Bayesian Network Automata for Modelling Unbounded Structures

James Henderson
Department of Computer Science
University of Geneva
Geneva, Switzerland
James.Henderson@unige.ch

Abstract

This paper proposes a framework which unifies graphical model theory and formal language theory through automata theory. Specifically, we propose Bayesian Network Automata (BNAs) as a formal framework for specifying graphical models of arbitrarily large structures, or equivalently, specifying probabilistic grammars in terms of graphical models. BNAs use a formal automaton to specify how to construct an arbitrarily large Bayesian Network by connecting multiple copies of a bounded Bayesian Network. Using a combination of results from graphical models and formal language theory, we show that, for a large class of automata, the complexity of inference with a BNA is bounded by the complexity of inference in the bounded Bayesian Network times the complexity of inference for the equivalent stochastic automaton. This illustrates that BNAs provide a useful framework for developing and analysing models and algorithms for structure prediction.

1 Introduction

Work in Computational Linguistics has developed increasingly sophisticated probabilistic models of language. For example, Latent Probabilistic Context-Free Grammars (LPCFGs) (Matsuzaki et al., 2005) have been developed with latent head labels (Prescher, 2005), multiple latent variables decorating each nonterminal (Musillo and Merlo, 2008), and a hierarchy of latent nonterminal subcategories (Liang et al., 2007). In this paper we propose a general framework which facilitates the specification and parsing of such complex models by exploiting graphical models to express local bounded statistical relationships, while still allowing the use of grammar formalisms to express the unbounded nature of natural language.

Graphical models were developed as a unification of probability theory and graph theory. They have proved a powerful framework for specifying and reasoning about probabilistic models. Dynamic Bayesian Networks (Ghahramani, 1998) extend this framework to models which describe arbitrarily long sequences. There has been work applying ideas from graphical models to more complex unbounded structures, such as natural language parse trees (e.g. (Henderson and Titov, 2010)), but the power of the architectures proposed for such extensions have not been formally characterised.

The formal power of systems for specifying arbitrarily large structures has been studied extensively in the area of formal language theory. Formal language theory has proved a wide range of equivalences and subsumptions between grammar formalisms, the most well known example being the Chomsky hierarchy (i.e. finite languages $<$ regular languages $<$ context-free languages $<$ context-sensitive languages $<$ recursively enumerable languages). It has also demonstrated the equivalence between formal grammars and formal automata (e.g. finite state automata generate regular languages, push-down automata generate context-free languages, Turing machines generate recursively enumerable languages). While grammar formalisms are generally more readable than automata, they appear in a wide variety of notational variants, whereas automata provide a relatively consistent framework in which different grammar formalisms can be compared. Automata also provide a clearer connection to Dynamic Bayesian Networks. For these reasons, this paper uses automata theory rather than grammars, although many of the ideas are trivially transferable to grammars.

We propose Bayesian Network Automata (BNAs) as a framework for specifying stochastic automata which generate arbitrarily large (i.e. unbounded) structures. A BNA is a standard stochastic automaton, but uses a Bayesian Network (BN) to specify its stochastic transition function. For example, the specification of a stochastic push-down automaton (PDA) requires a specification of the conditional probability distribution over *push(X)* and *pop* actions given the current state, input symbol, and top stack symbol. This distribution can be specified in a BN. While not changing the theoretical properties of the stochastic automaton, this use of BNs allows us to merge algorithms and theoretical results from BNs with those for the stochastic automaton, and thereby with those for the equivalent probabilistic grammar formalism. In the PDA example, the algorithm discussed in section 5.2 allows us to sum over all possible values for X in *push(X)* while at the same time summing over all possible parse tree structures, as is used in unsupervised grammar induction. We argue that this merging with graphical model theory simplifies the specification of more complex stochastic transition functions or grammar rules (e.g. (Musillo and Merlo, 2008)), and reduces the need for adhoc solutions to such inference problems.

BNAs can also be seen as a generalisation of Dynamic Bayesian Networks to more complex unbounded structures. BNAs simplify the specification and analysis of these more complex BNs by drawing a distinction between a finite set of statistical relationships, represented in a Bayesian Network, and the recursive nature of the unbounded structures, represented by the control mechanisms of the automaton. To illustrate the usefulness of this framework, we exploit the separation between the automaton and the Bayesian Network to provide bounds on the complexity of inference in some classes of BNAs. In particular, for a large class of grammar formalisms, the complexity of marginalising over variables in the Bayesian Network and the complexity of marginalising over structures from the automaton are independent factors in the complexity of inference with the BNA. We also exploit results from formal language theory to characterise the power of previously proposed models in terms of the generative capacity of their automata.

In the rest of this paper, we first define the framework of Bayesian Network Automata, and discuss its key properties. This provides the formal mechanisms we need to compare various Bayesian Network architectures that have been previously proposed. We then provide results on the efficiency of inference in BNAs.

2 Bayesian Network Automata

Bayesian Network Automata is a framework for specifying stochastic automata which generate unbounded structures. Formally, a BNA is simply an alternative notation for its equivalent stochastic automaton. In this section, we provide a formal definition of BNAs, without limiting ourselves to any specific class of automata.

