states*: intuitively $P(x, y)$ is the probability of 'moving' from state x to state y. The Markov chain provides a distribution over infinite state sequences. Let X_t be the state at (discrete) time t, then the transition probabilities determine all the necessary conditional probabilities via this conditional independence relation:

$$P(X_t = y | X_0 = x_0, \ldots X_{t-1} = x) = P(X_t = y | X_{t-1} = x) = P(x, y) \qquad t = 1, 2, \ldots \tag{4}$$

Markov chains are thus memoryless: the probability of moving to any given state depends only on the current state. Let μ_t represent the (marginal) probability distribution on X_t defined by the Markov chain. Since $\mathcal{X}$ is finite the transition probabilities can be arranged in a transition matrix P such that entry P_{ij} of P is the probability of moving from state number i to state number j according to some arbitrary enumeration of the elements (=states) of $\mathcal{X}$. We then have $\mu_t = \mu_0 P^t$, where μ_0 and μ_t are row-vectors of length $|\mathcal{X}|$.

If π is the distribution from which we wish to sample, the MCMC approach is to construct a Markov chain such that $||\mu_t - \pi|| \to 0$ as $t \to \infty$. For sufficiently large t, samples drawn from X_t will provide a good approximation to sampling from π. This convergence implies that $\pi P = \pi$. π is thus a *stationary distribution* for the chain.

4.2 The Metropolis-Hastings algorithm

Throughout we will restrict attention to posterior distributions π, where each $x \in \mathcal{X}$ represents a particular statistical model in some model space. Note that in this paper, attention is restricted to Bayesian network models, but that much of what follows is of general application. We have that $\pi(x) \propto L(x)\pi_0(x)$ where $L(x)$ is the likelihood associated with x for some fixed set of observed data, and where $\pi_0(x)$ is the prior probability of x.

In this paper each BN $x \in \mathcal{X}$ is unparameterised: no conditional probabilities are specified. It follows that the likelihood $L(x)$ must be a *marginal* likelihood: we integrate over possible parameter values rather than specify particular values. This integration requires the definition of prior density functions and we adopt the entirely standard approach of requiring these to be Dirichlet distributions. For the details of this approach see Heckerman et al. (1995b). If various independence assumptions are made about the joint distribution over parameters we have, for any BN $x \in \mathcal{X}$:

$$L(x) = \prod_{i=1}^{n} \prod_{j=1}^{q_i} \frac{\Gamma(\alpha_{ij})}{\Gamma(n_{ij} + \alpha_{ij})} \prod_{k=1}^{r_i} \frac{\Gamma(n_{ijk} + \alpha_{ijk})}{\Gamma(\alpha_{ijk})} \tag{5}$$

where i is a node in the BN, j is a joint instantiation of the parents of such a node in x and k is a value of the node. n_{ijk} is the data count of node i having value k when its parents have configuration j. α_{ijk} is the corresponding Dirichlet parameter. Finally, $n_{ij} = \sum_{k=1}^{r_i} n_{ijk}$ and $\alpha_{ij} = \sum_{k=1}^{r_i} \alpha_{ijk}$. In our experiments we have set $\alpha_{ijk} = N/(r_i q_i), \forall i, j, k$ where N (the *prior precision*) has either been $N = 1$ or $N = 10$. For either value of N, the marginal likelihood function is an instance of the *BDeu metric* for scoring BN structures. This metric respects *likelihood equivalence*: BN structures encoding the same conditional independence relations (*Markov equivalent* structures) will get the same score. The BDeu metric was introduced by Buntine (1991) and was given this name by Heckerman et al. (1995b) who showed it was a special case of a more general likelihood-equivalent metric, the BDe metric.

The central assumption of our method is that, although π is hard to sample from, π_0, or distributions closely related to π_0, *are* easy to sample from. We do not assume that $\pi_0(x)$ is easy to evaluate for any given $x \in \mathcal{X}$, so our method is particularly well-suited to prior distributions which are directly defined in terms of some sampling mechanism, as opposed to some closed form.

We use the Metropolis-Hastings algorithm to construct a chain with π as stationary distribution. This algorithm defines a Markov chain using an auxiliary *proposal distribution* q as follows. If $X_t = x$, then propose a new state y according to the proposal distribution $q(y|x)$. This restricts q to be such that values can be readily sampled from the conditional distributions $q(.|x), x \in \mathcal{X}$. Next compute an *acceptance probability* $\alpha(x, y)$ as defined by (6).

$$\alpha(x, y) = \min\left\{\frac{\pi(y)}{\pi(x)}\frac{q(x|y)}{q(y|x)}, 1\right\} \tag{6}$$

and accept the proposed y with probability $\alpha(x, y)$. If the proposal is rejected then the chain remains at state x. Under weak conditions such a Markov chain converges to π.

We now consider proposals for which the acceptance probability is particularly easy to compute. Suppose the proposal q is related to the *prior* as follows:

$$\frac{q(x|y)}{q(y|x)} = R_q(x, y)\frac{\pi_0(x)}{\pi_0(y)} \Leftrightarrow R_q(x, y) = \frac{q(x|y)\pi_0(y)}{q(y|x)\pi_0(x)} \qquad x, y \in \mathcal{X} \tag{7}$$

where $R_q(x, y)$ is some function which is easy to evaluate for any $x, y \in \mathcal{X}$. Whenever (7) is satisfied we have:

$$\begin{aligned} \alpha(x, y) &= \min\left\{\frac{\pi(y)q(x|y)}{\pi(x)q(y|x)}, 1\right\} \\ &= \min\left\{\frac{\pi_0(y)L(y)q(x|y)}{\pi_0(x)L(x)q(y|x)}, 1\right\} \\ &= \min\left\{R_q(x, y)\frac{L(y)}{L(x)}, 1\right\} \end{aligned} \tag{8}$$

So if (7) holds then the acceptance probability reduces to a likelihood ratio multiplied by the easily-evaluable $R_q(x, y)$. In this case, we do not need to be able to evaluate any other functions, in particular we do not need to be able to evaluate the prior or the proposal distribution.

4.2.1 The single-component Metropolis-Hastings algorithm

The ordered single-component Metropolis-Hastings algorithm is a variant which is applicable when models $x \in \mathcal{X}$ can be divided into components $x = \{x_1, x_2, \ldots x_n\}$. The details are given by (Gilks et al., 1996, p. 10) from whom we extract a brief description. Let $x_{t.i}$ denote the state of x_i at time t. Iteration $t+1$ is done in n steps. Candidate $y_{.i}$ is proposed by $q_i(y_{.i}|x_{t.i}, x_{t,-i})$ where

$$x_{t,-i} = (x_{t+1,1}, \ldots, x_{t+1,i-1}, x_{t,i+1}, \ldots x_{t,n})$$

so the proposal is conditional on the already-found new components of x for earlier components and the existing later components. Each proposed $y_{.i}$ is accepted with probability:

$$\alpha_i = \min\left\{\frac{\pi(y_{.i}|x_{.-i})q_i(x_{.i}|y_{.i}, x_{.-i})}{\pi(x_{.i}|x_{.-i})q_i(y_{.i}|x_{.i}, x_{.-i})}, 1\right\}$$

Ordered Gibbs sampling is a special case where $q_i(y_{.i}|x_{.i}, x_{.-i}) = \pi(y_{.i}|x_{.-i})$, the proposal distributions are the full conditional posterior distributions, so that $\alpha_i \equiv 1$. If conditional posterior distributions are not available to serve as proposals, but conditional *prior* distributions are, we can set $q_i(y_{.i}|x_{.i}, x_{.-i}) = \pi_0(y_{.i}|x_{.-i})$ in which case:

$$\alpha_i = \min\left\{\frac{L(y_{.i}, x_{.-i})}{L(x_{.i}, x_{.-i})}, 1\right\} \tag{9}$$

Call this algorithm the ordered *prior-Gibbs* sampler. In our case, due to the decomposability of the marginal likelihood, the likelihood ratio in (9) reduces to $L_i(y_{.,i})/L_i(x_{.,i})$ where L_i is the component of the likelihood associated with variable i. This cheaply computable acceptance probability is a big advantage for prior-Gibbs sampling.

Note that the Gibbs (resp. prior-Gibbs) sampler requires proposal distributions only for the n full conditional posterior (resp. prior) distributions. It follows that the posterior (resp. prior) need only be specified implicitly by these conditional distributions. However in a number of applications it has been found practical to use an MCMC algorithm in all respects like the ordered Gibbs sampler except that the n distributions used for sampling are *inconsistent*: there is no joint distribution for which they are full conditional distributions. Following Heckerman et al. (2000) we will call such a procedure *ordered pseudo-Gibbs sampling*.

As Gelman points out:

> This "ordered pseudo-Gibbs sampler" (Heckerman, Chickering, Meek, Rounthwaite, and Kadie 2001) is a Markov chain algorithm, and if the values of the parameters are recorded at the end of each loop of iterations, then they will converge to a distribution. This distribution may be inconsistent with various of the conditional distributions used in its implementation, but in general there will be convergence to *something* (assuming that the usual conditions hold for convergence of a Markov chain to a unique nondegenerate stationary distribution). (Gelman, 2004) [emphasis in the original]

Gelman notes that the inconsistency is a "theoretical flaw" but argues that in compensation "Conditional modeling allows enormous flexibility in dealing with practical problems.

In applied work, we have never been able to fit the joint models to a real dataset without making drastic simplifications." One interesting example of using inconsistent conditional distributions is the PHASE algorithm for haplotype reconstruction whose authors describe it as follows:

> Indeed, the algorithm implemented in PHASE was not actually developed in the conventional way of writing down a prior and likelihood and then developing a computational method for sampling from the corresponding posterior (and neither, incidentally, was the algorithm of Lin et al. [2002]). Rather, the posterior is defined implicitly as the stationary distribution of a particular Markov chain, which in turn is defined via a set of (inconsistent) conditional distributions. (Stephens and Donelly, 2003)

Lauritzen and Richardson (2002) note that even if the pseudo-Gibbs sampler is guaranteed to have a stationary distribution, it may have "a limiting distribution depending on the particular choice of ordering π in the sitewise updating" (Stephens and Donelly use a randomised component updating scheme for PHASE to avoid this problem.) They add that "it would be desirable to have a more precise understanding of the general relation between the limiting distribution μ of a stationary pseudo-Gibbs sampler and the conditional specifications q."

