Bayesian Logic Networks and the Search for Samples with Backward Simulation and Abstract Constraint Learning

Dominik Jain, Klaus von Gleissenthall, and Michael Beetz

Intelligent Autonomous Systems, Department of Informatics, Technische Universität München

Abstract. With Bayesian logic networks (BLNs), we present a practical representation formalism for statistical relational knowledge. Based on the concept of mixed networks with probabilistic and deterministic constraints, BLNs combine the probabilistic semantics of (relational) Bayesian networks with constraints in first-order logic. In practical applications, efficient inference in statistical relational models such as BLNs is a key concern. Motivated by the inherently mixed nature of models instantiated from BLNs, we investigate two novel importance sampling methods: The first combines backward simulation, i.e. sampling backward from the evidence, with systematic search, while the second explores the possibility of recording abstract constraints during the search for samples.

1 Introduction

When modelling real-world domains in the context of high-level AI applications, where both expressivity and tractability are key, we typically need to be able to cope with manifold relations concerning varying entities that are subject to uncertainty. Representation formalisms must therefore combine probabilistic semantics with ways of abstractly specifying rules that generalize across domains of relevant objects. In the field that has emerged as statistical relational learning and reasoning, a number of such formalisms have been proposed in recent years.

Among the most expressive such formalisms are Markov logic networks (MLNs), which elegantly extend first-order logic to a probabilistic setting by attaching weights to formulas [1]. The weighted formulas collectively represent a template for the construction of undirected graphical models (i.e. Markov random fields). Unfortunately, parameter learning in MLNs is an ill-posed problem [2] and approximate inference is typically expensive even for conceptually simple queries. Many alternative approaches [3,4,5] are based on Bayesian networks, which are often easier to deal with – both in terms of learning and inference. However, these formalisms tend to sacrifice expressivity[1] for tractability. Typically, one can easily express only local probabilistic constraints, while even simple relational properties required on a global level, such as the transitivity or symmetry of an uncertain relation, cannot easily be modelled. While, in theory, we are able to represent most such properties using most representation formalisms [6],

[1] We here understand *expressivity* not merely as the ability to express but rather as the ability to express *in concise terms*

we cannot, unfortunately, do so in practice without substantially increasing the complexity of the first-order model (which, notably, does not necessarily coincide with the complexity of ground models instantiated from it, however). From a knowledge engineering perspective, model simplicity and conciseness are key.

The representation formalism we describe in this work, Bayesian logic networks (BLNs), is a reasonable compromise in this regard. Its probabilistic components are based on conditional probability distribution templates (for the construction of a Bayesian network). Global constraints are supported by another model component, a template for the construction of a constraint network, in which we represent deterministic constraints using first-order logic.

Since learning can usually be done offline, especially the question of efficient inference in statistical relational models such as BLNs is a key concern. Exact methods being inapplicable in many larger, more complex models, sampling-based approaches are particularly widely used methods that allow to obtain any-time approximations. However, in models with determinism, sampling has constraint satisfaction as a subproblem and performance can be significantly limited due to the *rejection problem* (i.e. the problem of having to discard samples because they have zero probability). In this paper, we investigate two novel importance sampling methods that explicitly consider determinism: The first combines backward simulation, i.e. sampling backward from the evidence, with systematic search; the second explores the possibility of recording abstract constraints (*nogoods*) during the search for samples. The consideration of abstract constraints that are applicable to more than one context seems natural given the large number of repeated substructures typically found in instantiations of relational models.

The remainder of this work is organized as follows: In the following section, we review the fundamental graphical models. In Section 3, we provide details on Bayesian logic networks. In Section 4, we review standard sampling methods and introduce the two novel methods outlined above. We subsequently report on the performance of these methods in comparison to other methods on a variety of problem instances in Section 5. We conclude with an outlook on future work.

2 From Bayesian to Mixed Networks

Bayesian Networks. A Bayesian network is a tuple $B = \langle X, D, G, P \rangle$, where $X = \{X_1, \ldots, X_n\}$ is an ordered set of random variables, $D = \{D_1, \ldots, D_n\}$ is the corresponding set of domains, $G = \langle X, E \rangle$ is a directed acyclic graph representing a dependency structure over the variables X, and $P = \{P_1, \ldots, P_n\}$ is a set of (conditional) probability distributions with $P_i = P(X_i \mid \mathrm{Par}_{X_i})$, where Par_{X_i} denotes the set of parents of X_i in G. Let $\mathcal{X} = \mathrm{dom}(X) = \prod_i D_i$ be the set of possible worlds. B represents a probability distribution over $\mathcal{X}$ as a product of entries in P: For all $x \in \mathcal{X}$, $P(x) = \prod_i P(x_i \mid \mathrm{par}^x_{X_i})$, where $\mathrm{par}^x_{X_i}$ is the assignment of X_i's parents in x. Note that we write x as shorthand for $X = x$ and similarly for other (sets of) variables.