2.1 Generating Unbounded Structures with BNAs

The purpose of a BNA is to specify a probability distribution over an infinite set of unboundedly large structures. It does this by specifying an equivalent stochastic automaton. Each complete run of the automaton generates a single structure, which is derived from the semantics of the operations performed by the automaton. The sequence of operations performed in a complete run of the automaton is called a *derivation* for its associated structure. Typically automata are designed so that derivations are isomorphic to structures, but even if multiple derivations map to the same structure, to specify a probability distribution over structures it is enough to specify a probability distribution over derivations. Stochastic automata do this in terms of a generative process for derivations, generating each operation incrementally conditioned on the derivation prefix of preceding operations.

Standard specifications of stochastic automata include specific operations acting over specific data structures. These data structures are used to represent each state of the automaton (called the *configuration*) as it proceeds through the computation. For example, the configuration of a finite state machine is a single state symbol, and the configuration of a push-down automaton is a single state symbol plus an arbitrarily deep stack of stack symbols.[1] Here we do not want to restrict attention to any specific class of automata, so the operations and data structures of BNAs are necessarily abstract (see the top half of table 1 below). For concreteness, we will illustrate the definitions with the example of push-down automata for Probabilistic Context-Free Grammars (PCFGs), illustrated

[1]Normally the configuration of an automaton includes an input tape, whose string is either accepted or rejected by the automaton. Because we are using stochastic automata as a generative process, this string is considered part of the automaton's output and therefore is not included in the configuration.

in figure 1. The left side of figure 1(a) illustrates the data structures used in a PDA's configuration.

Standard non-stochastic automata specify how to choose which operation to apply in a given configuration using a nondeterministic mapping specified in a finite transition table. The symbols input to this mapping are determined by pointers into the configuration, called *read-heads*.[2] Stochastic automata simply add a probability distribution over each nondeterministic choice of operation given read-head values. Thus, the probabilistic transition table defines the conditional distribution over operations used in each step of generating a derivation.

The central difference between BNAs and stochastic automata is that a BNA specifies its probabilistic transition table in a Bayesian Network (BN) (T in table 1). Such a BN is illustrated on the right side of figure 1(a), discussed below. To determine the probability distribution to be used for generating the next operation, the values for the BN's input variables must be set to the values pointed to by the associated read-heads in the automaton's current configuration. Since these values were themselves determined by previous operations, setting the input variable values is equivalent to equating the input variables with the relevant previous output variables from these previous operations. The resulting compound Bayesian network is illustrated in figure 1(b) as the graph of circles and arrows.

2.2 BNA Configurations

The configurations of BNAs make use of this compound BN in specifying the intermediate states of derivations. Each configuration includes a compound BN that has been constructed by instantiating copies of the transition table BN T and equating their input variables with the appropriate previous output variables. To be more precise, this compound BN g_t is iteratively constructed by taking the BN g_{t-1} from the previous configuration, setting the values $\langle b_{t-1}, w_{t-1}, v_{1t-1}, \ldots, v_{kt-1}\rangle$ chosen in the previous operation, adding a new instantiation of the transition table BN T, and equating its input variables with the appropriate variables from g_{t-1}. This construction process is performed by the *inst* function in table 1. In the example in figure 1, the BN in (b) includes four copies

[2]For consistency, we generalise the notion of read-head to include all inputs to the transition function, including the state.

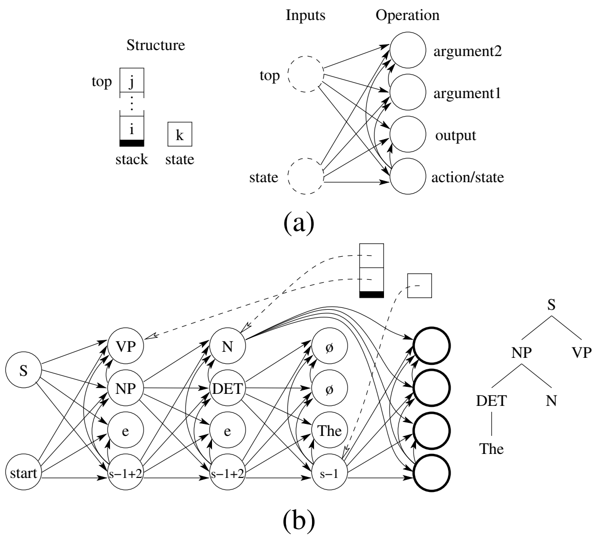

Figure 1: Illustrations of (a) a BNA specification and (b) its configuration after generating a partial tree

of the BN in (a). It includes all the instances of variables needed to specify the generation of the portion of tree shown on the right of figure 1(b), plus one set for choosing the next operation. The actions are "s-1" for popping one symbol from the stack and "s-1+2" for popping one and pushing two symbols. The pattern of edges reflects which variable instance was pointed to by the top of the stack at the time when each copy of the BN in (a) was added to the configuration's BN.

To determine which variables from g_{t-1} should be equated with each input variable in the new instantiation of T, BNA configurations include a second component that records information for the read-heads. This structural component c_t is equivalent to the data structures employed in standard automata, such as the stack of a PDA. The difference is that, instead of including symbols directly, these data structures include indices of variables in the compound BN g_t. The values of these variables are the symbols of the equivalent configuration in a standard automaton. This structural component is illustrated at the top of figure 1(b), where the indices are shown as dashed arrows. The read-heads R read indexes from c_{t-1} and pass them to the function *inst* so that it knows where to equate the associated variables from the new instantiation of T.

