Bayesian Network Structures and Inference Techniques for Automatic Speech Recognition

Geoffrey Zweig
IBM T. J. Watson Research Center, Yorktown Heights, NY, 10598
gzweig@us.ibm.com

Abstract

This paper describes the theory and implementation of Bayesian networks in the context of automatic speech recognition. Bayesian networks provide a succinct and expressive graphical language for factoring joint probability distributions, and we begin by presenting the structures that are appropriate for doing speech recognition training and decoding. This approach is notable because it expresses all the details of a speech recognition system - ranging from pronunciation dictionaries to language models - in a uniform way using only the concepts of random variables and conditional probabilities. A powerful set of computational routines complements the representational utility of Bayesian networks, and the second part of this paper describes these algorithms in detail. We present a novel view of inference in general networks - where inference is done via a change-of-variables that renders the network tree-structured and amenable to a very simple form of inference. We present the technique in terms of straightforward dynamic programming recursions analogous to HMM alpha-beta computation, and then extend it to handle deterministic constraints amongst variables in an extremely efficient manner. The paper concludes with a sequence of experimental results that show the range of effects that can be modeled, and that significant reductions in error-rate can be expected from intelligently factored state representations.

1 Introduction

Over the years, state-of-the-art speech recognition systems have grown in sophistication as code is added that addresses more and more of the phenomena that characterize natural speech in realistic environments. These phenomena include gender and age differences, pronunciation variability, differences in articulation, microphone and channel variability, and ambient noise. In some cases, current techniques address the underlying issues fairly directly, as in vocal-tract length normalization (VTLN) (Zhan & Waibel 1997), which attempts to compensate for variability in vocal tract length, or parallel model combination PMC (Varga & Moore 1990; Gales & Young 1992), which builds separate noise and speech models to separate out the effects of ambient noise. In other cases, a technique will address a wide and undefined range of phenomena simultaneously, as with likelihood-increasing transforms such as Maximum Likelihood Linear Regression (MLLR) (Leggeter & Woodland 1995), feature-space MLLR (FMLLR) (Gales 1998), or discriminant transforms such as LDA. Typically, systems that incorporate the full range of methods are quite effective, as in current broadcast news (Eide *et al.* 2000) or Switchboard (Hain *et al.* 2000) systems, but also significantly complex because the techniques were added in an evolutionary fashion over a long period of time.

The Bayesian network formalism offers an interesting and timely set of methods for addressing many of these issues in a highly systematic and unified way. The main idea of Bayesian networks as applied to speech recognition is to use a graphical structure to represent a concrete yet arbitrary set of variables within each time frame, and to specify an arbitrary factorization of the joint probability distribution over those variables. Because arbitrary sets of variables can be modeled, the formalism is ideally suited for representing detailed models of speech production and perception such as PMC, VTLN, probabilistic state classification (Luo & Jelinek 1999), and articulatory models (Deng & Erler 1992; Erler & Deng 1993; Zweig 1998; Richardson *et al.* 2000).

The application of Bayesian networks to speech recognition is best understood in terms of the broad categories of representational and computational power. As we will see in the following sections, what makes the formalism so unifying is that it reduces all phenomena to the common language of random variables and conditional probabilities. Hand-in-hand with this language is a collection of algorithms for computing with models described in the language. Once the algorithmic apparatus is in place, the same language is used to express