Mixed Networks. Mixed networks [7] extend Bayesian networks with explicit representations of deterministic constraints – simply by coupling them with constraint networks. Formally, a *mixed network* M is a pair $\langle B, R \rangle$, where $B = \langle X, D, G, P \rangle$ is a Bayesian network representing the joint probability distribution $P_B(x)$ and R is a constraint network. R is a tuple $\langle X, D, C \rangle$ where X and D are shared with B, and $C = \{C_i\}$ is a

set of constraints, each constraint C_i being a pair $\langle S_i, R_i \rangle$. $S_i \subseteq X$ is the scope of the constraint, and $R_i \subseteq \prod_{X_j \in S_i} D_j$ is a relation denoting the allowed combinations of values. R is a representation of the set of assignments satisfying all constraints, denoted as ρ. The mixed network M specifies a probability distribution over $\mathcal{X}$ as follows:

$$P_M(x) = \begin{cases} P_B(x) / \sum_{x' \in \rho} P_B(x') & \text{if } x \in \rho \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Assignments that do not satisfy the constraints in R are thus phased out and the remainder of the distribution represented by B is renormalized.

3 Bayesian Logic Networks

Bayesian logic networks (BLNs) extend the notion of mixed networks to a relational setting. Thus, a BLN can be thought of as a meta-model that represents a template for the construction of a mixed network. The model as such defines general principles that are universally applicable (akin to universal quantification in first-order logic) and that determine, for an arbitrary number of concrete entities, a concrete mixed network. The random variables appearing in a BLN are thus abstract, parametrized random variables, and the model defines which principles to apply in order to construct concrete random variables.

Formally, a BLN is a tuple $\mathcal{B} = \langle \mathcal{D}, \mathcal{F}, \mathcal{L} \rangle$, where

- $\mathcal{D} = \langle \mathcal{T}, S, E, t \rangle$ comprises the model's fundamental *declarations*. $\mathcal{T}$ is a taxonomy of types, which is represented as a directed forest (T, I), where T is the actual set of types and $I \subset T \times T$ is the *generalizes* relation (inverse is-a), i.e. $(T_i, T_j) \in I$ iff T_i is a generalization of T_j. S is a set of signatures of functions, and E is a set of (abstract) entities that are to exist in all instantiations, whose types are given by the function $t : E \to 2^T \setminus \{\emptyset\}$ which maps every entity to the non-empty subset of types $T = \{T_1, \ldots, T_{|T|}\}$ it belongs to. The function t thus induces a cover of the set of entities with sets $E_{T_i} = \{e \in E \mid T_i \in t(e)\}$. (We assume that t is consistent with $\mathcal{T}$, i.e. if $(T_i, T_j) \in I$, then $e \in E_{T_j}$ implies $e \in E_{T_i}$.)
  The set S contains the signature of every function f, defining the domain and the range of the function in terms of types, i.e.

  $$(f, (T_{i_1}, \ldots, T_{i_n}), T_r) \in S \;\Leftrightarrow\; f : E_{T_{i_1}} \times \cdots \times E_{T_{i_n}} \to E_{T_r}$$

  Logical predicates are simply Boolean functions, i.e. functions that map to $E_{T_r} = \mathbb{B}$, and we implicitly assume that the corresponding type symbol *Boolean* is always contained in T and that $\mathbb{B} = \{\mathit{True}, \mathit{False}\} \subseteq E$.
  We regard a set E_{T_r} that corresponds to a type $T_r \in T$ which appears as the return type of a function as a (fixed) *domain*, i.e. as a fixed set of entities that must be fully contained in E, whereas the extensions of other types may vary from instantiation to instantiation (and the corresponding subsets of E may even be empty).
- $\mathcal{F}$ is a set of *fragments* of conditional probability distributions. Every fragment defines a dependency of an abstract random variable $f(p_1, \ldots, p_n)$ (the *fragment variable*) on a set of other abstract random variables (the *parents*), where f is one of the functions defined in S, and the parameters $p_1, \ldots, p_n$ are either variables typed according to f's signature or entities in E belonging to the respective type. The dependencies are encoded in a *conditional probability function* (CPF), which

defines, for every setting of the parent variables (i.e. every element in the domain product of the parent variables, as specified by the ranges in the functions' signatures), a probability distribution over the elements in the domain of the fragment variable (i.e. the range of f).
Additionally, a fragment may define preconditions for its applicability, which may involve arbitrary logical statements about the parameters $p_1, \ldots, p_n$ (or parameters that can be functionally determined by these).

– The set $\mathcal{L}$ consists of formulas in first-order logic (with equality) over the functions/predicates defined in S, which represent hard deterministic dependencies. Such formulas may help us to model global constraints that cannot concisely be represented by the conditional probability fragments in the set $\mathcal{F}$.