After constructing a new compound BN g_t, we can use it to determine the distribution over the possible next operations by looking at the output variables in the new instantiation of T. For book-

Table 1: Bayesian Network Automata specifications

Γ	a finite set of symbols
Σ	a subset of Γ which are output symbols
B	a finite set of actions
f	a halt action, $f \in B$
O	a finite set of operations $\langle b, w, v_1, \ldots, v_k \rangle$, s.t. $b \in B$, $w \in \Sigma^*$, $v_i \in \Gamma \cup \{\emptyset\}$
s_c	a start configuration structure
trans	a mapping from configuration-structure,action,$\mathbb{I}^{k+2}$ triples to configuration structures
R	a mapping from configurations to read-head vectors of indices $\mathbb{I}^r$
s_g	a start configuration BN specifying $P(b, w, v_1, \ldots, v_k)$, s.t. $\langle b, w, v_1, \ldots, v_k \rangle \in O$
s_I	The $k+2$ indices in s_g of the variables $b, w, v_1, \ldots, v_k$
T	a Bayesian Network specifying $P(b, w, v_1, \ldots, v_k \mid u_1, \ldots, u_r)$, s.t. $\langle b, w, v_1, \ldots, v_k \rangle \in O$ and $u_i \in \Gamma \cup \{\emptyset\}$
inst	a mapping from BN,BN,$\mathbb{I}^r$,$\mathbb{I}^{k+2}$ quadruples to BNs, where *inst*(T, g, H, I) instantiates T in g with $u_1, \ldots, u_r$ indexed by H and $b, w, v_1, \ldots, v_k$ indexed by I

keeping purposes, the indices I_t of these output variables are stored in a third component of a BNA configuration. These variables are shown in figure 1(b) as bold circles.

In summary, a BNA configuration consists of three components $\langle c_t, g_t, I_t \rangle$, the structure c_t, the BN g_t, and the next operation variables I_t in g_t.

2.3 BNA Specifications

A formal definition of the generative process for BNA derivations is given in figure 2, according to the definitions given in table 1. The step of stochastically generating an operation of the automaton is given in line 3, which chooses a completely specified operation from the finite set of operations, O, which the equivalent stochastic automaton can perform. These operations specify the basic action b (e.g. *push* or *pop*), any arguments to that action $v_1, \ldots, v_k$, and any string w which should be concatenated to the output of the automaton.[3] The semantics of operations is de-

```
$\langle c_0, g_0, I_0 \rangle \leftarrow \langle s_c, s_g, s_I \rangle$
For $t = 0, 1, 2, \ldots$
    Stochastically generate an operation
        $\langle b_t, w_t, v_{1t}, \ldots, v_{kt} \rangle$ from distribution
        defined by variables $I_t$ in $g_t$
    Write $w_t$ to the output
    If $b_t = f$, then halt
    $c_{t+1} \leftarrow$ trans$(c_t, b_t, I_t)$
    Deterministically generate unique $I_{t+1} \in$
        $\mathbb{I}^{k+2}$
    $g_{t+1} \leftarrow$ inst$(T, g_t, R(c_{t+1}), I_{t+1})$
    Set the values of variables $I_t$ in $g_{t+1}$ to
        $\langle b_t, w_t, v_{1t}, \ldots, v_{kt} \rangle$
```

Figure 2: Pseudo-code for generating BNA derivations

fined by the function *trans*. As indicated in figure 2, *trans*(c_t, b_t, I_t) computes the next configuration structure c_{t+1} given the previous configuration structure c_t, the operation's action b_t, and the indices I_t of the variables $b_t, w_t, v_{1t}, \ldots, v_{kt}$ which select the action and its arguments. Any complete sequence of allowable operations is a derivation.

When choosing the next operation to perform, the stochastic automaton's transition function can only look at those symbols pointed to by its read-heads, specified in R in table 1. For example, the read-heads for a PDA identify the top of the stack and the state. In a BNA, this transition function is specified in the BN T, which has as many input variables as there are read-heads (r), and as many output variables as there are terms in an operation ($k + 2$). Conditioning on the variables pointed to by the automaton's read-heads is achieved by the *inst* function. As indicated in figure 2, *inst*$(T, g_t, R(c_{t+1}), I_{t+1})$ computes the next configuration BN g_{t+1} by adding to the previous configuration BN g_t an instance of T, such that the input variables of T are equated with the variables $R(c_{t+1})$ in g_t pointed to by the read-heads, and the output variables of T are assigned the new unique indices I_{t+1}. Any other variables in T are also assigned new unique indices.

The BN T is required to have no edges whose

[3]In addition to generating the elements of a standard automaton's "input tape", this specification is notationally different from standard ones in that there is a distinction between the action b and the arguments to that action $v_1, \ldots, v_k$. As discussed below, this is to distinguish between decisions which change the structure of statistical dependencies (the actions) and decisions which only change the labels which decorate that structure (the arguments).

destination is one of the input variables, so as to prevent edges whose direction conflicts with the temporal order of the automaton's operations. This property is important for efficient inference in BNAs, as will be discussed in the next section.