In this paper we use *pseudo-prior-Gibbs sampling* which is like prior-Gibbs sampling except that the proposals may be inconsistent: there may be no prior distribution for which they are conditional distributions. Much of what has been said about pseudo-Gibbs sampling applies to pseudo-prior-Gibbs sampling: it is an approximation to the theoretically sound prior-Gibbs sampling which works well in practice. When the proposals closely approximate proper conditional priors then we can expect a reasonable approximation but we lack a precise understanding of the nature of this approximation.

4.3 Metropolis-Hastings with SLPs

In this paper we implement the Metropolis-Hastings algorithm using SLPs. The prior π_0 is expressed by an SLP. The state space $\mathcal{X}$ is the set of consistent proof trees constructed from the SLP and a given user-supplied goal. Examples of consistent proof trees can be found in Figs 10 and 15. The proof tree in Fig 15 was sampled from the SLP version of Fig 4 using the initial goal `k([a,b,c], BN)`. As described in Section 3, SLPs define a distribution over yielded atoms. Our SLP priors are written in such a way that there is exactly one term in each yielded atom which represents a model—in this paper the model is a BN. Call this term the *model term*. For example, the tree in Fig 15 generates the model term `[b<-a,c<-b]` which represents the obvious BN structure. Each $x \in \mathcal{X}$ thus has a likelihood $L(x)$ assuming a fixed set of training data. This is just the (marginal) likelihood of its associated model. In the language of MCMC, the information provided by the proof tree over and above the model term is an 'auxiliary variable': part of the state space for the Markov chain but not part of the stationary distribution we are interested in. Naturally only the model term is recorded as an MCMC run progresses.

It remains to choose a proposal: how to move between proof trees. The basic idea is this: given a current proof tree $x \in \mathcal{X}$, a new one is proposed by deterministically deleting

part of x, thus producing a derivation tree x' and then using the prior π_0 to probabilistically re-grow x' to produce a proposed $y \in \mathcal{X}$. Derivation trees x' produced by deleting part of a consistent proof tree are called *backtrack points*. We begin by more precisely characterising backtrack points.

4.3.1 Backtrack points

Recall, from Section 3, the way in which an SLP (and initial goal) define a procedure which samples from the prior π_0. This procedure can be seen as the realisation of a Markov chain whose states are augmented derivation trees.

For any consistent proof tree $x \in \mathcal{X}$ and for any set of probabilistic atoms A in x let $bp(x, A)$ be the augmented derivation tree produced by deleting the trees under each $a \in A$ and reinitialising the choice point for each a to include all possible elementary trees. $bp(x, A)$ is called a *backtrack point* for x. For example, Fig 9 is a backtrack point $bp(x, A)$ where x is the consistent proof tree in Fig 10 and $A = \{$`e([c],a,[b<-a,c<-b],BN)`, `nc(BN)`$\}$. The derivation tree given in Fig 17 is a backtrack point for the tree in Fig 10 where $A =$ $\{$`e([c],b,[],TBN1)`$\}$. An important difference between the backtrack points in Fig 9 and Fig 17 is that the former must have been built at some stage in the construction of the consistent proof tree in Fig 10, whereas the tree in Fig 17 could never be built as some intermediary stage in the production of *any* consistent proof tree. This is because the SLP sampling mechanism, which defines our prior, always builds trees depth-first left-first. In Fig 17 the tree under `e([c],b,[],TBN1)` is not constructed even though the tree to its right has been constructed.

It is clear that backtrack points for any consistent proof tree are partitioned between those like Fig 9 which we shall call *chronological* backtrack points and those like Fig 17 which we shall call *non-chronological* backtrack points. Chronological backtrack points are so called since they can be reached from a consistent proof tree by 'rewinding history' to reach a stage which must have occurred at some point in the construction of that consistent proof tree.

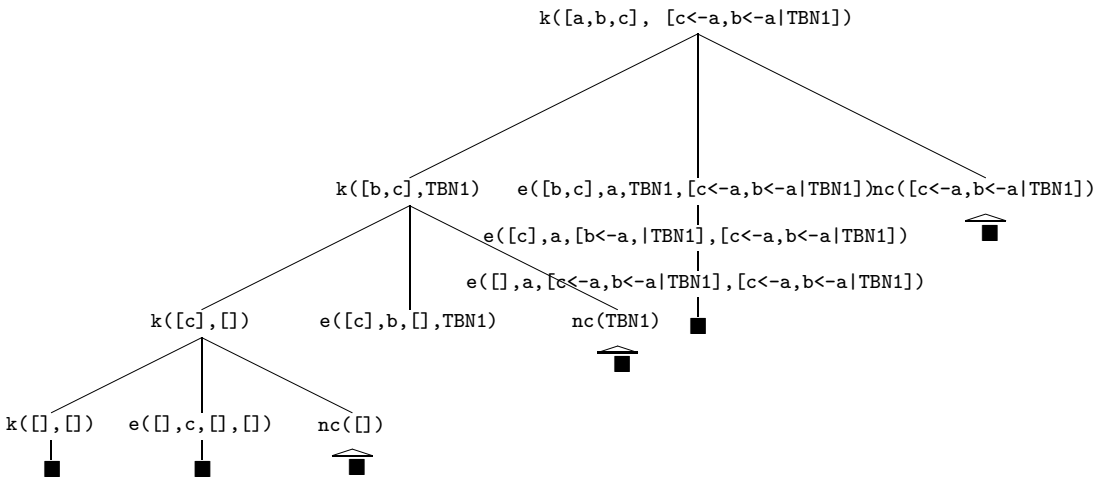

Figure 17: Non-chronological backtrack point

4.3.2 Chronological backtracking proposals

Any given consistent proof tree x has a finite number of chronological backtrack points which can be ordered chronologically. Let the number of chronological backtrack points for a consistent proof tree x be called the *depth* of x, denoted $d(x)$. (Note that this does not agree with the usual definition of the 'depth' of a tree.) Let $x_1, x_2, \ldots, x_{d(x)}$ be the chronological ordering of chronological backtrack points for x.[2] For any $k > d(x)$, define x_k to be x. It is now easy to define our first type of proposal.

Definition 4 *For any positive integer k,* the depth-k chronological backtracking proposal q_k *is defined as follows. Let the current consistent proof tree be $x \in \mathcal{X}$. To propose a new consistent proof tree move to the backtrack point x_k and use the SLP sampling mechanism defining the prior to extend x_k to a new consistent proof tree y. There will always be at least one such consistent proof tree, namely x. (Note that if $k > d(x)$ then x will be proposed.)*

Proposition 5 *Let q_k denote the depth-k chronological backtracking proposal then $\forall k, \pi_0$: $R_{q_k}(x, y) = \frac{q_k(x|y)\pi_0(y)}{q_k(y|x)\pi_0(x)} = 1$ and so*

$$\alpha_{q_k}(x, y) = \min\left\{\frac{L(y)}{L(x)}, 1\right\}$$

Proof Introducing an abuse of notation, for any augmented derivation trees x', x'' (not necessarily consistent proof trees) let $\pi_0(x')$ denote the probability that *at some point* the SLP prior-sampling mechanism produces x', and let $\pi_0(x''|x')$ denote the probability that x' gets extended to x'' by the mechanism. In the construction of a consistent proof tree x, all of x's chronological backtrack points will be passed through. It follows that $\forall k \in \{1, 2, \ldots d(x)\} : \pi_0(x) = \pi_0(x_k)\pi_0(x|x_k)$.

By definition $q_k(y|x) = \pi_0(y|x_k)$. So if $x_k = y_k$, then $\pi_0(x)q_k(y|x) = \pi_0(x_k)\pi_0(x|x_k)\pi_0(y|x_k) = \pi_0(y_k)\pi_0(x|y_k)\pi_0(y|y_k) = \pi_0(y_k)\pi_0(y|y_k)\pi_0(x|y_k) = \pi_0(y)q_k(x|y)$, and the result follows. If $x_k \neq y_k$ then $q_k(y|x) = q_k(x|y) = 0$, and a value for $\alpha_{q_k}(x, y)$ is never needed.

■

Fixed-depth backtracking is too fixed, much better is a proposal which sometimes makes small jumps (big k) and sometimes big ones (small k). This leads to our next type of proposal: *chronological uniform choice* q_{uc}. Instead of using a fixed depth k to backtrack to, k is chosen uniformly from $\{1, 2, \ldots, d(x)\}$. The acceptance probability is only slightly harder to compute:

Proposition 6 *Let q_{uc} denote the chronological uniform choice backtracking proposal then $\forall \pi_0 : R_{q_{uc}}(x, y) = \frac{q_{uc}(x|y)\pi_0(y)}{q_{uc}(y|x)\pi_0(x)} = d(x)/d(y)$ and so*

$$\alpha_{q_{uc}}(x, y) = \min\left\{\frac{d(x)}{d(y)}\frac{L(y)}{L(x)}, 1\right\}$$

2. In previous work (Angelopoulos and Cussens, 2005a) we numbered backtrack points from 0. Numbering from 1 makes some of the presentation clearer.