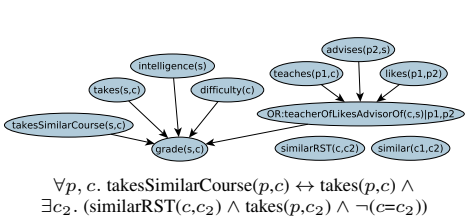
Instantiation. For any given set of entities E', whose types are given by a function $t' : E' \to 2^T \setminus \emptyset$, a Bayesian logic network $\mathcal{B}$ defines a *ground mixed network* $M_{\mathcal{B},E'} = \langle\langle X, D, G, P\rangle, \langle X, D, C\rangle\rangle$ as follows:

– E and t in $\mathcal{B}$ are augmented to include E', t'.
– The set of random variables X contains, for each function $(f, (T_{i_1}, \ldots, T_{i_n}), T_r) \in S$ and each tuple of applicable entities $(e_1, \ldots, e_n) \in E_{T_{i_1}} \times \cdots \times E_{T_{i_n}}$, one element $X_i = f(e_1, \ldots, e_n)$. The corresponding domain $D_i \in D$ is simply E_{T_r}.
– The conditional probability function $P_i \in P$ that is applicable to a random variable $X_i = f(e_1, \ldots, e_n)$ is determined by $\mathcal{F}$, which must either contain exactly one fragment for f whose preconditions are met given the actual parameters or must specify a *combining rule* [4] (e.g. noisy-or) that defines how to combine several fragments into a single conditional distribution. The connectivity of the directed graph G is such that there is a directed edge from every parent to the fragment variable – as indicated by the applicable fragments.
– For every grounding of every formula in $\mathcal{L}$ (obtained by substituting quantified variables by the applicable elements of E accordingly), the set C contains one constraint $C_i = \langle S_i, R_i\rangle$, where S_i, the scope of the constraint, is the set of random variables mentioned in the ground formula, and R_i is the relation indicating the combinations of values for which the formula is satisfied.

In the special case where $\mathcal{L} = \emptyset$, the mixed network contains no explicit constraints and is thus equivalent to the Bayesian network $\langle X, D, G, P\rangle$.

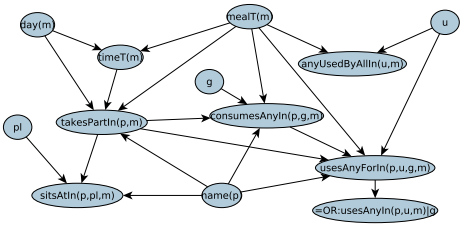
Some algorithms can essentially operate directly on mixed networks [7,8]. For purposes of inference, we may also convert any mixed network $M_{\mathcal{B},E'}$ into an *auxiliary Bayesian network* $B_{\mathcal{B},E'}$, allowing us to leverage the large body of inference methods available for Bayesian networks. $B_{\mathcal{B},E'}$ is constructed from $M_{\mathcal{B},E'}$ by adding to X for every constraint $C_i = (S_i, R_i) \in C$ a Boolean auxiliary variable A_i that represents the corresponding constraint and has as its parents in G the set of nodes S_i. The probability function associated with A_i is to return a probability of 1 as the value for *True* for every parent configuration contained in R_i and 0 otherwise. Since all constraints are required to be satisfied, when we perform inference in the auxiliary Bayesian network, we condition on the auxiliary variables, requiring that they all take on a value of *True*. Therefore, if $|C| = k$, we have

$$P_{M_{\mathcal{B},E'}}(X = x) = P_{B_{\mathcal{B},E'}}(X = x \mid A_1 = \textit{True}, \ldots, A_k = \textit{True}) \tag{2}$$



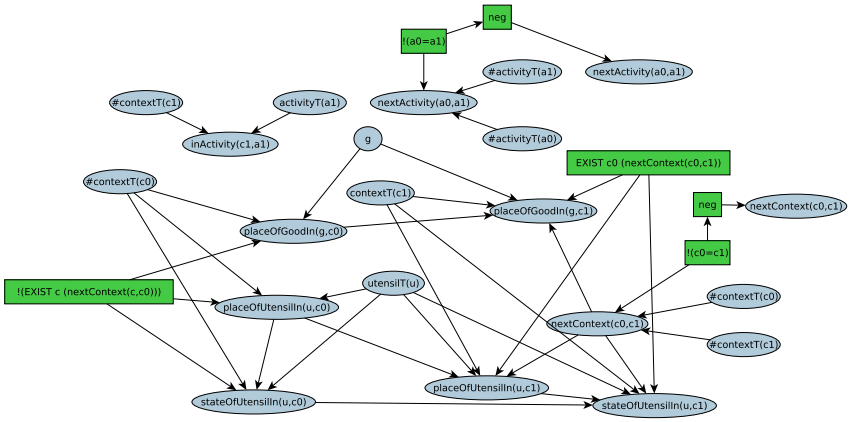
$\forall p, c.$ takesSimilarCourse(p,c) $\leftrightarrow$ takes(p,c) $\wedge$
$\exists c_2.$ (similarRST(c,c_2) $\wedge$ takes(p,c_2) $\wedge$ $\neg$(c=c_2))