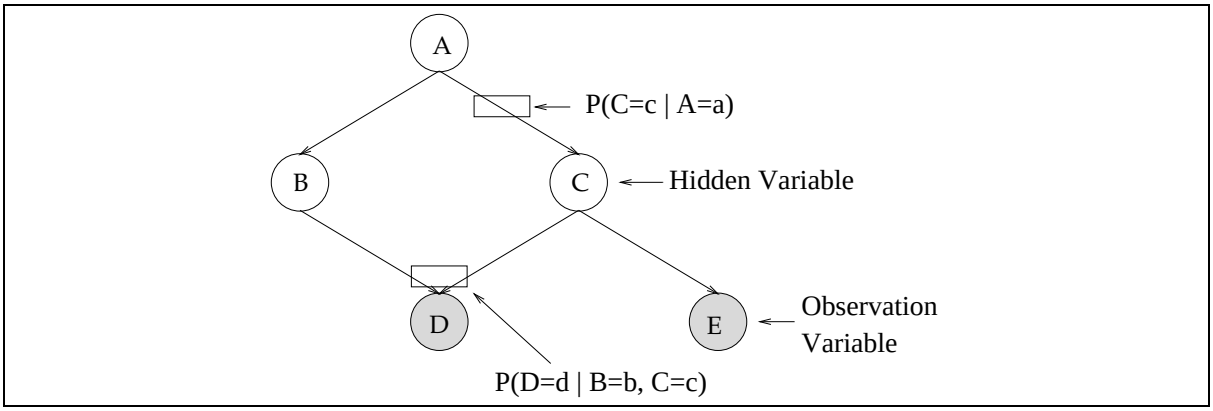

Figure 1: A Bayesian network with five variables. Variables with know values are shaded. Conditional probability functions (indicated by boxes) are associated with each variable and used to return numerical values for conditional probabilities.

everything from pronunciation dictionaries and trigram language models (Zweig 1998) to sparse Gaussian covariances (Bilmes 2000); and the same code is used to optimize the parameters for these seemingly disparate models, and to decode with them.

1.1 Bayesian Networks as Representational Tools

A Bayesian network specifies a set of variables and a factorization of the joint probability distribution in a very straightforward way. The network is defined by a directed acyclic graph or DAG, in which the nodes represent variables and the edges induce a factorization of the joint probability distribution. Specifically, let the variables in the network be denoted by capital letters, e.g. X_i, and specific values by lower case letters, e.g. x_i. Further, suppose that the immediate predecessors or parents of a variable X_i are $\mathbf{X}_{\pi(i)}$ and that the values of the parents are $\mathbf{x}_{\pi(i)}$. Then the joint distribution is factored as:

$$P(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n) = \prod_i P(X_i = x_i | \mathbf{X}_{\pi(i)} = \mathbf{x}_{\pi(i)})$$

In general, we will refer to a variable and its immediate predecessors as a family **f** in the graph. Figure 1 illustrates a simple Bayesian network over five variables. As indicated by the graph topology, the joint distribution is given by

$$\begin{aligned} &P(A = a, B = b, C = c, D = d, E = e) = \\ &\quad P(A = a)P(B = b|A = a)P(C = c|A = a)P(D = d|B = b, C = c)P(E = e|C = c) \end{aligned}$$

Figure 1 also introduces the distinction between observed variables or "observations", whose value is unambiguously know (and which are shaded in the figure), and hidden variables whose values are unknown. We will refer to the set of hidden variables as **h** and observed variables as **o**. In the case that hidden variables are present, one may still reason in concrete terms by assuming various assignments of values to the hidden variables to obtain concrete data cases. The concept of concrete variable assignments is quite important, as it is the key to understanding the inference algorithms. As we shall see, decoding speech utterances will proceed by finding the single likeliest assignment to all the hidden variables in a network, and training will involve summing over all possible variable assignments.

In order to get numerical values for specific assignments of values to the variables, it is necessary to define concrete conditional probability functions, and here again there is a wide degree of flexibility. Some of the typical functions include tabular representations for discrete variables, and Gaussian distributions for real-valued variables. The set of operations that can be done with a graph depends on the types of variables in the graph, and the conditional probability functions that are used. In particular, the presence of hidden real-valued variables significantly complicates the associated algorithms and in some cases makes them computationally intractable. However, in speech recognition, continuous variables most commonly occur as observations, and in this paper we will assume that continuous variables are always observed. In this case, the structure of the inference algorithms is unchanged from the simplest fully-discrete case.

1.2 Bayesian Networks as Computational Tools

To be truly useful, it must be possible to compute with a representational framework, and here Bayesian networks are powered by a set of very general and yet efficient algorithms for computing the quantities that are relevant to speech recognition. Specifically, there are three main computational tasks that can be performed:

1. Computation of the probability of the observed variable values: $P(\mathbf{o}) = \sum_{\mathbf{h}} P(\mathbf{o}, \mathbf{h})$
2. Computation of the likeliest hidden variable values: $argmax_{\mathbf{h}} P(\mathbf{o}, \mathbf{h})$
3. Parameter estimation from a set of observations $\{\mathbf{o}_k\}$ via EM: $argmax_{\theta} \prod_k P(\mathbf{o}_k|\theta)$

‘

With the assumption of observed continuous variables, the process of finding the single likeliest assignment does not depend at all on the form of the conditional probability functions that is used. Similarly, the process of summing over all possible hidden variable assignments to obtain marginal probabilities is insensitive to the local distributions; however, in order to use the the parameter optimization routines, it must be possible to define an EM update rule for conditional probabilities represented in that form.