Proof Given $x, y \in \mathcal{X}$, let $d(x,y)$ be the largest k such that $x_k = y_k$. Note that there is always such a k since $x_1 = y_1, \forall x, y \in \mathcal{X}$. It is clear that if $k' > d(x,y)$ then $\pi_0(y|x_{k'}) = \pi_0(x|y_{k'}) = 0$. From this and the definition of q_{uc} we have the following equation:

$$\begin{aligned}
&\pi_0(x)q_{uc}(y|x) \\
&= \pi_0(x)\sum_{k=1}^{d(x)}\frac{1}{d(x)}\pi_0(y|x_k) \\
&= \pi_0(x)\frac{1}{d(x)}\sum_{k=1}^{d(x)}\pi_0(y|x_k) \\
&= \pi_0(x)\frac{1}{d(x)}\sum_{k=1}^{d(x,y)}\pi_0(y|x_k)
\end{aligned}$$

So $\pi_0(x)q_{uc}(y|x)d(x) = \pi_0(x)\sum_{k=1}^{d(x,y)}\pi_0(y|x_k)$. Similarly, $\pi_0(y)q_{uc}(x|y)d(y) = \pi_0(y)\frac{1}{d(y)}\sum_{k=1}^{d(x,y)}\pi_0(x|y_k)$. However, $\forall k : 1 \leq k \leq d(x,y) : \pi_0(x)\pi_0(y|x_k) = \pi_0(y)\pi_0(x|y_k)$ from the proof of Proposition 5. So $\pi_0(x)\sum_{k=1}^{d(x,y)}\pi_0(y|x_k) = \pi_0(y)\sum_{k=1}^{d(x,y)}\pi_0(x|y_k)$ and thus $\pi_0(x)q_{uc}(y|x)d(x) = \pi_0(y)q_{uc}(x|y)d(y)$. The result follows. ■

4.3.3 Non-chronological backtracking proposals

In practice a big problem with chronological backtracking proposals is that to change the tree under some probabilistic atom, it is necessary to destroy everything built later to its right. Non-chronological backtracking avoids this problem. To define non-chronological backtracking first note that every chronological backtrack point x_k has an associated selected probabilistic atom a_k. Each such x_k has an associated *non-chronological* backtracking point x^k which is just x with the sub-tree under a_k deleted. Both x_k and x^k have the tree under a_k deleted, the difference is that x_k also has everything 'to the right' deleted.

Fixed-depth (q^k) and uniform choice (q^{uc}) non-chronological backtracking proposals are like their chronological counterparts except that backtracking jumps to x^k rather than x_k. (Note that chronological proposals are denoted using subscripts and non-chronological via superscripts.) A tree is then built under a_k, the selected atom of x^k using the standard sampling mechanism. Note that there is at least one consistent proof tree sampleable in this fashion: namely x. In general, x^k would never be reached when sampling normally from the prior. Nonetheless we can use the prior-sampling mechanism to sample the tree under a_k, exactly as if it were any other selected atom.

In the general case, we do not (as yet) have expressions for acceptance probabilities for non-chronological backtracking. We do, however, have easily computable acceptance probabilities in an important special case. To define this special case, consider first the following sampling mechanism defining a different prior π_0^k over consistent proof trees. Until the kth choice point is created proceed as for π_0. If and when the kth choice point is created then, instead of building a tree under the atom a_k, that atom is *frozen* (to use the language of logic programming): the *next* available atom is instead selected, using the normal selection

ordering for atoms. Indeed atom a_k is only selected when no other atom is available for selection: in short the tree under a_k is built last. We now define π_0^k formally.

Definition 7 *This equation defines π_0^k: $\forall x \in \mathcal{X} : \pi_0^k(x) = \pi_0(x_k)\pi_0(x^k|x_k)\pi_0(x|x^k)$, where x_k is the kth chronological backtrack point, x^k is the kth non-chronological backtrack point and $\pi_0(x^k|x_k)$ denotes the probability of extending x_k to x^k by freezing the atom a_k and probabilistically extending x_k 'to the right' of a_k.*

In general $\pi_0 \neq \pi_0^k$. If we build the tree under a_k first (as normal) then this may produce variable bindings on atoms to be selected later thus constraining which subtrees can be built under them. These variable bindings will not be there if the tree under a_k is delayed until the end of the process. When π_0 *does equal* π_0^k we have the following useful result.

Proposition 8 *If $\pi_0 = \pi_0^k$ then $\forall x, y \in \mathcal{X} : \pi_0(x)q^k(y|x) = \pi_0(y)q^k(x|y)$.*

Proof If $x^k \neq y^k$ then $q^k(y|x) = q^k(x|y) = 0$ and the result follows, so we consider the case where $x^k = y^k$. In this case we clearly also have $x_k = y_k$

$$\begin{aligned}
&\pi_0(x)q^k(y|x) \\
&\quad = \pi_0(x_k)\pi_0(x^k|x_k)\pi_0(x|x^k)\pi_0(y|x^k) \text{ by definitions} \\
&\quad = \pi_0(y_k)\pi_0(y^k|y_k)\pi_0(x|x^k)\pi_0(y|x^k) \text{ from above} \\
&\quad = \pi_0(y)q^k(x|y) \text{ by definitions}
\end{aligned}$$

■

From Proposition 8 it follows immediately that if $\pi_0 = \pi_0^k$ then $R_{q^k}(x, y) = 1$. If $\pi_0 = \pi_0^k$ for all k then (by a small alteration to the proof of Proposition 6) we also have that $R_{q^{uc}}(x, y) = d(x)/d(y)$. In some of the experiments where we have used non-chronological backtracking we have ensured that $\pi_0 = \pi_0^k$ for all values of k which can ever be selected and so we can use these simple acceptance probabilities. This is the case when the BN structure prior defined by the SLP is *modular* (see Section 2.2.1).

4.3.4 Backtracking proposals and single-component MCMC

Restricting to modular priors runs counter to our goal of allowing the user to define priors which encapsulate whatever prior knowledge there is. So we have also done experiments using non-chronological backtracking with non-modular priors (so that $\pi_0 \neq \pi_0^k$). This amounts to pseudo-prior-Gibbs sampling for the following reasons.

It is not difficult to see that all the backtracking proposals implement variants of the prior-Gibbs sampler. Each proof sub-tree that is pruned and resampled using the prior can be viewed as a 'component' of the entire proof tree. These components form a hierarchy rather than a flat sequence as in normal single-component sampling. In the case of chronological backtracking a block of components is resampled. If the backtracking is fixed-depth then the same component (or block of components) is always selected; if uniform-choice the choice of component is randomised.

When using pseudo-prior-Gibbs sampling, in step i we propose a new set of parents for node i by pruning the proof sub-tree which determines its current parents and resampling using the SLP prior mechanism as previously described for non-chronological backtracking. With the exception of the last node, proposing like this only approximates proposing via the conditional prior. However, since the underlying logic program is the same, at least we have that zero prior probability models are never proposed (and hence have posterior probability zero). This is a partial explanation for the good results this approach has yielded. We denote our pseudo-prior-Gibbs sampler by q^{sc}.

Note that to effect this approach we need to restrict the number of available backtrack points to be only those that correspond to choosing a whole new parent set for a node. In our current implementation this restriction is achieved via a directive like:

```
:- s_no_bpoint( choose_pa/3 ).
```

which states that backtracking should ignore all backtrack points where the selected atom has `choose_pa/3` as its predicate symbol.

4.3.5 Implementation

We will give just a brief overview of how MCMC with SLPs is actually implemented: further details are available from (Angelopoulos and Cussens, 2005b). The prior π_0 is defined in a SLP source file. To sample from the prior π_0 defined by this SLP a call can be made to `bn/4` with the first three variables instantiated. When the call returns (i.e. when a consistent proof tree has been constructed), the fourth argument will be instantiated to a first-order term representing the sample BN.

Source SLPs are transformed to normal Prolog programs and loaded into memory. Each clause is augmented by extra arguments carrying the probabilistic labels and a path of (probabilistic) backtracking points. The latter is used both for choosing a proposal backtrack point and for speeding the process of finding the proposed model. Prolog's own backtracking which implements a left to right search, is used when sampling. At each iteration after the current model has been reached we need to backtrack to the chosen (proposal) backtrack point. Prolog's search strategy cannot be used effectively for backtracking to the proposal point, instead this is achieved by rerunning the original top level query with its path argument partially instantiated.

A proposal strategy has two components. Firstly, it has a path of backtrack points. Secondly, it has a function for choosing one of the points in the path. The latter is controlled by an experimental parameter. In all proposals described here, except pseudo-prior-Gibbs sampling, a point is picked uniformly from those in the path. *Subpaths* can also be created which allow non-chronological backtracking. If a point in a subpath is chosen, all points not in the subpath are assumed independent to those in the subpath and it is unaffected by the transition from the current model to the proposed one. In effect, this allows changes to a part of the model which are independent of other subparts.

5. Experimental setup

Since the goal of our experiments is to test our MCMC system rather than to perform Bayesian inference for some particular application, all our experiments are done using syn-

BN	Nodes	Arcs	Ref
ASIA	8	8	Lauritzen and Spiegelhalter (1988)
ALARM	37	46	Beinlich et al. (1989)
HAILFINDER	56	66	Abramson et al. (1996)
INSURANCE	27	52	Binder et al. (1997)
PARITY1	22	31	Koivisto and Sood (2004)
PARITY2	100	53	Koivisto and Sood (2004)

Table 1: True BNs

thetic data. In each experiment a known fully parameterised Bayesian network (TrueBN) is chosen and a specified number (Size) of complete joint instantiations are generated from TrueBN by forward sampling. This constitutes the data for an experiment. Next an SLP prior (SLPPrior), prior precision for the Dirichlet parameters (N) and proposal (Proposal) are selected. Using the selected data, prior and proposal the Metropolis-Hastings algorithm is run for 250,000 iterations with a record of all visited BNs and their marginal log-likelihoods recorded. Each such run is done three times with three different random seeds. Size was either 500 or 1000. N was either 1 or 10. The following three sections describe each of the other experimental variables (TrueBN, SLPPrior and Proposal) in more detail.