(a) grades extended



$\forall\ p,p_2,pl,m.$ sitsAtIn(p,pl,m) $\wedge$ sitsAtIn(p_2,pl,m) $\rightarrow$ p=p_2

(b) meal habits



$\forall\ c,a_1,a_2.$ inActivity(c,a_1) $\wedge$ inActivity(c,a_2) $\rightarrow$ a_1=a_2

(c) sequential activities

Fig. 1: Excerpts of three BLN models: A graphical representation of the set of fragments $\mathcal{F}$ is shown (elliptical nodes are associated with conditional probability functions, rectangular nodes indicate preconditions for fragments to be applicable) along with one exemplary logical formula from the set $\mathcal{L}$ for each model. These models are used in the experiments of Section 5, where they are briefly described.

BLNs are implemented in the open-source toolbox PROBCOG. In particular, the implementation uses a graphical representation for the set of fragments $\mathcal{F}$ (see Fig. 1) and allows to conveniently augment evidence based on Prolog rules, which, given seed evidence, can compute additional pieces of evidence. For further details, we refer to the project homepage.[2]

4 Sampling Techniques

In this section, we propose inference algorithms that are specifically tailored towards applicability in probabilistic models with determinism as we obtain them by instantiating a BLN. We consider the inference task of computing the distribution of one or more query variables $Q \subset X$ given an assignment to evidence variables $E \subset X$, i.e. to compute the posterior distribution $P(Q \mid E = e)$. First, we review the fundamental sampling techniques.

[2] https://ias.in.tum.de/probcog-wiki

4.1 Fundamental Sampling Techniques

Rejection Sampling. In Bayesian networks, sampling from the prior $P(x)$ is particularly simple: If random variables are ordered topologically with respect to G, we sample from each (conditional) distribution in order to generate a full assignment to all variables in X. Let $S = \{x^{(1)}, \ldots, x^{(N)}\}$ be a set of samples taken from $P(x)$. For sufficiently large N, the relative frequency of an assignment will be an approximation of its probability, i.e. for all $x \in \mathcal{X}$, $n(S,x)/N \approx P(x)$, where $n(S,x)$ is the number of samples in S satisfying the assignment x. We can therefore compute a query as $P(q \mid e) \approx n(S, q \wedge e)/n(S,e)$. Since all samples that do not satisfy e are irrelevant to our query, they can be ignored (and are consequently rejected by the sampler).

Importance Sampling. Importance sampling [9] allows to reduce the problem of rejections by sampling from some *importance function* $Q(x)$ instead of the prior $P(x)$, the idea being that $Q(x)$ should somehow use the evidence and the structure of the model to avoid sampling irrelevant assignments. If Q is a probability distribution, the only additional constraint that needs to be placed on Q is that it must not assign zero probability to any $x \in \mathcal{X}$ for which $P(x \mid e) \neq 0$. Given a set of samples S (as above but now taken from $Q(x)$), we have that for all $x \in \mathcal{X}$, $n(S,x)/N \approx Q(x)$, and therefore $n(S,x)/N \cdot P(x)/Q(x) \approx P(x)$ for all x satisfying $E = e$. Thus a query can be computed as

$$P(q \mid e) \approx \frac{\sum_{x \in \mathcal{X}, x \models q \wedge e} n(S,x)/N \cdot P(x)/Q(x)}{\sum_{x \in \mathcal{X}, x \models e} n(S,x)/N \cdot P(x)/Q(x)} = \frac{\sum_{x^{(i)} \in S, x^{(i)} \models q \wedge e} w(x^{(i)})}{\sum_{x^{(i)} \in S, x^{(i)} \models e} w(x^{(i)})} \quad (3)$$

where $w(x) := P(x)/Q(x)$. Therefore, by assigning a weight of $P(x)/Q(x)$ to each sample, we compensate having sampled from Q instead of P and can approximate the desired posterior.

Backward Simulation. Fung and Del Favero [10] presented one way of focusing the importance function Q on the evidence. Instead of sampling forward in a topological ordering, we can sample backward from the evidence, i.e. we can assign the parents of a node X_i whose value is already given by sampling an assignment to Par_{X_i} according to the conditional distribution of X_i: We sample $\mathrm{Par}_{X_i} = \mathrm{par}_{X_i}$ with probability proportional to $P(x_i \mid \mathrm{par}_{X_i})$. The backward sampling process is repeated until all ancestors of evidence nodes have been instantiated. Any remaining nodes to which no values have been assigned are forward sampled to obtain a complete assignment to all variables.

The algorithm first determines a sampling order $\mathcal{O}$. This step partitions the set of nodes into three sets: backward sampled nodes $\mathcal{O}_B$, forward sampled nodes $\mathcal{O}_F$ and unsampled nodes $\mathcal{O}_O$ that are outside the sampling order. To compute an ordering, we manage a list of backward sampling candidates – initially the list of evidence nodes. For each candidate X_i, we determine whether any of its parents are yet uninstantiated and if so, we add it to $\mathcal{O}$ as a backward sampled node and add its parents to the list of candidates (as during sampling, X_i's value will be used to *instantiate* the previously uninstantiated parents). Otherwise the node is outside the actual sampling order and is part of $\mathcal{O}_O$. All remaining uninstantiated nodes are forward sampled (make up the set $\mathcal{O}_F$) and are added to the sampling order in topological order (a forward sampled node instantiates itself).