1.3 Organization

The remainder of this paper will focus in turn on the representational and computational aspects of Bayesian networks. Section 2 focuses on the representational, and presents the Bayesian network structures that are necessary to represent all the components of a typical speech recognizer. In contrast to other comparisons of Bayesian networks with Hidden Markov Models, e.g. (Smyth *et al.* 1996), we make explicit all the detail required to capture the a-priori knowledge, such as baseform sequences and parameter tying, that is present in recognition systems. Section 3 shifts to the computational, and presents the inference algorithms that enable training and decoding. In contrast to other descriptions of inference, that of Section 3 is tailored to efficiently handling speech recognition structures (which are characterized by deterministic relationships between many of the variables), and cast in terms of dynamic programming recursions that are direct extensions of the HMM recursions. Section 4 presents a set of experimental results for a Bayesian network system applied to an isolated-word recognition task, and we summarize in Section 5.

2 Bayesian Network Structures for ASR

2.1 HMMs and Bayesian Networks

Since the most commonly used model for speech recognition is the hidden Markov model (HMM) (Baker 1975; Jelinek 1997; Rabiner & Juang 1993), we begin by relating HMMs to Bayesian networks. It has long been realized that the HMM is a special case of the more general class of dynamic Bayesian networks (Smyth *et al.* 1996; Pearl 1988), and Figure 2 illustrates the representation of the simplest possible HMM.

Recall that in the classical definition (Rabiner & Juang 1993), an HMM consists of:

- The specification of a number of states
- An initial state distribution π
- A state transition matrix A, where A_{ij} is the probability of transitioning from state i to j between successive observations
- An observation function $b(i, t)$ that specifies the probability of seeing the observed acoustics at time t given that the system is in state i.

In this formulation, the joint probability of a state sequence $s_1, s_2, \ldots s_T$ and observation sequence $o_1, o_2, \ldots o_T$ is given by $\pi_{s_1} \prod_{i=1}^{T-1} A_{s_i s_{i+1}} \prod_{i=1}^{T} b(s_i, i)$. In the case that the state sequence or alignment is not known, the marginal probability of the observations can still be computed, either by enumerating all possible state sequences and summing the corresponding joint probabilities, or via dynamic programming recursions. Similarly, the single likeliest state sequence can be computed.

Figure 2 shows the simplest Bayesian network representation of an HMM. It is a segmented model, in which each time frame has two variables: one whose value represents the value of the state at that time, and one

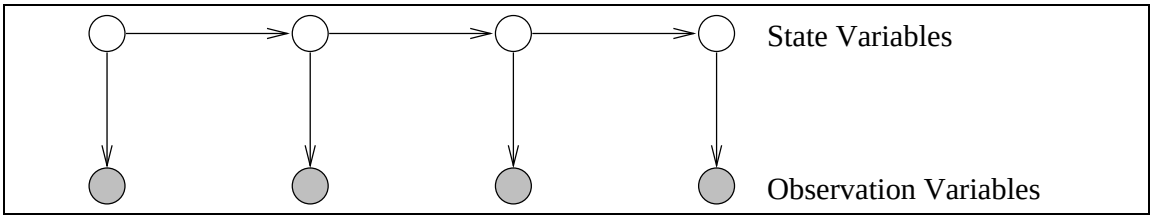

Figure 2: Representation of the simplest HMM.

that represents the value of the observation. The conditioning arrows indicate that the probability of seeing a particular state at time t is conditioned on the value of the state at time $t-1$, and the actual numerical value of this probability reflects the transition probability between the associated states in the HMM. The observation variable at each frame is conditioned on the state variable in the same frame, and the value of $P(o_t|s_t)$ reflects the output probabilities of the HMM.