5.1 True Bayesian networks

Six different BNs were used to generate synthetic data: ASIA, ALARM, HAILFINDER, INSURANCE, PARITY1 and PARITY2. Their sizes are given in Table 1 which also cites the original references which provide further information on them. The six values for TrueBN were chosen to give a range of structure size and topology, and because each had been used at least once in previous (non-Bayesian) BN structure learning work.

5.2 Prior distributions

In total 6 prior distributions were used with differing levels of information about the true BN. For all of our 6 priors, any BN with a topological ordering conflicting with that of the true BN has zero prior probability. The priors are listed below where we have indicated whether each is *modular* (see Section 2.2.1).

Prior A In this prior a consistent topological ordering is enforced: a node A can only be a parent of a node B if A comes before B in a particular (arbitrary) topological ordering of the nodes in the true BN. If A is a possible parent for B, then $P(A \to B \in BN) = 1/2$, for all A, B. MODULAR

Prior B The topological ordering constraint is imposed as for Prior A. For any node B, let $|Pa(B)|$ be the number of parents B has in the true BN. If A is a possible parent for B, then $P(A \to B \in BN) \propto |Pa(B)|$, for all A, B. MODULAR

Prior C This is as Prior B, except that the prior is told which nodes are orphans, i.e. any BN with the wrong set of orphans has zero probability. MODULAR

SLPPrior	A	B	C	C	C	D	C	E	F
Proposal	$q_{uc(1)}$	$q_{uc(1)}$	$q_{uc(1)}$	$q^{uc(2)}$	$q^{uc(3)}$	$q_{uc(4)}$	$q_{uc(5)}$	$q^{uc(2)}$	q^{sc}
Combo	1	2	3	4	5	6	7	8	9

Table 2: Prior + proposal combinations used in our experiments

Prior D This is as Prior C except that the prior knows all pairs of variables which are truly independent, i.e. any BN which does not respect these independence relations will have zero probability. NOT MODULAR

Prior E A variant of Prior B. For each node the number of parents is always the true number. The prior chooses uniformly from each possible parent set. MODULAR

Prior F Like Prior E, except that the prior, like Prior D, also knows true independence relations. NOT MODULAR

5.3 Proposals

Proposals differ only in terms of backtracking.

Proposal $q_{uc(1)}$ A backtrack point is chosen uniformly from all possible *chronological* backtrack points.

Proposal $q^{uc(2)}$ A backtrack point is chosen uniformly from a subset of *non-chronological* backtrack points. There is one backtrack point for each node in the BN. For each node, backtracking to its associated backtrack point has the effect of removing all arrows pointing to that node, so that it has no parents. New parents will subsequently be chosen (according to whichever prior is being used) for that node.

Proposal q^{sc} Similar to $q^{uc(2)}$ except that a single-component MH approach is taken. See Section 4.3.4 for more on this proposal. Note that on each iteration a new parent set is proposed for each node, so that each iteration is more computationally intensive than for other proposals.

Proposal $q^{uc(3)}$ Like Proposal $q^{uc(2)}$ except that no backtrack points for true orphan nodes are considered.

Proposal $q_{uc(4)}$ Like Proposal $q^{uc(2)}$, except that only chronological backtrack points are considered.

Proposal $q_{uc(5)}$ Chronological version of $q^{uc(3)}$.

5.4 Experiments actually run

Given that we have 6 values for TrueBN, 2 values for Size, 6 values for SLPPrior, 2 values for N, 4 values for Proposal and 3 different random seeds are used for each MCMC run, there were $6 \times 2 \times 6 \times 2 \times 4 \times 3 = 1728$ runs of 250,000 iterations that we could have run. We judged that our approach could be evaluated with significantly fewer runs. Firstly, not

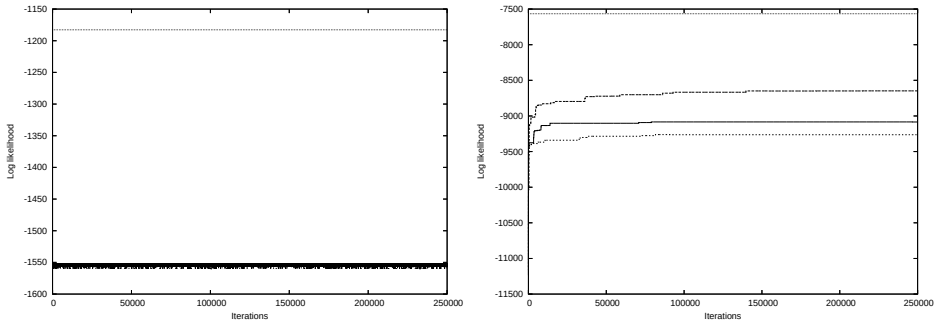

Figure 18: Log-likehood trajectories for 3 different runs. LHS+RHS: Size = 500, N = 10, SLPPrior = D, Proposal = $q_{uc(4)}$ (Combo 6). LHS: TrueBN = ASIA. RHS: TrueBN = INSURANCE

all 24 prior+proposal combinations were considered. Instead only those 9 combinations shown in Table 2 were tried. From now on Combo will be used to refer to a prior+proposal combination.

Secondly, as discussed in Section 6, most choices for Combo led to demonstrably poor MCMC convergence. Since all experiments for N = 10 were done prior to those for N = 1, we chose to restrict N = 1 runs to only those 4 values of Combo (4, 5, 8 and 9) which had shown reasonable convergence in the N = 10 experiments. So for N=10, runs were done for each combination of TrueBN, Combo, Size and seed leading to $6 \times 9 \times 2 \times 3 = 324$ runs, whereas for N = 1 only $6 \times 4 \times 2 \times 3 = 144$ runs were done.

6. Results

6.1 Negative convergence results with chronological proposals

As previously mentioned some Combos showed clear evidence of poor convergence. Poor convergence was observed in all cases where non-chronological backtracking was used *even when the prior was highly biassed towards the true BN*. Consider, for example, Combo 6 which has SLPPrior = D and Proposal = $q_{uc(4)}$. SLPPrior D is highly biassed towards the true BN since it rules out any BN which does not respect all the conditional independence relations in the true BN. Nonetheless the poor movement through the state space which a chronological proposal affords leads to poor results. Fig 18 shows log-likelihood trajectories produced from MCMC runs applying Combo 6 to 500 datapoints generated from ASIA and INSURANCE, respectively. In both cases N, the prior precision is set to 10. Trajectories from the 3 independent runs are shown as well as a horizontal line giving the log-likelihood of the true model. This is, of course, the marginal log-likelihood, $\log L(x)$ where $L(x)$ is given by (5). All log-likelihood trajectories will be of this format in this paper.

In the ASIA case the three different runs do at least give similar results, so that in the LHS figure it is difficult to distinguish the different trajectories. However, in each case no model is visited with a log-likelihood even close to that of the true one. The ASIA BN is a very simple learning task, a more challenging task is given, for example, by INSURANCE. The INSURANCE trajectory shows quite clearly that the 3 MCMC runs using Combo 6

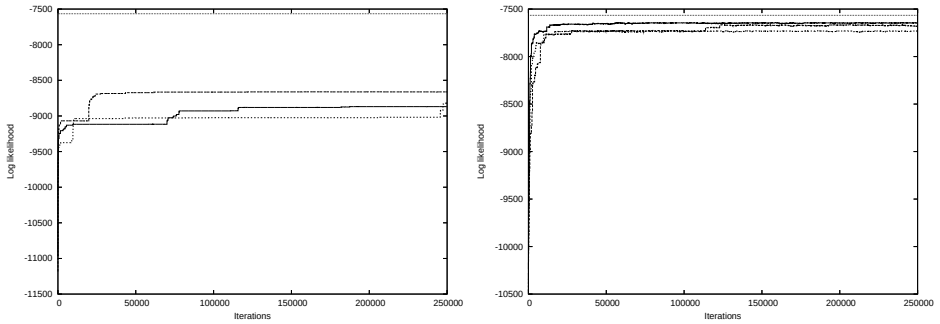

Figure 19: Log-likehood trajectories for 3 different runs. Both: TrueBN = INSURANCE, Size = 500, N = 10, SLPPrior = C. LHS: Proposal = $q_{uc(1)}$ (Combo 3). RHS: Proposal = $q^{uc(2)}$ (Combo 4)

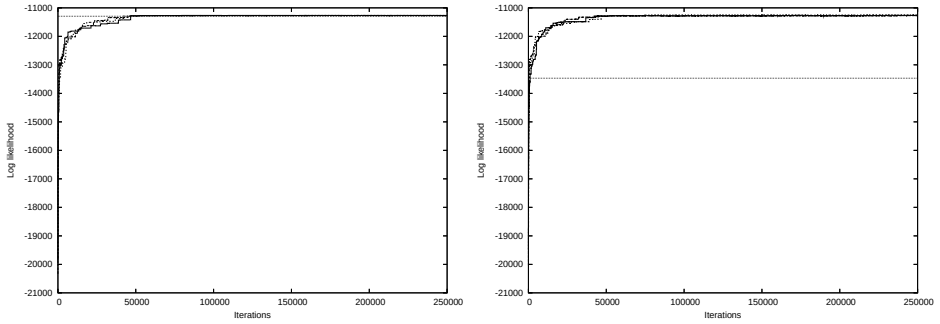

Figure 20: Log-likehood trajectories for 3 different runs. Both: TrueBN = ALARM, Size = 1000, SLPPrior = C, Proposal = $q^{uc(3)}$ (Combo 5). LHS: N = 1. RHS: N = 10

each get stuck in different parts of the model space. The trajectories in Fig 18 are quite typical of the 'sticking' problems which occur using a chronological proposal.

Combos 3 and 4 have the same prior (C) with the former having a chronological proposal and the latter a non-chronological one, so comparing them brings out the advantage of non-chronological proposals particularly clearly. Fig 19 shows the trajectories for these 2 Combos on runs identical in all other respects. The non-chronological Combo 4 comes closer to converging and moreover explores models close (in log-likelihood) to the true BN.