3 Bounded Versus Unbounded Generalisations

Bayesian networks capture generalisations by using the same parameters for more than one pattern of variable values. By specifying how parameters are reused across different cases, they specify how to generalise from data on one pattern to previously unseen patterns. This specification is done in the BN's model structure. Because this model structure is finite, BN's can only express generalisations over a bounded number of cases.

In contrast, formal language theory captures generalisations with formalisms that can handle an infinite number of patterns. By proving that a formalism can handle an infinite number of cases for some variation, one proves that the formalism must be capable of generalising across that variation. For example, we know that regular grammars and finite state automata can generalise across positions in a string, because they can generate arbitrarily large strings despite having a finite specification. They can model a finite number of specific string positions, but there will always be a larger string position for which they have no special case, and therefore they must treat it in the same way as some other smaller string position. Proofs of this form are called pumping lemmas. Context-Free Grammars and push-down automata generalise to arbitrarily long dependencies within a string, using arbitrarily deep tree structures. Tree Adjoining Grammars generalise to arbitrarily long dependencies within these trees, using arbitrarily deep trees which themselves generate trees.

3.1 Conditionally Bounded Models

Previous work has developed formalisms for specifying graphical models that generalise to an infinite number of cases. Dynamic Bayesian Networks (DBNs) (Ghahramani, 1998), such as Hidden Markov Models, and linear-chain Conditional Random Fields (Lafferty et al., 2001) generalise to unbounded string lengths. These models specify both a template model structure which is used for each position in the string, and the model structure which connects two adjacent position in the string (or any finite window of contiguous positions). In addition, the strings are padded with start and stop symbols, with the constraint that no positions can exist beyond these symbols. These symbols act like the halt action for BNAs, and are used to ensure a proper probability distribution over sequence lengths. Given a string of a specific length, it is possible to construct the entire relevant model structure for that string, and then apply normal graphical model inference techniques to this constructed model.

If you are given the string length, it is sometimes possible to take this approach even for more complex models, such as Probabilistic Context Free Grammars. It is common practice with PCFGs to transform them into a form where the depth of the tree is bounded by the length of the string. This transformation (such as binarisation, or Chomsky normal form) does not generate the same tree structures, but it does generate the same strings. With the transformed PCFG, given the string length, it is possible to construct the whole relevant bounded-depth model structure for that string, for example using case-factor diagrams (McAllester et al., 2004), or sum-product networks (Poon and Domingos, 2011). Inference can then be done, for example, using belief propagation (Smith and Eisner, 2008).

However, this approach is limited to inference problems where the string length is known, which in turn limits the potential training methods. We cannot necessarily use this pre-construction method if we want to take a generative approach, where the string length is determined by a generative process, or if we want to do incremental interpretation, where we want to do inference on prefixes of the string without knowing to total string length. In this situation, we might need to consider an infinite amount of model structure, for all the possible string lengths. In particular, this is true for undirected graphical models, such as Conditional Random Fields, which are globally normalised (c.f. (Rohanimanesh et al., 2009)).

3.2 Model Structure Prediction

BNAs allow us to do generative, incremental inference because they use directed graphical models, in particular Bayesian Networks. Because BNs are locally normalised, it is possible to solve some inference problems independently of the infinite unconstrained portion of the model structure. For example, a Hidden Markov Model can be applied to any string prefix to infer the probability distribution over the following element of the string, without needing to know the length of the total string. In general, an inference can be done independently of the unconstrained portion of the model structure provided there are no edges directed from the unconstrained portion to the portion where information is given. More precisely, consider the set $\mathcal{G}$ of complete model structures consistent with the given visible variables V, and let G be the intersection of all the $G' \in \mathcal{G}$. If for all $G' \in \mathcal{G}$, for all $h \in (G' - G)$, and for all $v \in V$, there is no directed path from h to v, then for all $G' \in \mathcal{G}$, $P(V|G') = P(V|G)$. This follows directly from well known properties of BNs, which rely on the fact that normalising $P(V|G)$ can be done locally to G. Thus $\sum_{G'} P(G')P(V|G') = P(V|G)$, and therefore we can do inference looking only at the known portion G of the model structure.[4]

For BNAs, this means that, given a prefix of an automaton's derivation, we can determine a unique model structure which is sufficient to compute the probability distribution for the automaton's next operation. There is no need to consider all possible future derivations.

Interestingly, not all variables in the derivation prefix must be given in order to determine a finite specification of the model structure. In some formalisms, it is sufficient to know the length of the string generated by the derivation prefix. But this approach may not take full advantage of existing algorithms for doing inference with grammars or automata. On the other extreme, the approach we took in section 2 of completely specifying all variables does not take full advantage of existing algorithms for doing inference in graphical models. As we will see in section 5.2, BNAs provide a framework where these two types of algorithms can be combined in a conceptually and computationally modular way.

More specifically, BNA derivations make a distinction between variables b_t which specify the action and variables $w_t, v_{1t}, \ldots, v_{kt}$ which specify the arguments to the action. The action (e.g. *push* versus *pop*) must be sufficient to determine the structure c_t of the configuration. The configuration structure in turn determines the placement of the read-heads $R(c_t)$, which determines the model structure of the configuration's BN g_t. Therefore, the sequence of derivation actions $b_0, \ldots b_n$ is sufficient to uniquely determine the model structure of the final configuration's BN g_{n+1}.