One important thing to note about the Bayesian network representation is that it is explicit about time: each time frame gets its own separate segment in the model. In Figure 2, there are exactly four time-slices represented, and to represent a longer time series would require a graph with more segments. This is in significant contrast to the HMM, which has no inherent mechanism for representing time. Instead, in the HMM framework, time is represented in the auxiliary data structures used for specific computations.

Figure 3 makes this more explicit. At the top of this figure is an HMM that represents the word "digit." There are five states (unshaded circles) representing the different sounds in the word, and a dummy initial and final state. The arcs in the HMM represent possible transitions, and *not* conditioning relationships as in the Bayesian network. This is a "left-to-right" HMM in which it is only possible to stay in the same state, or move forward in the state sequence. Because of self-loops, this kind of representation need not be explicit about time: it can represent 100 frame occurrences of the word "digit" as easily as 10 or 1000 frame occurrences. Of course, actual computations must be specific about time, and the temporal aspect is introduced into an HMM via the notion of a computational grid or its equivalent. This is shown at the upper right for a seven frame occurrence of the word "digit." The horizontal axis represents time; the vertical axis represents HMM-state, and a path from the lower-left corner of the grid to the upper-right corner represents an explicit path through the states of the HMM. In this case, the first frame is aligned to the /d/ state, the second and third frames are aligned to the /ih/ state, and so forth. Although the structure of the HMM is defined by the graph at the left, computations on the HMM are defined with reference to the temporally-explicit grid.

The Bayesian network representation of the same utterance is shown at the bottom of Figure 3. This is a temporally-explicit structure with seven repeated segments, one for each of the time-slices. The assignment to the state variables of this model corresponds to the same HMM path that is represented in the computational grid. Note that different paths through the grid will correspond to different assignments of values to the state variables in the Bayesian network. Whereas computation in the HMM typically involves summing or maximizing over all *paths*, computation in the Bayesian network typically involves summing or maximizing over all possible *assignments to the hidden variables*. The analogy between a path in an HMM and an assignment of values to the hidden variables in a Bayesian network is quite important and should be kept in mind at all times.

Although the Bayesian network in Figure 3 superficially represents the information in an HMM, closer consideration reveals that in fact there is some extra information that is typically present that this structure does not account for. One example of this is that in practice, the utterances that are associated with an HMM represent complete examples of words. A specific recording of the word "digit" will start with the /d/ sound and end with the /t/ sound; i.e. the whole word will be spoken. This extra piece of information is not captured in the previous Bayesian network. To see this, suppose the value /d/ is assigned to every occurrence of the state variable. (This will actually happen in the course of inference, either explicitly or implicitly depending on the algorithm used.) This concrete assignment will have some probability (gotten by multiplying together many instances of the self-loop probability for /d/ along with the associated observation probabilities), and this probability will not be zero. Yet this assignment violates our prior knowledge that a complete occurrence of "digit" has been spoken.

In the HMM framework, this corresponds to a path that does not end in the upper-right corner of the computational grid, and the problem is solved in the "C-code" that does inference by treating the upper-right corner in a special way. In the Bayesian network framework, prior knowledge such as this must be encoded in the representation itself: the inference algorithms are pure in the sense that they really do sum or maximize over all possible variable assignments.

Another, more subtle, problem concerns the use of time-invariant conditional probabilities. Consider the