6.2 Convergence for non-chronological priors

From now on we only consider experiments done using non-chronological proposals. Convergence, as far as can be judged by log-likelihood trajectories, is generally good for these proposals, in contrast to chronological ones. In all cases runs for Size = 500 and Size = 1000 tell a similar story, so we will restrict attention to the latter since it is the harder learning scenario. It is harder because with a greater amount of data the posterior will be further from the prior making it harder for our Metropolis-Hastings algorithm (driven by proposals guided by the prior) to converge.

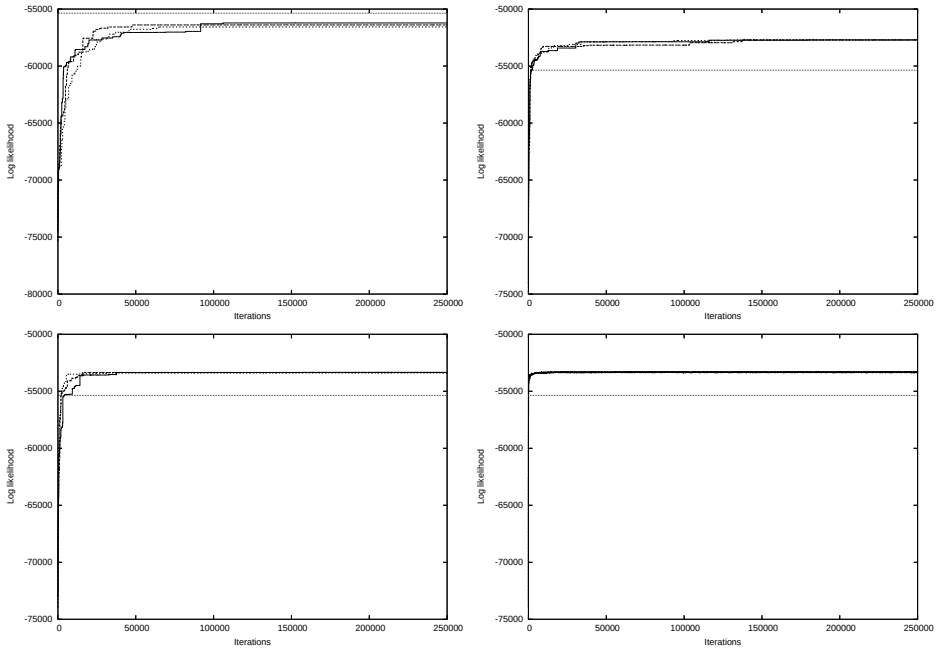

Figure 21: Log-likehood trajectories for 3 different runs. All: TrueBN = HAILFINDER, Size = 1000, N = 1, SLPPrior = C. Top-left: Proposal = $q^{uc(2)}$ (Combo 4). Top-right: Proposal = $q^{uc(3)}$ (Combo 5). Bottom-left: Proposal = $q^{uc(2)}$ (Combo 8). Bottom-right: Proposal = q^{sc} (Combo 9). [Note that the top-left y-scale differs from the others.]

In most cases, trajectories for $N = 1$ and $N = 10$ were similar. The exception to this was the set of results for TrueBN = ALARM. For ALARM, runs for $N = 10$ consistently found models whose log-likelihoods were above that of the true model but this was not the case for $N = 1$. Fig 20 shows a typical example of this contrast using Combo 5.

From now we will restrict attention to the case where prior precision N was set to 1. The basic story is that Combos 5, 8 and 9 consistently gave better results than Combo 4 and that PARITY2 was significantly more difficult than the other BNs. Fig 21 shows results for HAILFINDER which are similar to those obtained for all other BNs apart from PARITY2. For Combos 5, 8 and 9 each of the 3 realisations of the Markov chain finds its way to a region of the model space with high scoring models and stays there. Note that Combo 9 converges especially rapidly. For HAILFINDER, INSURANCE and PARITY1 this region contains models with log-likelihood scores significantly higher than that of the true model. For ASIA and ALARM the scores are the same.

These phenomena can be explained as follows. Recall that each likelihood we consider is a *marginal* likelihood which is a 'weighted average' of fitted likelihoods for the parameterised model for each possible choice of parameters. (The 'weights' are given by the choice of Dirichlet distribution over possible parameters.) Modulo sampling variation, the *fitted* likelihood of TrueBN will be high if parameterised with the 'true' parameters (the ones used

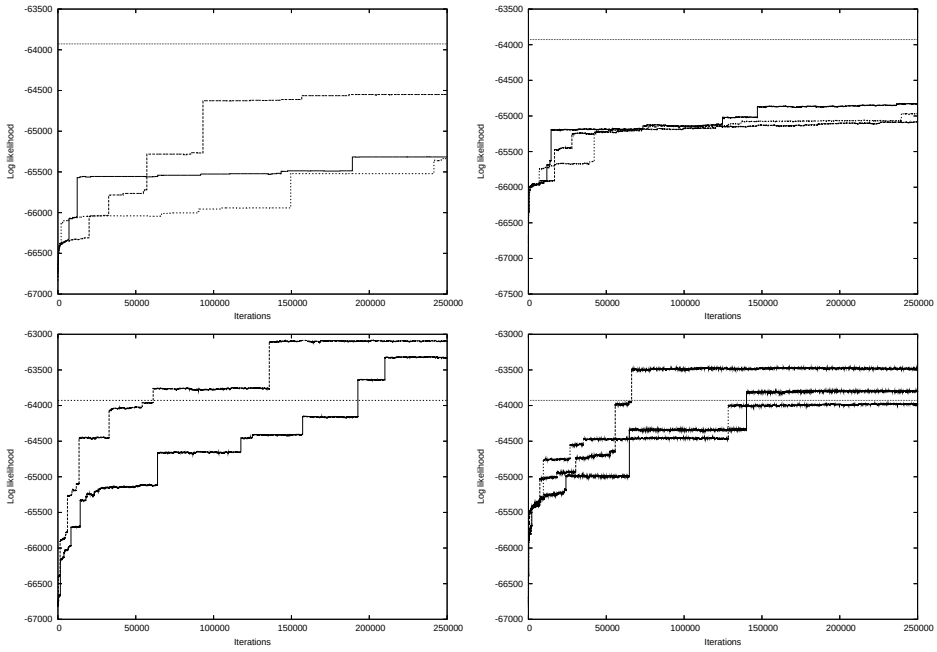

Figure 22: Log-likehood trajectories for 3 different runs. All: TrueBN = PARITY2, Size = 1000, N = 1, SLPPrior = C. Top-left: Proposal = $q^{uc(2)}$ (Combo 4). Top-right: Proposal = $q^{uc(3)}$ (Combo 5. Bottom-left: Proposal = $q^{uc(2)}$ (Combo 8). Bottom-right: Proposal = q^{sc} (Combo 9). [Note different y-axis scales.]

to generate the data) but there may be large areas of the parameter space defining fitted models with low fitted likelihood leading to a low marginal likelihood overall.

We can get a rough idea of the extent of this problem by looking at the number and proportion of true parameters of value zero for various models. We have the following (with some rounding) HAILFINDER = 501/3760 = 13.3%, INSURANCE = 302/1419 = 21.3%, ASIA = 4/36 = 11.1%, ALARM = 5/747 = 1.0%, PARITY1 = 0/194 = 0.0%, PARITY2 = 0/496. Roughly speaking, a high number of zero parameters puts the true parameters towards the edge of the parameter space, and so most other parameterisations of the true structure, being distant from these true parameters, lead to low fitted likelihoods and thus a low marginal likelihood overall. Clearly, this also obtains if we have many near-zero true parameters as well—which happens to be the case for PARITY1 and to a lesser extent for PARITY2. We conjecture that the small number of zeroes for the true parameters of ASIA and ALARM explain why these true BNs had high marginal log-likelihood.

PARITY2 is atypical in that our chains show clear evidence of non-convergence. Note though that, for Combos 8 and 9 the MCMC runs nearly always end up visiting models with higher log-likelihoods than the true PARITY2 model. These phenomena are demonstrated by Fig 22.

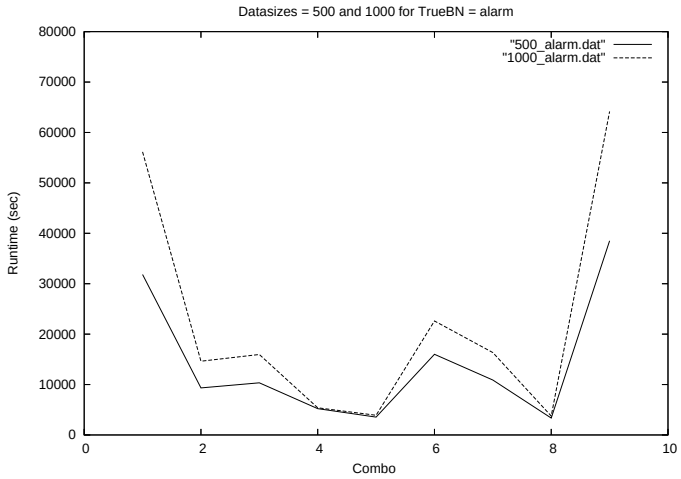

Figure 23: Runtimes for TrueBN=Alarm and Sizes of 500 and 1000. Combos order: 1-9

6.3 Running times for MCMC

The experiments were run on SICStus Prolog (version 3.12). The computers used run the Linux operating system and have modest workstation capabilities (3-4GHz single processor, 0.5 or 1GB memory). We recorded runtimes that include total cpu time and garbage collection. Figures 23 and 24 illustrate some of the execution times. In all cases: the y-axis plots runtimes in seconds, the x-axis plots Combos, each value is the average of three separate runs, and all times are from experiments where $N = 10$. Experiments with $N = 1$ are very similar as the only difference is the value of a constant with no significant effect on performance.