Given the semantics of forward and backward sampling, the importance distribution of backward simulation is

$$Q(x) = \prod_{X_i \in \mathcal{O}_B} \frac{P(X_i = x_i \mid \mathrm{par}^x_{X_i})}{Z_i} \cdot \prod_{X_i \in \mathcal{O}_F} P(X_i = x_i \mid \mathrm{par}^x_{X_i}) \tag{4}$$

where Z_i is a normalization constant. Thus sample weights are computed using

$$w(x) = \frac{P(x)}{Q(x)} = \prod_{X_i \in \mathcal{O}_B} Z_i \prod_{X_i \in \mathcal{O}_O} P(X_i = x_i \mid \mathrm{par}^x_{X_i}) \tag{5}$$

SampleSearch. Even though importance sampling techniques such as backward simulation can considerably reduce the rejection problem, there may still be too many rejections to obtain a sufficient number of samples within a reasonable time frame. To address this problem, Gogate and Dechter [11] proposed the SampleSearch scheme, which systematically searches for samples. Rather than discarding a sample as soon as its weight becomes zero and restarting, we can go back and reassign variables until we find an assignment that has a non-zero weight.

The simplest version of SampleSearch samples from the prior $P(x)$, consecutively assigning values to each random variable in topological order. If it encounters an evidence variable where the probability of the evidence is 0 given the values previously assigned to the variable's parents, we backtrack to the previous variable in the topological ordering, excluding the value that was previously assigned, renormalizing the conditional distribution and choosing a different value. Should all the values within a domain have been excluded, we backtrack further – and so on, until all variables have been assigned a value that is compatible with the evidence. Of course, more elaborate versions of SampleSearch can make use of advanced techniques known from satisfiability testing [11] or constraint satisfaction problems (e.g. arc consistency and conflict-driven backjumping [12]), and, instead of the prior $P(x)$, we could use an importance function computed via a message-passing algorithm such as loopy belief propagation.

It is important to note that the probability $Q(x)$ with which a sample is selected in SampleSearch cannot directly be used to compute weights that will yield an asymptotically unbiased estimator for the desired posterior. We may select the same sample with different probabilities depending on the exclusions we had to make in order to obtain it. In effect, the algorithm samples from the backtrack-free distribution and hence the highest possible probability (maximum number of exclusions) in each sampling step will yield an unbiased estimator (see [11] for proof). This probability can be approximated using the set of samples collected.

4.2 Backward SampleSearch

In the following, we combine the notion of backward simulation with a sample search scheme to obtain *Backward SampleSearch* (BSS), i.e. we perform backward simulation and apply backtracking whenever the sample weight becomes zero – until we find an assignment that represents a solution to the induced constraint satisfaction problem.

Since chronological backtracking is typically highly inefficient, we use conflict-directed backjumping [12] in the variant BSS-BJ. Whenever we encounter a partial assignment that cannot be completed to a full assignment, we jump back to the node with the highest index in the sampling order that has bearing on the conflict that we

encountered. With Algorithm 1, we provide pseudocode for the generation of a sample with BSS-BJ (some of the symbols used are introduced further on). Line 11 performs, for $X_i \in \mathcal{O}_B \cup \mathcal{O}_F$, either backward sampling or forward sampling (using the given exclusions and returning a new state), and, for $X_i \in \mathcal{O}_O$, it checks whether the current assignment is valid given the previously sampled parents of X_i. In sampledIndices[i], we store the index of the value (node assignment) that we sampled from the respective distribution. Lines 14-22 realize conflict-directed backjumping for the specific semantics of backward simulation. The set B_i is used to collect order indices of candidate nodes for a backjump from X_i.

In our implementation, we augment the distribution we use for backward sampling using a more informed estimate of the probability of a parent assignment. The original backward simulation algorithm goes backward "blindly" in the sense that it does not explicitly consider any information from evidence that is further up – beyond the parents of the variable X_i that is backward sampled. As a computationally simple improvement, we use, for backward sampling, a crude approximate belief $\tilde{P}(X_j = x_j)$ for parents X_j of X_i in addition to the conditional distribution $P(x_i \mid \mathrm{par}_{X_i})$. $\tilde{P}$ propagates evidence only forward. If $X_i \in E$, then $\tilde{P}(X_i = x_i) = 1$ if $E = e \models X_i = x_i$ and 0 otherwise.