In BNA derivations, the arguments (e.g. *push*(A) versus *push*(B)) only affect the labels in the configuration's BN g_t. We can exploit this fact by only choosing specific values for the action variables, and leaving the argument variables unspecified. The constructed BN g_t will then specify a distribution over values for the argument variables. For example, a BNA configuration for a PDA must specify a specific depth for the stack, but could specify a distribution over the symbols in each position on a stack of that depth.

To illustrate this distinction between actions and arguments, consider an alternative two-pass generative process for BNA derivations. In the first pass of the generative process, it chooses the sequence of actions $b_0, \ldots b_n$ for a derivation, which predicts the model structure of the final configuration's BN g_{n+1}. The probability distributions for generating this sequence is defined by marginalising out the values for the argument variables. The probability distribution over argument variables defined by the final configuration's BN g_{n+1} then defines a distribution over argument variable values. In the second pass of the generative process, the argument values are chosen according to this distribution. This two-pass generative process will generate the same distributions over derivations as the characterisation in section 2.3.

4 Related Architectures and Grammars

In this section we discuss how BNAs are related to some previous proposals for the probabilistic modelling of unbounded structures.

Dynamic Bayesian Networks extend BNs to arbitrarily long sequences. A DBN specifies a bounded BN that models one position in the sequence, including input and output variables. A new instance of this BN is created for each position in the given sequence, with the outputs for one position's BN equated with the inputs of the subsequent position's BN. DBNs are equivalent to BNAs with finite state automata. The bounded BN of the DBN corresponds to the T of the BNA. The DBN has no need for an explicit representation of the BNA's configuration structure because

[4]This argument for directed models was recognised previously in (Titov and Henderson, 2007). Garg and Henderson (2011) proposes a model which mixes directed and undirected edges, but which still has this property because undirected edges are all local to individual derivation steps.

the configuration of a finite state automaton consists only of a state variable with a bounded number of values, so all information about the configuration can be encoded in the bounded BN. The only structure-modifying actions which are necessary are *continue* versus *stop*, which are used to generate the length of the sequence.

Switching DBNs (Ghahramani, 1998; Murphy, 2002) are DBNs which include variables that switch between multiple BN models during each position in the sequence. They are also covered by finite state BNAs, by the same argument. Although different BN models may be chosen for different positions, there are only a bounded number of possible BNs, so they can all be included in a single T. The switching decisions must be encoded in the structure-modifying actions.

The previous work on graphical models which is closest to our proposal is Incremental Sigmoid Belief Networks (ISBNs) (Henderson and Titov, 2010). Sigmoid Belief Networks (SBNs) (Neal, 1992) are a type of Bayesian Network, and ISBNs use the same technique as here for modelling arbitrarily large structures, namely incrementally constructing an SBN that generates the derivation of the structure. Henderson and Titov (2010) used this framework to reinterpret previous work on neural network parsing (Henderson, 2003) as an approximation to inference in ISBNs, and proposed another approximate inference method for natural language parsing. However, the control mechanism needed to construct an SBN for a derivation was not formalised.

Without any formal specification of how to construct an SBN for a derivation, it is hard to determine the power of the ISBN architecture. The specific ISBN models developed for natural language parsing cannot be expressed with a finite state controller, and thus they are not DBNs. They require at least a push-down automaton. The derivations modelled in (Henderson and Titov, 2010) are predictive LR derivations, which are equivalent in power to context-free derivations. However, the statistical dependencies which are specified as edges in their constructed SBN are not restricted to the context-free structure of the derivations. Edges may refer to any "structurally local" variables decorating the derived tree, even if they are not immediately local (such as the leftmost sibling). To translate such a model into a BNA, the read-head R would have to have access to symbols which are not on the top of the stack, and therefore the derivation structure would not be context-free. Given the restriction of locality, it is probably possible to devise chains of variables which pass the necessary information through the context-free structure of the tree. But without such a transformation, our analysis of inference algorithms in the next section suggests that it would be difficult to devise efficient algorithms for exact inference with this model. Henderson and Titov (2010) avoid this question by only considering approximate inference methods, since inference in the bounded SBN of their model is already intractable even before combining it with a parsing algorithm.

Koller et al. (1997) propose a framework for specifying complex recursive probabilistic models in terms of a stochastic functional programming language. As with BNAs, these models are locally normalised. They propose an inference algorithm for this framework, and discuss Bayesian networks and context-free grammars as examples of what can be implemented in it. The use of a functional programming language suggests a close relationship to the context-free derivation structures discussed in the next section.

From the grammar formalism side, the model that is closest to our proposal is Latent Probabilistic Context-Free Grammars (LPCFGs) (Matsuzaki et al., 2005). LPCFGs are PCFGs where the nonterminal labels are augmented with a latent variable. There has been much work recently on different training methods and different restrictions to the pattern of values which the latent variables are allowed to take (e.g. (Matsuzaki et al., 2005; Prescher, 2005; Petrov et al., 2006; Musillo and Merlo, 2008)). LPCFGs can be modelled as BNAs with push-down automata. The bounded BN of the equivalent BNA includes both a variable for the visible nonterminal label of the LPCFG and a variable for the latent extension of the nonterminal label. As a more specific example, the latent head labels of (Prescher, 2005) could be specified using a BN with a switching variable that selects which child's head variable is propagated to the head variable of the parent. Musillo and Merlo (2008) extend LPCFGs to include multiple latent variables decorating each nonterminal, with linguistically motivated constraints on how they can be related to each other. The BN of BNAs would provide a more perspicuous method to express these constraints.