Runtimes vary greatly depending on four main factors. The average size of (BN) families, the number of known orphans, the size of the data and the time taken at each iteration. Fig 23 illustrates the effect of Size on execution time. The x-axis shows Combos in the same order as that of Table 2. In Combos 4, 5 and 8 the y-values for Size = 1000, is only marginally larger than the corresponding values for Size = 500. In all three cases, the proposal is non-chronological and at each iteration a single family is reconsidered and it usually is a small one. The computation of the likelihood, which depends on the size of the data, in these cases is very fast. From the remaining Cobmos we can deduce that the likelihood computation is a major contributor to the overall runtimes as the difference in y-values between 500 and 1000 is almost half the variation from the base cases of Combos 4, 5 and 8.

The two plots in Fig 24 separate the TrueBNs into two groups which accommodate the variability of the displayed runtimes. Hailfinder and Parity2 register longer execution times as they have more variables than the rest of the TrueBNs. The two plots are drawn on different y-axis ranges. Note that Combo 9, the leftmost on x-axis, although non-chronological, is by design spending much longer than other non-chronological proposals at each iteration, as each and every family in the network are reconsidered in turn.

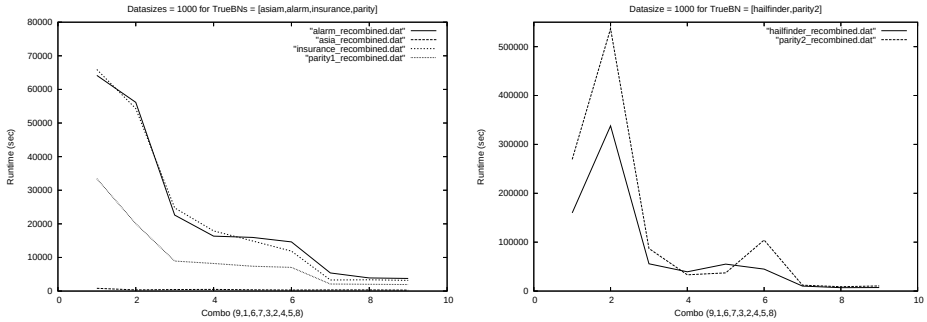

Figure 24: LHS: runtimes versus Combo for TrueBNs Alarm, Asian, Insurance and Parity1. RHS: runtimes versus Combo for Hailfinder and Parity2. Combos' order: 9,1,6,7,3,2,4,5,8. Note that the y-axis ranges are not drawn to the same scale

6.4 Concentration of the posterior around the true model

Log-likelihood trajectories constitute a useful tool providing evidence for convergence/non-convergence. However, we are interested not only in how well our MCMC runs allows us to approximate the posterior, but also in the nature of the posterior itself. Specifically, to what extent is the posterior distribution concentrated around the true BN?

To address this question it is necessary to introduce a measure of 'distance' between any given BN and the true BN. We do this in the standard way by considering various *loss functions*. A loss function measures the loss one would suffer by using a given BN instead of the true one (so that the loss of the true BN is always zero). There is no such thing as a 'correct' universal loss function since the loss suffered depends on what a user is doing with the learned BNs, and this will vary between applications. We consider three simple loss functions:

Naive 0-1 loss (ID) is a crude loss function where it is necessary to identify the correct BN structure *exactly* to avoid a maximal loss of 1.

$$L_{\mathrm{ID}}(BN, \mathsf{TrueBN}) = \begin{cases} 0 & \text{if } BN = \mathsf{TrueBN} \\ 1 & \text{otherwise} \end{cases} \tag{10}$$

Naive edge symmetric difference loss (NE) penalises a BN for each extra or missing edge it has with respect to TrueBN. Let E_{BN} be the set of edges in BN and let E_{TrueBN} be the set of edges in E_{TrueBN}. To allow easier comparison we normalise: let n be the number of vertices in the true BN, then we have

$$L_{\mathrm{NE}}(BN, \mathsf{TrueBN}) = \frac{|E_{BN} \triangle E_{\mathsf{TrueBN}}|}{n^2} \tag{11}$$

Given the normalisation used, $L_{\mathrm{NE}}(BN, \mathsf{TrueBN})$ is the probability that a randomly chosen pair of vertices are correctly (dis)connected. This was essentially the structural loss function used in Heckerman et al. (1995a) "the structural difference we use is $\sum_{i=1}^{n} \delta_i$ where δ_i is the symmetric difference of the parents of x_i in the gold-standard network and the parents of x_i in the learned network."

Essential graph edge symmetric difference (CK) is the same as naive edge symmetric difference except that the essential graph of the BN is compared to the essential graph of TrueBN. An undirected edge $A - B$ in an essential graph is considered to be a pair of directed edges $A \to B, A \leftarrow B$. This is the metric used by Castelo and Kočka (2003).

Define the *posterior expected loss* $E_{\pi,\mathsf{TrueBN}}(L)$ for a given posterior π, loss function L, and true model TrueBN:

$$E_{\pi,\mathsf{TrueBN}}(L) = \sum_{BN} L(BN, \mathsf{TrueBN})\pi(BN) \tag{12}$$

$E_{\pi,\mathsf{TrueBN}}(L)$ essentially measures the cost of picking a BN randomly chosen from π and using it instead of TrueBN. The more the posterior is concentrated around BNs 'near' to TrueBN, the smaller $E_{\pi,\mathsf{TrueBN}}(L)$ will be.

Posterior expected loss is a measure of how concentrated the posterior is around the true model. We only have the approximation to the posterior supplied by MCMC, but since, with the exception of PARITY2, convergence appears reasonable we can use the sample MCMC to provide approximations to (12). Table 3 show the results of doing this for all combinations of TrueBN, non-chronological Proposal and loss function. Results using the unsuccessful Combo 1 are also provided to provide a baseline. In each case posterior loss estimates from 3 different MCMC runs are given.

Table 3 shows the following:

- NE and CK results unsurprisingly tell a similar story. This is essentially because at most one representative from each Markov equivalence class is in our model space. We restrict attention to ID and NE loss from now on.
- The baseline Combo 1 NE results are often close to 1/4, which is the expected NE loss for a uniform distribution over the set of BNs which respect the variable ordering.
- Results are generally stable across MCMC runs.
- The Loss=ID results show that the true BN is *never visited* except when TrueBN is ASIA and an informative prior is used. Note that, since we impose the true variable ordering as a constraint, at most one representative from each Markov equivalence class of BNs is in the model space. It follows that, with the exception just noted, no BN in the true Markov equivalence class is visited.
- Recall that $L_{\mathrm{NE}}(BN, \mathsf{TrueBN})$ measures the probability that a pair of vertices in BN is correctly (dis)connected. So, notwithstanding the previous point, the many very low values for NE loss show that these posteriors are concentrated on BNs which are structurally similar to the true BN, even for PARITY2.

6.5 Single model learning

To provide a comparison with standard Bayesian net learning algorithms we extracted from each MCMC run: BN_{MAP}, the most visited model and BN_{MLE}, the model with

ASIA									
Combo	ID			NE			CK		
1	1.00	1.00	1.00	0.1215	0.1252	0.1286	0.1474	0.1500	0.1539
4	1.00	1.00	1.00	0.0934	0.1221	0.0933	0.1396	0.1552	0.1399
5	0.99	0.99	0.99	0.0433	0.0423	0.0432	0.0654	0.0642	0.0649
8	0.16	0.16	0.16	0.0052	0.0051	0.0050	0.0053	0.0051	0.0051
9	0.16	0.16	0.16	0.0051	0.0052	0.0051	0.0051	0.0052	0.0052
ALARM									
Combo	ID			NE			CK		
1	1.00	1.00	1.00	0.1953	0.1982	0.1981	0.1971	0.2007	0.2012
4	1.00	1.00	1.00	0.0216	0.0200	0.0226	0.0245	0.0229	0.0255
5	1.00	1.00	1.00	0.0086	0.0092	0.0087	0.0108	0.0115	0.0110
8	0.99	1.00	1.00	0.0030	0.0032	0.0032	0.0031	0.0032	0.0032
9	1.00	1.00	1.00	0.0028	0.0029	0.0029	0.0029	0.0029	0.0029
HAILFINDER									
Combo	ID			NE			CK		
1	1.00	1.00	1.00	0.2401	0.2384	0.2300	0.2433	0.2416	0.2326
4	1.00	1.00	1.00	0.0271	0.0251	0.0235	0.0290	0.0273	0.0259
5	1.00	1.00	1.00	0.0110	0.0115	0.0113	0.0143	0.0149	0.0145
8	1.00	1.00	1.00	0.0123	0.0115	0.0115	0.0151	0.0147	0.0148
9	1.00	1.00	1.00	0.0118	0.0117	0.0116	0.0151	0.0149	0.0148
INSURANCE									
Combo	ID			NE			CK		
1	1.00	1.00	1.00	0.1907	0.1820	0.1923	0.2004	0.1878	0.1997
4	1.00	1.00	1.00	0.0555	0.0547	0.0547	0.0664	0.0663	0.0659
5	1.00	1.00	1.00	0.0555	0.0547	0.0547	0.0664	0.0663	0.0659
8	1.00	1.00	1.00	0.0307	0.0305	0.0299	0.0397	0.0394	0.0386
9	1.00	1.00	1.00	0.0300	0.0300	0.0301	0.0389	0.0389	0.0391
PARITY1									
Combo	ID			NE			CK		
1	1.00	1.00	1.00	0.2220	0.2306	0.2295	0.2242	0.2372	0.2345
4	1.00	1.00	1.00	0.1227	0.1206	0.1195	0.1248	0.1264	0.1257
5	1.00	1.00	1.00	0.1136	0.1096	0.1123	0.1165	0.1125	0.1152
8	1.00	1.00	1.00	0.0990	0.0990	0.0999	0.0995	0.0994	0.1004
9	1.00	1.00	1.00	0.0988	0.0995	0.0990	0.0993	0.1000	0.0995
PARITY2									
Combo	ID			NE			CK		
1	1.00	1.00	1.00	0.2469	0.2475	0.2492	0.2469	0.2475	0.2492
4	1.00	1.00	1.00	0.0075	0.0069	0.0074	0.0075	0.0069	0.0076
5	1.00	1.00	1.00	0.0062	0.0061	0.0063	0.0062	0.0062	0.0064
8	1.00	1.00	1.00	0.0062	0.0062	0.0071	0.0062	0.0062	0.0072
9	1.00	1.00	1.00	0.0060	0.0059	0.0060	0.0060	0.0060	0.0060

Table 3: Estimated posterior expected loss results for all BNs

the highest log-likelihood. BN_{MAP} is an estimate for the BN with the highest posterior probability under the prior encoded by the SLP and BN_{MLE} is an estimate of the highest posterior probability model with a uniform prior distribution over models with positive prior probability according to the SLP prior. Table 4 show the ID and NE loss for these models for all TrueBNs and Combos 1, 4, 5, 8 and 9.