Algorithm 1 SAMPLE-BSS-BJ

```
 1: state ← empty assignment
 2: i ← 0
 3: backtracking ← false
 4: while i < |X| do
 5:     X_i ← nodes[samplingOrder[i]]
 6:     if backtracking then
 7:         exclusions[i].append(sampledIndices[i])
 8:     else
 9:         exclusions[i] ← empty list
10:         B_i ← ∅
11:     sampledIndices[i], state ← sample(state, X_i, exclusions[i])
12:     if sampledIndices[i] is null then
13:         backtracking ← true
14:         if X_i ∈ O_B then
15:             B_i ← B_i ∪ {j | The j-th node in the order instantiated X_i or a node in I^{X_i}}
16:         else
17:             B_i ← B_i ∪ {j | The j-th node in the order instantiated a node in Par_{X_i}}
18:             if X_i ∈ O_O then B_i ← B_i ∪ {j | The j-th node in the order instantiated X_i}
19:         if B_i = ∅ then return "inconsistent evidence"
20:         i_prev ← i
21:         i ← max B_i
22:         B_i ← B_i ∪ B_{i_prev} \ {i}
23:     else
24:         backtracking ← false
25:         i ← i + 1
26: return state
```

If $X_i \notin E$, we compute it as

$$\tilde{P}(X_i = x_i) := \sum_{\overline{x} \in \mathrm{dom}(\overline{X})} P(X_i = x_i \mid \overline{x}, \mathrm{par}^e_{X_i}) \cdot \prod_{X_j \in \overline{X}} \tilde{P}(X_j = \overline{x}_j) \tag{6}$$

where $\overline{X} = \mathrm{Par}_{X_i} \setminus E$ and $\mathrm{par}^e_{X_i}$ is the assignment of $\mathrm{Par}_{X_i} \cap E$ in e.

When backward sampling from X_i, some of X_i's parents are yet uninstantiated while others may already be instantiated (either because they are evidence variables or because they were instantiated by a previous backward sampling step, as determined by the sampling order). Par_{X_i} is partitioned into two sets U^{X_i} and I^{X_i} accordingly. We sample the yet uninstantiated parents U^{X_i} of X_i given previous assignments via

$$Q(U^{X_i} = u^{X_i} \mid \cdot) = \frac{1}{Z_i} P(X_i = x_i \mid U^{X_i} = u^{X_i}, I^{X_i} = i^{X_i}) \prod_{X_j \in U^{X_i}} \tilde{P}(X_j = u^{X_i}_j) \tag{7}$$

where Z_i is a normalization constant and $u^{X_i}_j$ is X_j's value in u^{X_i}. Forward sampled nodes are treated as in backward simulation. Therefore, we can compute sample weights as follows:

$$w(x) = \prod_{X_i \in \mathcal{O}_B} \frac{Z_i}{\prod_{X_j \in U^{X_i}} \tilde{P}(X_j = x_j)} \prod_{X_i \in \mathcal{O}_O} P(X_i = x_i \mid \mathrm{par}^x_{X_i}) \tag{8}$$

As in SampleSearch, directly using these weights will yield a biased estimator. Fortunately, we can obtain an unbiased estimator by keeping track of the highest probability values with which each particular partial assignment was sampled (cf. [11]), which we obtain for the smallest Z_is in Eq. 7 we observed in the sampling process.

A similar sample searching method has not been previously investigated. However, the backward approach is loosely related to formula-based inference [13], as the backward sampling of a node can be viewed as treating parts of the sampling problem at a formula level – since backward sampling assigns several variables (the yet uninstantiated parents) at once according to the constraint imposed by the child's setting.

4.3 SampleSearch with Abstract Constraint Learning

When sampling, we are repeatedly faced with the same problem – generating a sample from a distribution – and when applying SampleSearch, we thus repeatedly encounter the same dead-ends and are forced to backtrack in precisely the same way. Even during the generation of a single sample, we may encounter the same local dead-end several times. In the constraint solving community, this has given rise to the notion of constraint (or *nogood*) learning [12] – the natural principle of learning from one's past mistakes. The combination of SampleSearch with constraint learning is essentially straightforward: Whenever we backtrack/backjump, we record a constraint that forbids the configuration that caused the dead-end, and whenever an assignment that is relevant to the constraint is made in the future, we check whether the constraint is applicable and if so, we add a domain exclusion. Should all domain values have been exhausted, we backtrack, recursively generating further constraints.

In instances of relational models such as BLNs, we not only encounter specific dead-ends repeatedly as described above but also dead-ends that pertain to different variables but are otherwise completely analogous. This is due to the repeated substructures found within instantiations of a relational model. Thus, the recording of abstract

constraints that allow to represent such analogous constraints in a single form suggests itself. The use of abstract constraints potentially has several advantages:

- There are fewer abstract constraints and fewer constraints are more quickly found. Therefore, the search for samples can potentially be speeded up.
- In the best case, the recording of constraints successfully trades speed for memory usage. With fewer constraints, an unrestricted recording of abstract constraints may still be feasible, while the number of propositional constraints may already be too large to fit in memory.
- Because they are not specific to a particular instantiation, abstract constraints can be saved and stored for future use in order to apply them across different instantiations of the same relational model.