5 Inference in Bayesian Network Automata

To illustrate the usefulness of the BNA framework, we look at algorithms for exact inference in BNAs. The way in which BNAs allow us to separate issues of structure from issues of labelling greatly facilitates the analysis and design of algorithms for calculating probabilities and exploring the space of derivations. In many important cases, previous algorithms for grammars can be combined directly with previous algorithms for Bayesian Networks, and the resulting algorithm has a complexity where the complexity of its two components are independent factors.

A BNA is a specification of a generative probabilistic model, and as such can be used to answer many questions about the probability distributions for some variables given others. We might want to find the most probable derivation given some sequence of output symbols (as in statistical parsing, or decoding), to marginalise over the derivations (as in language modelling), or to find the most probable values for a subset of the variables while marginalising over others (as in latent variable models of parsing). Although in some ways the most interesting, this last class of problems is in general NP-hard. Even for models based on finite state automata (e.g. HMMs), mixing the maximisation of some probabilities with the sum over others results is an NP-hard problem (Lyngsø and Pedersen, 2002). Because for reasons of space we are limiting our discussion to exact inference, we therefore also limit our discussion to the first two types of problems. However, the arguments given in section 5.2 for marginalising over all the unknown variables can also be applied to many approximations to latent variable models of parsing.

5.1 Fully-Specified BN Computation

The first case we consider is when the space of inputs and operations of the BN T is small enough that it is feasible to compute the complete conditional probability distribution of all possible operations given each possible input. In this case, the conditional probability distribution can be precomputed and used to fill the transition table for the equivalent stochastic automaton. Then any standard algorithm for the stochastic automaton can be applied. Under this assumption, if the complexity of inferring the complete conditional probability distribution for T is $O(F_B)$, and the complexity of the standard algorithm is $O(F_A)$, then the complexity of the complete problem is $O(F_B + F_A)$. In other words, computation time is dominated by whichever of these complexities is worse. For most previously proposed statistical parsing models, the parsing complexity does dominate and this pre-compilation strategy is in effect adopted. However, the interest of the BNA framework is in allowing the specification of more complicated models, where inference in the BN T is not so simple.

The second case we consider also uses any standard algorithm for the stochastic automaton, but computes probabilities with the BN T on-line during the running of the algorithm. This strategy may require forward inference of the distribution over operations given a specific input, or backward inference of the distribution over inputs given a specific operation, but in general it avoids the need to compute the complete joint distribution over both. In this case, if the complexity of the necessary inference in T is $O(F'_B)$, and the complexity of the standard algorithm is $O(F_A)$, then the complexity of the complete problem is $O(F'_B F_A)$. This is a good strategy if $O(F'_B)$ is much smaller than $O(F_B)$ from the previous case, as might be the case for lexicalised models, where $O(F_B)$ is a function of the vocabulary size but $O(F'_B)$ is only a function of the number of words in the sentence.

5.2 Marginalisation in the BN

Both of the above cases require computing probabilities given completely-specified values for all the inputs and/or all the outputs of the BN T. To fully exploit previous work on inference in BNs, we would like to develop inference algorithms which work directly with probability distributions over values for both the inputs and outputs of T. A full treatment of such algorithms is beyond the scope of this paper, but here we provide some results on cases where inference algorithms for BNs can be extended to BNAs. In particular, we look at what classes of automata can be combined with belief propagation (Pearl, 1988).

Belief propagation is a general method for efficiently computing marginal probabilities in Bayesian Networks (i.e. summing over latent variables). Provided that the BN has a tree structure (ignoring the directionality of the edges), this algorithm is guaranteed to stop with the exact marginal probability in time linear in the size of the BN. Therefore, if we can guarantee that the configuration BN g_t constructed by a BNA is always a tree, then we can apply belief propagation to inference in g_t, or in any sub-graph of g_t.

The structure of g_t reflects the structure of conditioning allowed by the derivations of the automaton. In formal language theory, this is called the derivation structure. If necessary, we can abstract away from any undirected cycles that might be introduced by the specific form of the BN T by collapsing all the non-input non-action variables in T into a single variable. By using this collapsed version of T to construct g_t, the structure of g_t becomes isomorphic to the derivation structure.[5] Thus, we can guarantee that (a version of) g_t will have a tree structure if the derivation structures of the automaton are trees, or more precisely, if the automaton has context-free derivation structures.

The most important class of grammar formalisms which have context-free derivation structures is Linear Context-Free Rewriting Systems (LCFRS) (Weir, 1988). In addition to Context-Free Grammars, popular examples of LCFRS include Synchronous Context-Free Grammars and Tree Adjoining Grammars (Joshi, 1987). This latter grammar formalism can specify classes of languages much larger than context-free languages, and Synchronous CFGs express languages over pairs of strings. For our purposes, the observations can be of any form and the formalism can have any method for combining the observations generated by sub-derivations, provided the derivation structures are context-free and there exists an algorithm for marginalising over the derivation structures given an observation.