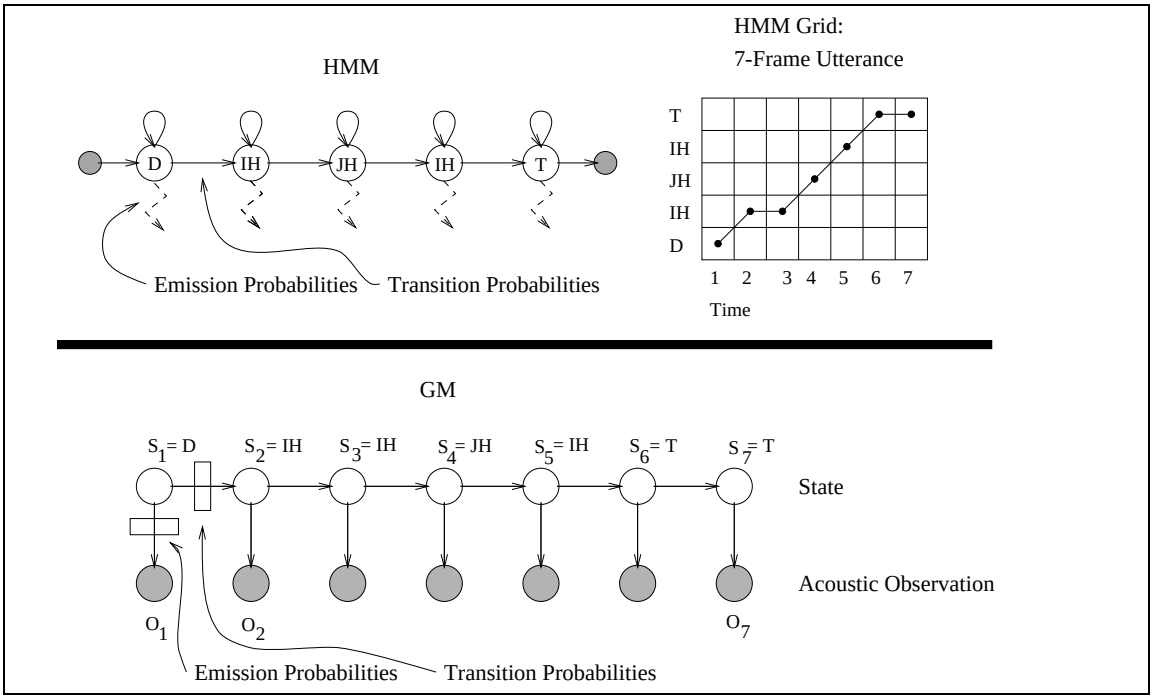

Figure 3: Comparison of HMM and Bayesian network. The dashed lines in the HMM represent the acoustic emissions that occur at each time frame.

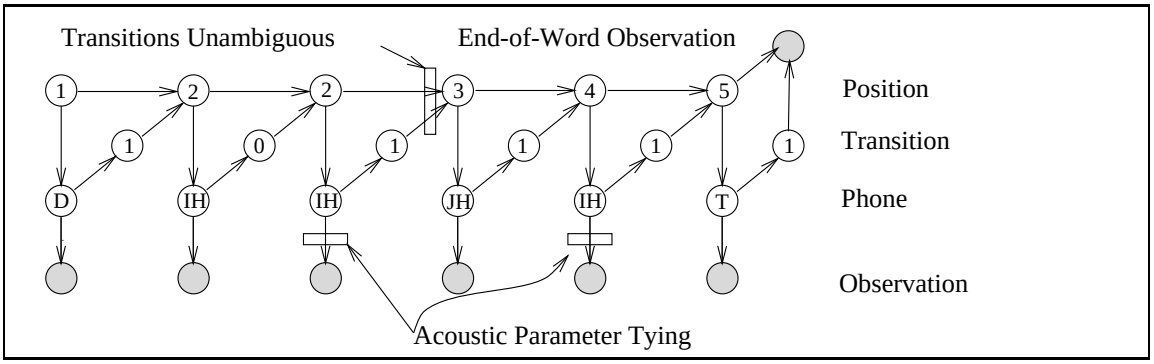

Figure 4: A Bayesian network representation of a typical speech-recognition HMM.

"digit" example. Because the first occurrence of the phone /ih/ is followed by /jh/, it must be the case that $P(S_t = /jh/|S_{t-1} = /ih/) > 0$ - so that the transition is possible - and $P(S_t = /t/|S_{t-1} = /ih/) = 0$ - so that the transition is not skipped. However, because the second occurrence of /ih/ is followed by /t/, it must be the case that $P(S_t = /t/|S_{t-1} = /ih/) > 0$ and $P(S_t = /jh/|S_{t-1} = /ih/) = 0$ which is in direct contradiction to the previous requirements.

One way of solving this problem is to distinguish between the first and second occurrences of /ih/: to define /ih1/ and /ih2/. This is not satisfactory, however, because we may desire that the two share the same output probabilities. This problem is compounded when we consider instances of multiple words. For example, if the word "fit" - /f/ /ih/ /t/ - occurs in the database, should its /ih/ be the same as /ih1/ or /ih2/?

Similar to its handling of word-end constraints, the Bayesian network formalism solves the parameter tying problem via an explicit structural representation. A Bayesian network that incorporates the word-end and parameter tying constraints that are appropriate to a typical practical application is shown in Figure 4.