Table 4 shows the following:

- Results are generally stable across different realisations of the same Markov chain.
- Excluding the baseline Combo 1: For ASIA, MLE and MAP losses were always identical; for ALARM, INSURANCE and PARITY1 MLE has lower or equal loss, for HAILFINDER and PARITY2 results were mixed. This suggests that the principal influence given by our priors is the hard constraint effected by zero prior probabilities.
- As expected both MAP and MLE losses are smaller than posterior expected loss: selecting an appropriate model using the (approximate) posterior renders a better model than randomly selecting one according to the posterior.

In two cases we can do a rough comparison with existing work. Castelo and Kočka (2003) (among many other experiments) apply their HCMC algorithm to a dataset of 1000 points sampled from the ALARM BN. They run this (randomised) algorithm with different parameter settings, the best CK score averaging at 0.0073. So, as Tables 3 and 4 show, the basic story is that our ALARM results (whether expected loss, MAP loss or MLE loss) are worse than theirs with Combos 1, 4 and 5 and better with the Combos 8 and 9. Acid and de Campos (2003) also learn from data sampled from ALARM. Their smallest dataset has 3000 datapoints, from which their algorithm learned a BN with an NE loss of only 0.0014, better than all our scores, but this is using three times as much data. They also did experiments on datasets generated from INSURANCE and HAILFINDER. For INSURANCE with 10,000 datapoints the learned BN had NE loss of 0.0247 (a little better than our scores produced using 1,000 datapoints). For HAILFINDER, again with 10,000 datapoints, an NE loss of only 0.00765 was achieved: considerably better than our HAILFINDER losses (using only 1,000 data points) which are in the 0.011-0.012 region.

7. Conclusions

The research presented here lies at the intersection of two research areas: Bayesian model averaging of BNs via MCMC (Madigan and York, 1995; Friedman and Koller, 2003) and the use of rich representations to represent the model space (Langseth and Nielsen, 2003; Segal et al., 2005). In our case the representation language (first-order logic) and the MCMC approach are tightly coupled since the Metropolis-Hastings proposals operate by pruning and re-growing proof trees.

First-order logic has proved to be an effective way of incorporating domain knowledge into the prior. For example, to add pairwise independence constraints the d-separation criterion was written in clausal logic (concretely a Prolog program) and just added as an extra constraint on what a legal BN was. It was straightforward to apply (variants of) many of the approaches to defining priors described in Section 2. The system employed, called MCMCMS, is not specific to BNs; other recent work has applied it to Bayesian learning of

		ASIA						ALARM					
C	BN	ID			NE			ID			NE		
1	MAP	1	1	1	0.0625	0.0781	0.0937	1	1	1	0.1869	0.1950	0.1972
1	MLE	1	1	1	0.0937	0.0937	0.0937	1	1	1	0.2132	0.1957	0.1957
4	MAP	1	1	1	0.0937	0.1093	0.0937	1	1	1	0.0182	0.0175	0.0204
4	MLE	1	1	1	0.0937	0.1093	0.0937	1	1	1	0.0189	0.0175	0.0197
5	MAP	1	1	1	0.0312	0.0312	0.0312	1	1	1	0.0073	0.0080	0.0087
5	MLE	1	1	1	0.0312	0.0312	0.0312	1	1	1	0.0065	0.0058	0.0065
8	MAP	0	0	0	0.0	0.0	0.0	1	1	1	0.0029	0.0029	0.0029
8	MLE	0	0	0	0.0	0.0	0.0	1	1	1	0.0029	0.0029	0.0029
9	MAP	0	0	0	0.0	0.0	0.0	1	1	1	0.0029	0.0029	0.0029
9	MLE	0	0	0	0.0	0.0	0.0	1	1	1	0.0029	0.0029	0.0029
		HAILFINDER						INSURANCE					
Combo	BN	ID			NE			ID			NE		
1	MAP	1	1	1	0.2397	0.2382	0.2292	1	1	1	0.1893	0.1810	0.1865
1	MLE	1	1	1	0.2397	0.2382	0.2302	1	1	1	0.1906	0.1810	0.1906
4	MAP	1	1	1	0.0277	0.0258	0.0239	1	1	1	0.0548	0.0562	0.0576
4	MLE	1	1	1	0.0277	0.0232	0.0229	1	1	1	0.0562	0.0534	0.0521
5	MAP	1	1	1	0.0102	0.0092	0.0095	1	1	1	0.0301	0.0329	0.0329
5	MLE	1	1	1	0.0117	0.0092	0.0098	1	1	1	0.0301	0.0329	0.0329
8	MAP	1	1	1	0.0140	0.0108	0.0108	1	1	1	0.0301	0.0301	0.0301
8	MLE	1	1	1	0.0133	0.0121	0.0102	1	1	1	0.0274	0.0301	0.0274
9	MAP	1	1	1	0.0127	0.0102	0.0108	1	1	1	0.0301	0.0301	0.0301
9	MLE	1	1	1	0.0114	0.0121	0.0114	1	1	1	0.0301	0.0301	0.0301
		PARITY1						PARITY2					
Combo	BN	ID			NE			ID			NE		
1	MAP	1	1	1	0.2066	0.2272	0.2272	1	1	1	0.2466	0.2470	0.2486
1	MLE	1	1	1	0.2086	0.2272	0.2293	1	1	1	0.2465	0.2475	0.2488
4	MAP	1	1	1	0.1198	0.1239	0.1198	1	1	1	0.0075	0.0071	0.0072
4	MLE	1	1	1	0.1219	0.1198	0.1198	1	1	1	0.0074	0.0065	0.0076
5	MAP	1	1	1	0.1115	0.1095	0.1115	1	1	1	0.0056	0.0058	0.0060
5	MLE	1	1	1	0.1157	0.1095	0.1074	1	1	1	0.0063	0.0058	0.0058
8	MAP	1	1	1	0.0950	0.0991	0.0991	1	1	1	0.0054	0.0060	0.0064
8	MLE	1	1	1	0.1033	0.0991	0.0991	1	1	1	0.0054	0.0064	0.0078
9	MAP	1	1	1	0.0950	0.1033	0.1033	1	1	1	0.0058	0.0058	0.0060
9	MLE	1	1	1	0.0909	0.0991	0.0950	1	1	1	0.0058	0.0058	0.0060

Table 4: Loss figures for MAP and MLE models for all BNs

classification trees (Angelopoulos and Cussens, 2005a,c). The first of these papers shows how to apply the distance based approach to priors (see Section 2.2.2) to classification trees. In the work on classification trees, non-chronological priors have also been used to good effect.

A central goal of this research has been investigate the effect of informative structural priors on MCMC for Bayesian nets. To this end even our least informative priors have made use of the true variable ordering. But, given a variable ordering, *exact* Bayesian approaches are possible, so the MCMC methods in this case are questionable. For example, Koivisto and Sood (2004) present an algorithm which finds the MAP BN given an ordering. Exact computation of posterior quantities is also exploited by Friedman and Koller (2003) as a 'subroutine' in MCMC over variable orderings. However, in both these cases it is required that the structural priors are modular. Here this is not required, although to use non-modular priors we do have to resort to pseudo-prior-Gibbs. Note that the restriction to modular structural priors has to be dropped to allow independence constraints to be used.

Our results indicate that our most complex proposal: pseudo-prior-Gibbs, is the most successful despite being the theoretically flawed one. Future work will focus on this approach. Although our approach is intended for use when domain knowledge is there to be exploited and a variable ordering is the most common sort of domain knowledge, it would be interesting to use priors which do not use such an ordering, perhaps compensating with (conditional) independence constraints. Even with this best proposal our MCMC runs failed on PARITY2. A possible solution is to employ *tempering* which we used successfully for classification trees (Angelopoulos and Cussens, 2005c), and it has been successfully used by Laskey and Myers (2003) for Bayesian nets.

Acknowledgements

This work was partially supported by UK EPSRC MathFIT project *Stochastic Logic Programs for MCMC* (GR/S30993/01). We would like to thank Mikko Koivisto and Kismat Sood for supplying their datasets.

References

B. Abramson, J. Brown, A. Murphy, and R. L. Winker. Hailfinder: A Bayesian system for forecasting severe weather. *International Journal of Forecasting*, 12:57–71, 1996.

Silvia Acid and Luis M. de Campos. Searching for Bayesian network structures in the space of restricted acyclic partially directed graphs. *Journal of Artificial Intelligence Research*, 18:445–490, 2003.

Christophe Andrieu, Nando de Freitas, Arnaud Doucet, and Michael I. Jordan. An introduction to MCMC for Machine Learning. *Machine Learning*, 50:5–43, 2003.