Given an automaton whose derivations have a context-free tree structure, we can apply belief propagation to compute the marginal probability for any derivation generated by that automaton (if necessary collapsing the non-input non-action variables in T into a single variable). This will be feasible if it is feasible to propagate beliefs through one instance of T. This requirement is different from the one in the previous subsection in that the given information is a probability distribution (the belief), whereas above it was assumed to be completely specified values. So, assuming that it is feasible to propagate beliefs through T and given an automaton with context-free derivations, we can feasibly compute the marginal probability of any derivation using belief propagation.

$$\begin{aligned}
&In(i-1,i,a) = P(a \Rightarrow w_i|a)\\
&In(i,k,a) =\\
&\quad \sum_{j=i+1}^{k-1} \sum_{b,c} P(a \Rightarrow bc|a) In(i,j,b) In(j,k,c)\\
&Out(0,|w|,c) = \text{ 1 if } c = S;\ \text{ 0 otherwise}\\
&Out(i,k,c) =\\
&\quad \sum_{j=0}^{i-1} \sum_{a,b} Out(j,k,a) In(j,i,b) P(a \Rightarrow bc|a) +\\
&\quad \sum_{j=k+1}^{|w|} \sum_{a,b} Out(i,j,a) P(a \Rightarrow cb|a) In(k,j,b)
\end{aligned}$$

Figure 3: The Inside-Outside algorithm for PCFGs, where a, b, c are symbols, i, j, k are indices in string w, and $a \Rightarrow bc$, $a \Rightarrow w_i$ are CFG rules

So far in this subsection we have only considered inference for a given derivation structure. In general, inference in BNAs requires marginalising both over labellings and over the structure itself, as is commonly required for unsupervised or partially-supervised grammar induction. Marginalising over structures can be done with the inside-outside algorithm for Context-Free Grammars (Baker, 1979) (given in figure 3) or the inside-outside algorithm for Tree-Adjoining Grammars (Schabes, 1992). These are dynamic programming algorithms. The inside calculations are very similar to bottom-up parsing algorithms such as CKY (Younger, 1967), except they compute sums of probabilities instead of taking the maximum probability. The outside calculations are also done in a single pass, but top-down.

Both inside and outside calculations work by incrementally considering equivalence classes of increasingly large sub-graphs of the derivation structure. For example in the equations in figure 3, each step computes a sum over the various ways that an operation can be used to construct a larger sub-derivation out of smaller ones, such that they all have the same start i, end k and label. As discussed at the end of section 3.2, with BNAs we can easily represent $In(i,k,\cdot)$ and $Out(i,k,\cdot)$ as distributions over labels, i.e. the beliefs. The sums over symbols in these equa-

[5]Note that all action variables $b_{t'}$ in g_t will have known values, because this is necessary in order to know the model structure. Thus, the concern here is the structure of the interdependencies between the possibly-unknown variables $w_{t'}, v_{1t'}, \ldots, v_{kt'}$ in g_t, because these are the variables we need to marginalise over using belief propagation.

tions ($\sum_{b,c} P(a \Rightarrow bc|a)In(i,j,b)In(j,k,c)$ and $\sum_{a,b} Out(i,j,a)P(a \Rightarrow cb|a)In(k,j,b)$) can then be done using belief propagation through the BN representation of $P(a \Rightarrow cb|a)$.

If the configuration BN g_t is a tree, then any connected sub-graph of g_t will also be a tree, so we can apply belief propagation to the sub-graph of g_t for any of the sub-derivations. In addition, belief propagation can be applied either forward or backward through the edges of g_t, so the probability distribution for any node in a sub-graph of g_t can be computed in a single pass over the sub-graph by propagating beliefs towards the desired node. This allows belief propagation to be integrated into an inside-outside algorithm; whenever the inside-outside algorithm considers building larger sub-derivations out of smaller ones, belief propagation is applied to continue the propagation of beliefs out of the smaller sub-derivations' graphs and into the instance of T for the operation used in the combination. This belief propagation can then be interleaved with the sums over structural alternatives (the sums over j in figure 3).

Because each use of an operation considered by the inside-outside algorithm corresponds to belief propagation through a single instance of T, the time complexity of performing belief propagation through T is a constant factor in the complexity of the entire algorithm. For the inside part of the algorithms, only backward inference through T is required, and then for the outside computations forward inference is required. Thus, given an $O(F_A)$ inside-outside inference algorithm and an $O(F''_B)$ inference algorithm for propagating beliefs (forward or backward) through T, the complexity of parsing will be $O(F''_B F_A)$. As with the results from the previous subsection, this complexity result factors the two components of the algorithm.

6 Conclusions

In this article we have proposed a framework for specifying and reasoning with complex probabilistic models of unbounded structures, called Bayesian Network Automata. The BNA framework combines the theory of graphical models with automata theory and formal language theory. It uses Bayesian Networks to provide a perspicuous representation of local statistical generalisations. It uses automata to provide a precise specification of how to generalise to arbitrarily large model structures, even when the model structure must be predicted during inference. Together they provide a precise and perspicuous representation of probability distributions over unbounded structures.

Using this framework, we have clarified the power of various previously proposed probabilistic models of unbounded structures. Without any additional control structure, Dynamic Bayesian Networks are equivalent to BNAs with finite state automata. This limited power also applies to switching DBNs. Incremental Sigmoid Belief Networks potentially have greater power, but previous work did not formally characterise the nature of the additional control structure which they employ. The model of parsing which has been proposed for ISBNs appears to be a BNA with a push-down automaton, but the nature of the BN used prevents direct application of the efficient parsing methods we have discussed.