The main aspect of this representation is that there is now an explicit variable representing the position in the underlying HMM. The position within the HMM is distinct from the phone labeling the position, which is explicitly represented by a second set of variables. The tying problem is solved by mapping from position to phone via a conditional probability distribution. In the example of Figure 4, positions 2 and 4 both map to /ih/. The probability distribution that will be used to model the acoustic observations will be the same in

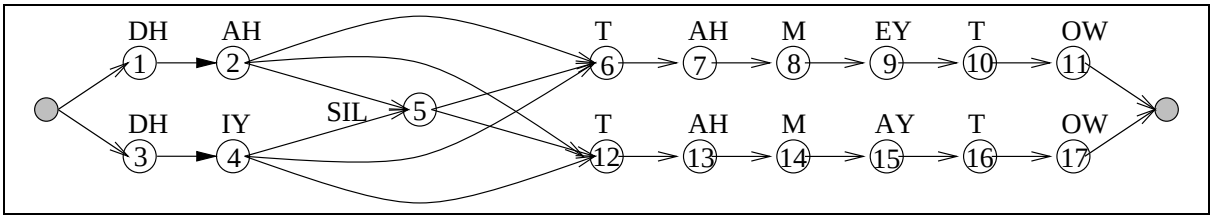

Figure 5: Stochastic finite state automaton for a training utterance. Each state is uniquely numbered.

both cases. Time invariance is achieved via an explicit transition variable, which is conditioned on the phone. Either there is a transition or not, and the probability depends only on the phone. The position is a function of the previous position and the previous transition value: if the transition value was 0, then the position remains unchanged; otherwise it increments by one. These relationships can be straightforwardly encoded as conditional probabilities, ensuring representational consistency.

In this representation, the "end-of-word" observation is assigned an arbitrary value of 1, and ensures that all variable assignments that have non-zero probability must end on a transition out of the final position. This is done by conditioning this observation on the state variable, and setting its conditional probability distribution so that the probability of its observed value is 0 unless the position variable has the value of the last position. Similarly, its probability is 0 unless the transition value is 1. This ensures assignments in conformance with the usual HMM convention that all paths end in a transition out of the final emitting state.

In this model, the conditional probabilities need only be specified for the first occurrence of each kind of variable, and then can be shared by all subsequent occurrences of analogous variables. Thus, we have achieved the goal of making the conditional probabilities time-invariant. However, it is important to note that the probabilities *do* depend on the specific utterance being processed. In this model, the mapping from position to phone changes from utterance to utterance, as does the final position. Thus an implementation must support some mechanism for representing and reading in conditional probabilities on an utterance-by-utterance basis. This is analogous to reading in word-graphs, lattices, or scripts on an utterance-by-utterance basis in an HMM system.

A final nuance of this model is that some of the conditional probability relationships are in fact deterministic, and not subject to parameter estimation. For example, the distribution controlling the position variable encodes a simple logical relationship: if the transition-parent is 0, then $Position_t = Position_{t-1}$; otherwise, $Position_t = Position_{t-1} + 1$. Efficiently representing deterministic relationships, and exploiting them in inference, is important for an efficient implementation of a Bayesian network system.

The Bayesian network in Figure 4 is useful in a variety of tasks:

- Parameter estimation when the exact sequence of words and therefore phones, including silences, is known.
- Finding the single best alignment of a sequence of frames to a known sequence of words and phones.
- Computing the probability of acoustic observations, given a known word sequence - which is useful for rescoring the n-best hypotheses of other recognition systems.

The main drawback of this model is that a fully specified state sequence must be used, which is only present if one knows where silence resides between words. For example, the state sequence corresponding to "hi there" will in general be different from "hi *sil* there," though in practice one does not know where the silences occur.

In order to properly handle training and decoding, it is necessary to employ networks that capture the underlying semantics of finite state models that are somewhat more complex than that of Figure 4. In training, this occurs for two reasons: because of the uncertain occurrence of silence between transcribed words, and because of multiple word pronunciations. Neither the presence of silence nor the particular pronunciation variant that was used is indicated in the word transcriptions used for training. In decoding, the problem arises from the complete uncertainty in the word sequence. We now look more closely at these models, beginning with training - because with the assumption of fixed probabilities for pronunciation variants and optional silence, the training network is actually no different in structure from the previous one. We note here, too, that there are many ways of structuring training and decoding networks, and the approach we take here is just one of several possible.