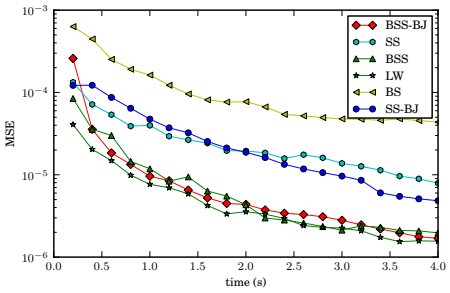
For purposes of abstraction, we consider an equivalence relation over the random variables X. In BLNs, we can uniquely identify the CPF that was generated from the fragments for any variable X_i and use this information – along with information on evidence – to define the relation. To obtain an equivalence relation that is guaranteed to be correct with respect to the domain exclusions it will later result in, we can use a colour-passing scheme analogous to the one described in [14]. In practice, simpler equivalence relations based, for example, on the CPF identifiers of all the variables in a variable's Markov blanket often suffice, because information on relevant evidence will also be represented within the constraints themselves.

We represent an abstract constraint as a graph whose nodes are equivalence classes and whose edges indicate the relationship that is to exist between the two nodes against which the edge is matched. Our relationships are based on the topology of the ground network, and we thus consider the i-th-parent relation and its inverse. To represent the actual constraint, any node in the graph may involve a check for a particular assignment of the node it is matched against. We implemented SampleSearch with abstract learning (SS-AL) based on this definition of abstract constraints.

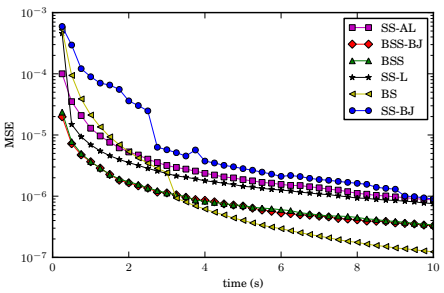
It should be clear that abstraction comes at a price. Abstract constraints are larger, because the graph representation requires internal nodes that would not appear in a propositional constraint which does not need to concern itself with relationships between variables. Moreover, checking an abstract constraint incurs an unavoidable overhead, since it requires matching a relational structure. To evaluate whether the positive or the negative aspects of abstraction prevail, we implemented SampleSearch with propositional learning (SS-L) as a point of reference.

5 Experiments

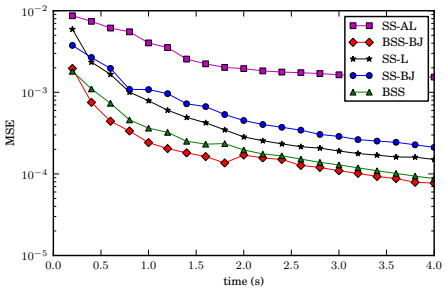
In our experiments, we compare the inference methods described above on several instantiations of BLN models (see Figures 1 and 2): The "grades" models are adaptations of the university model from [3]; the "meals habits" model captures the consumption and utensil usage habits of people during their daily meals; the "kitchen organization" model associates the spatial configuration of storage locations and devices in a kitchen with consumable objects and tools; and the "sequential activities" model is concerned with object placements and states as a result of activities of daily life being performed over time. For each model, we selected an inference task from our pool of tasks at random. In Figures 2a-2e, we plot the mean squared error in the posterior marginals of



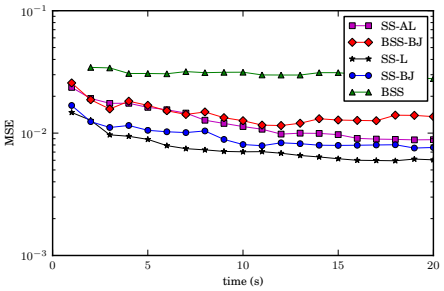
(a) grades



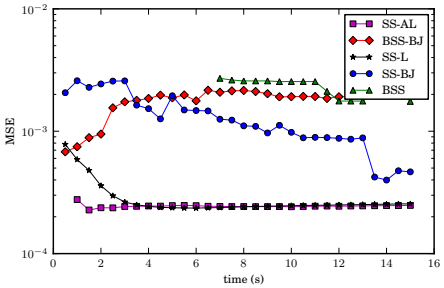
(b) grades extended



(c) meal habits



(d) kitchen organization



(e) sequential activities

	grades	grades ext.	meal habits	kitchen org.	seq. act.
#nodes	144	1200	562	125	153
N(LW)	14603	0	0	0	0
N(SS)	49760	0	0	0	0
N(SS-BJ)	56341	5085	2634	1063	6947
N(BS)	1379	31004	6	0	0
N(BSS)	7170	6883	4502	13	7
N(BSS-BJ)	9429	6586	4502	102	1292
N(SS-L)	9210	3469	2910	32648	39910
N(SS-AL)	7615	2702	827	704	10977