We have also used this framework to explore how the complexity of inference with the bounded Bayesian Network interacts with the complexity of inference with the automaton. We have shown that for a large class of automata (which can generate more than just context-free languages), the complexity of inference with a BNA is simply the multiplication of the complexity of inference in the Bayesian Network times the complexity of inference with the automaton.

BNAs have the potential to greatly expand the class of problems which we can effectively model with graphical models, through their simple mechanism for increasing the power of these models and the large body of existing theory and algorithms that help us limit this power in ways that retain tractability. They provide a useful framework for future work on many issues, including approximate inference methods that interface well with parsing algorithms. We believe that the BNA framework will be most useful with large Bayesian Networks where only approximate inference methods are tractable.

Acknowledgements

The author would like to thank the many reviewers that have contributed to making this a better paper. This work was partly funded by European Community FP7 grant 216594 (CLASSiC, www.classic-project.org) and Swiss NSF grant 200021 125137.

References

J. K. Baker. 1979. Trainable grammars for speech recognition. *The Journal of the Acoustical Society of America*, 65(S1):S132–S132.

Nikhil Garg and James Henderson. 2011. Temporal restricted boltzmann machines for dependency parsing. In *Proc. 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 11–17.

Zoubin Ghahramani. 1998. Learning dynamic bayesian networks. In C. Giles and M. Gori, editors, *Adaptive Processing of Sequences and Data Structures*, pages 168–197. Springer-Verlag, Berlin, Germany.

James Henderson and Ivan Titov. 2010. Incremental sigmoid belief networks for grammar learning. *Journal of Machine Learning Research*, 11(Dec):3541–3570.

James Henderson. 2003. Inducing history representations for broad coverage statistical parsing. In *Proc. joint meeting of North American Chapter of the Association for Computational Linguistics and the Human Language Technology Conf.*, pages 103–110.

Aravind K. Joshi. 1987. An introduction to tree adjoining grammars. In Alexis Manaster-Ramer, editor, *Mathematics of Language*. John Benjamins, Amsterdam.

D. Koller, D. McAllester, and A. Pfeffer. 1997. Effective Bayesian inference for stochastic programs. In *Proc. 14th National Conf. on Artificial Intelligence (AAAI 1997)*, pages 740–747.

John Lafferty, Andrew McCallum, and Fernando Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. 18th International Conf. on Machine Learning*, pages 282–289. Morgan Kaufmann, San Francisco, CA.

Percy Liang, Slav Petrov, Michael Jordan, and Dan Klein. 2007. The infinite PCFG using hierarchical dirichlet processes. In *Joint Conf. on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 688–697, Prague, Czech Republic.

Rune B. Lyngsø and Christian N.S. Pedersen. 2002. The consensus string problem and the complexity of comparing hidden markov models. *Journal of Computer and System Sciences*, 65:545–569.

Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. 2005. Probabilistic CFG with latent annotations. In *Proc. 43rd Annual Meeting on Association for Computational Linguistics*, pages 75–82.

David McAllester, Michael Collins, and Fernando Pereira. 2004. Case-factor diagrams for structured probabilistic modeling. In *Proc. 20th Conf. on Uncertainty in Artificial Intelligence*, pages 382–391.

Kevin P. Murphy. 2002. *Dynamic Belief Networks: Representation, Inference and Learning*. Ph.D. thesis, University of California, Berkeley, CA.

Gabriele Musillo and Paola Merlo. 2008. Unlexicalised hidden variable models of split dependency grammars. In *Proc. 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 213–216.

Radford Neal. 1992. Connectionist learning of belief networks. *Artificial Intelligence*, 56:71–113.

Judea Pearl. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann.

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. 2006. Learning accurate, compact, and interpretable tree annotation. In *Proc. 44th annual meeting of the ACL and the 21st Int. Conf. on Computational Linguistics*, pages 433–440.

Hoifung Poon and Pedro Domingos. 2011. Sum-product networks: A new deep architecture. In *Proc. 12th Conf. on Uncertainty in Artificial Intelligence*, pages 337–346.

Detlef Prescher. 2005. Head-driven PCFGs with latent-head statistics. In *Proc. 9th Int. Workshop on Parsing Technologies*, pages 115–124.

Khashayar Rohanimanesh, Michael Wick, and Andrew McCallum. 2009. Inference and learning in large factor graphs with adaptive proposal distributions and a rank-based objective. Technical Report UM-CS-2009-008, University of Massachusetts.

Yves Schabes. 1992. Stochastic Tree-Adjoining Grammars. In *Proc. workshop on Speech and Natural Language*, pages 140–145.

David A. Smith and Jason Eisner. 2008. Dependency parsing by belief propagation. In *Proc. 2008 Conference on Empirical Methods in Natural Language Processing*, pages 145–156.

Ivan Titov and James Henderson. 2007. Incremental bayesian networks for structure prediction. In *Proc. 24th International Conf. on Machine Learning*, pages 887–894.

David Weir. 1988. *Characterizing Mildly Context-Sensitive Grammar Formalisms*. Ph.D. thesis, University of Pennsylvania, Philadelphia, PA.

Daniel H. Younger. 1967. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10(2):189–208.