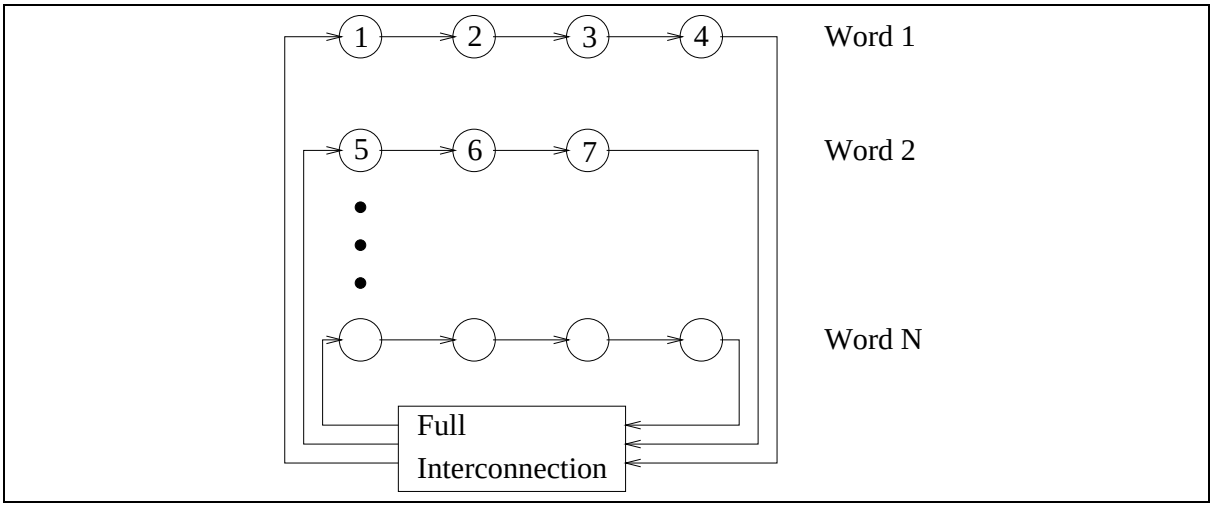

Figure 6: Stochastic finite state automaton for a simple decoding network. There is an arc from the final state of each word to the initial state of every word. The states of the first two words are numbered.

2.2 Training with Optional Silence and Lexical Variants

When there are multiple word pronunciations and optional silences, the automaton corresponding to possible word sequences is more complex than the linear sequences we have seen previously. Figure 5 shows an automaton that is appropriate for the word sequence "the tomato", where both words have two possible pronunciations, and there might be silence in the acoustics between them. Note that self-loops are implicit in all the emitting (unshaded) states.

The model of Figure 4 is able to represent this complexity without modification. Position now refers to the position in the new automaton. The mapping from position to phone is straightforward, as before. However, the conditional probability distribution associated with the position variable is more complicated, and defined below. In this and other descriptions, when we refer to the value of one variable (e.g. transition) affecting another (e.g. position), the first variable will always be a direct parent of the second.

The position variable behaves as follows:

- If there is no transition, the position value is the same as its previous value with probability 1.
- If there is a transition, then there are two cases:
  1. If the position is not word-terminal (e.g. 8 in the above example), then with probability 1 the position variable takes a value one greater than its previous value (e.g. 9).
  2. Otherwise, if silence is a possible successor (e.g. as it is for states 2 and 4), the beginning state of silence has a probability equal to the probability of inserting optional silence, P_s. The initial states of the following words have a probability equal to $1 - P_s$ multiplied by their pronunciation probability. If silence is not a possible successor (e.g. as with state 5), the initial states of the following words have a probability equal to their pronunciation probability.

In this approach, we have implicitly combined optional silence and pronunciation probabilities into a single representation in the conditional probability distribution of the position variable. This has no repercussions as long as the probabilities are fixed. If one desires to learn them, however, it is necessary to create a network with explicit variables indicating when silence should be inserted, and what pronunciations are being used. This is useful in keeping the sufficient statistics of these two different distributions separate.