Nicos Angelopoulos and James Cussens. Markov chain Monte Carlo using tree-based priors on model structure. In Jack Breese and Daphne Koller, editors, *Proceedings of the Seventeenth Annual Conference on Uncertainty in Artificial Intelligence (UAI–2001)*, Seattle, August 2001. Morgan Kaufmann. URL `ftp://ftp.cs.york.ac.uk/pub/aig/Papers/james.cussens/uai01.ps.gz`.

Nicos Angelopoulos and James Cussens. Extended stochastic logic programs for informative priors over C&RTs. In Rui Camacho, Ross King, and Ashwin Srinivasan, editors, *Proceedings of the work-in-progress track of the Fourteenth International Conference on Inductive Logic Programming (ILP04)*, pages 7–11, Porto, September 2004a. URL `ftp://ftp.cs.york.ac.uk/pub/aig/Papers/james.cussens/ilp04_wip.pdf`.

Nicos Angelopoulos and James Cussens. On the implementation of MCMC proposals over stochastic logic programs. In *Colloquium on Implementation of Constraint and LOgic Programming Systems. Satellite workshop to ICLP'04*, Saint-Malo, France, 2004b. URL `ftp://ftp.cs.york.ac.uk/pub/aig/Papers/james.cussens/ciclops04.ps.gz`.

Nicos Angelopoulos and James Cussens. Exploiting informative priors for Bayesian classification and regression trees. In *Proc. 19th International Joint Conference on AI (IJCAI-05)*, Edinburgh, August 2005a.

Nicos Angelopoulos and James Cussens. *MCMCMS 0.3.4 User Guide*. University of York, 2005b. URL `http://www.cs.york.ac.uk/aig/slps/mcmcms/MCMCMS_uguide.html`.

Nicos Angelopoulos and James Cussens. Tempering for Bayesian C&RT. In *Proceedings of the 22nd International Conference on Machine Learning (ICML05)*, Bonn, 2005c.

I. A. Beinlich, H. J. Suermondt, R. M. Chavez, and G. F. Cooper. The alarm monitoring system: A case study with two probabilistic inference techniques for belief networks. In *Proceedings of the European Conference on Artificial Intelligence in Medicine*, pages 247–256, 1989.

J. Binder, D. Koller, S. Russell, and K. Kanazawa. Adaptive probabilistic networks with hidden variables. *Machine Learning*, 29:213–244, 1997.

Susanne G. Bøttcher and Claus Dethlefsen. deal: A package for learning Bayesian networks. *Journal of Statistical Software*, 8(20), 2003.

W. L. Buntine. Theory refinement of Bayesian networks. In Bruce D'Ambrosio, Philippe Smets, and Piero Bonissone, editors, *Proceedings of the Seventh Annual Conference on Uncertainty in Artificial Intelligence (UAI–1991)*, pages 52–60, 1991. URL `http://citeseer.nj.nec.com/buntine91theory.html`.

Robert Castelo and Tomáš Kočka. On inclusion-driven learning of Bayesian networks. *Journal of Machine Learning Research*, 4:527–574, 2003.

G. Cooper and E. Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9:309–347, 1992. URL `http://smi-web.stanford.edu/pubs/SMI_Reports/SMI-91-0355.pdf`. Appeared as 1991 Technical Report KSL-91-02 for the Knowledge Systems Laboratory, Stanford University (also SMI-91-0355).

James Cussens. Stochastic logic programs: Sampling, inference and applications. In *Proc. UAI-00*, pages 115–122, San Francisco, CA, 2000. Morgan Kaufmann. URL `ftp://ftp.cs.york.ac.uk/pub/ML_GROUP/Papers/uai00.ps.gz`.

James Cussens. Parameter estimation in stochastic logic programs. *Machine Learning*, 44 (3):245–271, 2001. URL `ftp://ftp.cs.york.ac.uk/pub/ML_GROUP/Papers/jcslpmlj.ps.gz`.

Thore Egeland, Petter Mostad, Bente Mevåg, and Margurethe Stenersen. Beyond traditional paternity and identification cases. Selecting the most probable pedigree. *Forensic Science International*, 110(1), 2000.

William Feller. *An Introduction to Probability Theory and Its Applications*, volume 1. John Wiley, New York, third edition, 1950.

G. Frege. Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens, 1879.

Nir Friedman and Daphne Koller. Being Bayesian about network structure: A Bayesian approach to structure discovery in Bayesian networks. *Machine Learning*, 50:95–126, 2003. URL `http://robotics.stanford.edu/~koller/papers/order-mcmc.ps`. Expanded version of UAI-2000 paper.

Andrew Gelman. Parameterization and Bayesian modeling. *Journal of the American Statistical Association*, 99(466):537–545, 2004.

W. R. Gilks, S. Richardson, and D. J. Spiegelhalter, editors. *Markov Chain Monte Carlo in Practice*. Chapman & Hall, 1996.

Olle Häggström. *Finite Markov Chains and Algorithmic Applications*, volume 52 of *London Mathematical Society Student Texts*. Cambridge University Press, Cambridge, 2002.

D. Heckerman, D. Geiger, and D. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20:197–243, 1995a. URL `ftp://ftp.research.microsoft.com/pub/tr/tr-94-09.ps`. Also appears as Technical Report MSR-TR-94-09, Microsoft Research, March, 1994 (revised December, 1994).

David Heckerman, David Maxwell Chickering, Christopher Meek, Robert Rounthwaite, and Carl Kadie. Dependency networks for inference, collaborative filtering, and data visualization. *Journal of Machine Learning Research*, 1:49–75, October 2000.

David Heckerman, Dan Geiger, and David M. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20(3):197–243, 1995b.

Søren Højsgaard and Bo Thiesson. BIFROST—block recursive models induced from relevant knowledge, observations, and statistical techniques. *Computational Statistics & Data Analysis*, 19:155–175, 1995.

Colin Howson and Peter Urbach. *Scientific Reasoning: The Bayesian Approach*. Open Court, La Salle, Illinois, 1989.

Mikko Koivisto and Kismat Sood. Exact Bayesian structure discovery in Bayesian networks. *Journal of Machine Learning Research*, 5:549–573, 2004.

Helge Langseth and Thomas D. Nielsen. Fusion of domain knowledge with data for structural learning in object oriented domains. *Journal of Machine Learning Research*, 4:339–368, July 2003. URL `http://www.jmlr.org/papers/volume4/langseth03a/langseth03a.pdf`.

Kathryn Blackmond Laskey and James W. Myers. Population Markov chain Monte Carlo. *Machine Learning*, 50:175–196, 2003.

S.L. Lauritzen and D.J. Spiegelhalter. Local computations with probabilities on graphical structures and their applications to expert systems. *Journal of the Royal Statistical Society A*, 50(2):157–224, 1988.

Steffen L. Lauritzen and Thomas S. Richardson. Chain graph models and their causal interpretations. *Journal of the Royal Statistical Society (B)*, 64(3):321–361, 2002.

D. Madigan and J. York. Bayesian graphical models for discrete data. *International Statistical Review*, 63:215–232, 1995.

David Madigan, Jonathan Gavrin, and Adrian E. Raftery. Eliciting prior information to enhance the predictive performance of Bayesian graphical models. *Communications in Statistics: Theory and Methods*, 24:2271–2292, 1995. URL `http://www.stat.washington.edu/tech.reports/tr270.ps`. Appeared as 1994 Technical Report 270, University of Washington.

David Madigan and Adrian E. Raftery. Model selection and accounting for model uncertainty in graphical models using Occam's window. *Journal of the American Statistical Association*, 89:1535–1546, 1994. URL `http://www.stat.washington.edu/www/research/reports/1991/tr213.ps`. First version was 1991 Technical Report 213, University of Washington.

Stephen Muggleton. Stochastic logic programs. In Luc De Raedt, editor, *Advances in Inductive Logic Programming*, volume 32 of *Frontiers in Artificial Intelligence and Applications*, pages 254–264. IOS Press, Amsterdam, 1996.

Ulf Nilsson and Jan Małuszyński. *Logic, Programming and Prolog*. John Wiley, Chichester, second edition, 1995. URL `http://www.ida.liu.se/~ulfni/lpp/`.

Matthew Richardson and Pedro Domingos. Learning with knowledge from multiple experts. In *Proceedings of the Twentieth International Conference on Machine Learning*, Washington, DC, 2003. Morgan Kaufmann. URL `http://www.cs.washington.edu/homes/pedrod/papers/mlc03.pdf`.

Christian P. Robert and Robert Casella. *Monte Carlo Statistical Methods*. Springer, New York, second edition, 2004.

Eran Segal, Dana Pe'er, Aviv Regev, Daphne Koller, and Nir Friedman. Learning module networks. *Journal of Machine Learning Research*, 6:557–588, 2005.

Nuala Sheehan and Daniel Sorensen. Graphical models for mapping continuous traits. In Peter J. Green, Nils Lid Hjort, and Sylvia Richardson, editors, *Highly Structured Stochastic Systems*, pages 382–386. Oxford University Press, Oxford, 2003.

Sampath Srinivas, Stuart Russell, and Alice M. Agogino. Automated construction of sparse Bayesian networks from unstructured probabilistic models and domain information. In Max Henrion, Ross Schachter, Laveen Kanal, and John Flemmer, editors, *Uncertainty in Artificial Intelligence: Proceedings of the Fifth Conference (UAI-1989)*, pages 295–308, New York, NY, 1990. Elsevier Science Publishing Company, Inc.

Matthew Stephens and Peter Donelly. A comparison of Bayesian methods for haplotype reconstruction from population genotype data. *American Journal of Human Genetics*, 73:1162–1169, 2003.

Robert Valdueza Castelo and Arno Siebes. Priors on network structures. Biasing the search for Bayesian networks. Technical Report INS-R9816, Centruum voor Wiskunde en Informatica (CWI), Amsterdam, The Netherlands, December 1998. URL `http://www.cwi.nl/ftp/CWIreports/INS/INS-R9816.pdf`.

V.N. Vapnik and A. Ya. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2):264–280, 1971.