(f) problem sizes and #samples drawn

Fig. 2: Evaluation of the algorithms on instances of different BLNs. Mean squared errors in non-evidence variables are plotted in 2a-2e. The table 2f provides data on problem sizes and the number of samples drawn by the end of the time limit.

non-evidence variables against runtime, averaged across five trials. The algorithms used are: likelihood weighting (LW), SampleSearch with chronological backtracking (SS) and backjumping (SS-BJ), backward simulation (BS), backward SampleSearch (BSS, BSS-BJ), and learning-based search methods with backjumping (SS-L, SS-AL).[3]

We observe that, in three of our scenarios, BSS methods are among the best, clearly outperforming their forward SS counterparts in two scenarios. In the two other scenarios, the roles are reversed. However, it is the learning-based methods that perform best in these scenarios. In "sequential activities", this is due to the sequential nature of this particular model, which causes frequent backtracking that can be prevented by learning

[3] All methods were implemented in Java. For all search-based methods, asymptotically unbiased estimators were used.

constraints. The abstract learning method SS-AL never surpasses its propositional counterpart SS-L. Apparently, the overhead incurred by abstraction cannot be amortized by the reduced number of constraints in the instances we tested. Because constraint learning in general constitutes additional overhead, it is a disadvantage in cases where it is essentially unnecessary. Conceptually simple methods such as LW and BS are unable to compute approximations for any of the challenging scenarios. The quality of the samples generated by the algorithms does not seem to vary widely, as approximation quality appears to be strongly correlated with the number of samples drawn (cf. Fig. 2f).

We also experimented with a Markov chain Monte Carlo method, MC-SAT [8], which is frequently the method of choice for Markov logic networks. In most runs, we were, however, unable to draw a single sample due to the size of the SAT problems that resulted from the large number of weighted formulas that is generated for each of the problem instances.

6 Conclusion

In this work, we presented Bayesian logic networks as a practical representation language for statistical relational knowledge that bases its semantics on mixed networks. Efficient inference being a key concern in practical applications, we investigated two importance sampling techniques that explicitly address the problem of dealing with a combination of deterministic and probabilistic knowledge as it is typically found in instances of BLNs. Our initial results are encouraging and do indicate that both backward simulation and constraint learning are appropriate options for the search for samples (under the right circumstances). The recording of abstract constraints, however, incurs too much of an overhead, which cannot be compensated by gains in compactness – at least in the instances we tried. Experience in lifted satisfiability solving, however, suggests that abstraction can lead to significant speed-ups [15]. Therefore, directions for future work include finding better representations for abstract constraints that may alleviate the respective problems.

References

1. Richardson, M., Domingos, P.: Markov Logic Networks. Mach. Learn. **62** (2006) 107–136
2. Jain, D., Kirchlechner, B., Beetz, M.: Extending Markov Logic to Model Probability Distributions in Relational Domains. In: Proceedings of the 30th German Conference on Artificial Intelligence (KI-2007). (2007) 129–143
3. Getoor, L., Friedman, N., Koller, D., Pfeffer, A., Taskar, B.: Probabilistic Relational Models. In Getoor, L., Taskar, B., eds.: An Introduction to Statistical Relational Learning. MIT Press (2007)
4. Kersting, K., Raedt, L.D.: Bayesian Logic Programming: Theory and Tool. In Getoor, L., Taskar, B., eds.: Introduction to Statistical Relational Learning. MIT Press (2007)
5. Laskey, K.B.: MEBN: A Language for First-Order Bayesian Knowledge Bases. Artif. Intell. **172** (2008) 140–178
6. Jaeger, M.: Model-Theoretic Expressivity Analysis. In Raedt, L.D., Frasconi, P., Kersting, K., Muggleton, S., eds.: Probabilistic Inductive Logic Programming. Volume 4911 of LNCS, Springer (2008) 325–339
7. Mateescu, R., Dechter, R.: Mixed Deterministic and Probabilistic Networks. Ann. Math. Artif. Intel. (2008)
8. Poon, H., Domingos, P.: Sound and Efficient Inference with Probabilistic and Deterministic Dependencies. In: AAAI, AAAI Press (2006)
9. Rubinstein, R.: Simulation and the Monte Carlo Method. John Wiley & Sons, Inc. (1981)
10. Fung, R.M., Favero, B.D.: Backward Simulation in Bayesian Networks. In: UAI. (1994) 227–234
11. Gogate, V., Dechter, R.: SampleSearch: A Scheme that Searches for Consistent Samples. In: AISTATS. (2007)
12. Dechter, R., Frost, D.: Backjump-Based Backtracking for Constraint Satisfaction Problems. Artif. Intell. **136** (2002) 147–188
13. Gogate, V., Domingos, P.: Formula-Based Probabilistic Inference. In: UAI. (2010)
14. Kersting, K., Ahmadi, B., Natarajan, S.: Counting Belief Propagation. In: UAI. (2009)
15. Parkes, A.J.: Lifted Search Engines for Satisfiability. PhD thesis (1999)