2.3 Decoding Networks

In this section, we will assume that speech is modeled by a simple finite state process that encodes a full bigram language model. The extensions to trigram language models, sparse models, and smoothed language models (e.g. (Kneser & Ney 1995)) follow by analogy. Figure 6 illustrates the underlying finite state system. In this figure, every lexical variant of every word is represented on a separate line. The states that are present on a line represent the internal acoustic states of the lexical variant; for example, one of them might correspond to

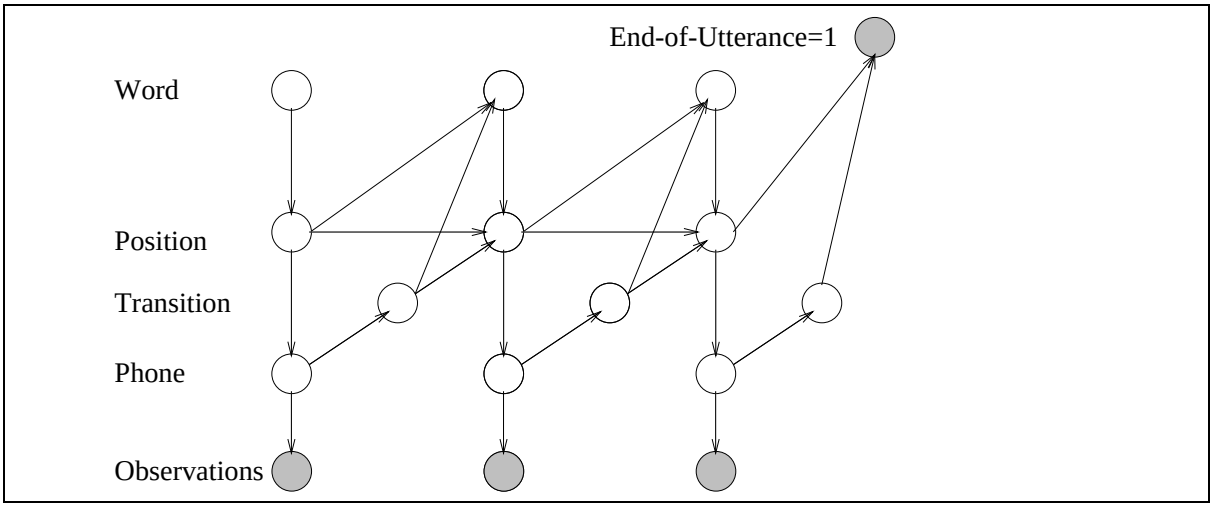

Figure 7: A Bayesian network for a word-internal acoustic context, bigram language-model decoder.

the phone sequence /D IH JH IH T/. Self-loops are implicit in all the states of this automaton, though they are not drawn to simplify the picture. From the end of the state sequence for a particular word there are transition arcs to the initial states of all the words in the vocabulary. The transition probabilities correspond to the bigram language model probabilities of the system. In this model, it is possible to use any sequence of acoustic states for a word, and in particular word-internal state-clustered triphones can be used; to incorporate cross-word context, a more sophisticated automaton is necessary. The states are uniquely numbered, although some of the states may represent the same acoustic models.

Figure 7 illustrates a Bayesian network that corresponds to the previous figure. Assignments of values to the hidden variables correspond to paths through the automaton, and will receive the same probabilities as in the automaton. The meaning of the phone, transition, and observation variables is the same as in the previous networks. The value of the position variable, however, now refers to the position in the automaton of Figure 6, rather than position in a linear sequence as before. There is also a new variable, the word variable, that indicates what word is being spoken at a given time frame. The conditional probability distributions associated with the word and position variables are as follows:

- Position. When the transition value is 0, the position variable takes its previous value with probability 1. When the transition value is 1, there are two cases:

    1. The previous position value did not correspond to the end of a word (e.g. it had value 3 in the Bayesian network corresponding to Figure 6). In this case, the new position value is one greater than the previous value, with probability 1 (e.g. it takes a value of 4).
    2. The previous position value *did* correspond to the end of a word (e.g. it had value 7). In this case, the position variable takes the value of the first position of the word indicated by the word variable, with probability 1. For example if the word variable had value 2, the position value would be 5.

- Word. When the transition value is 0, the word variable takes the value of the word that the previous position variable indicates, with probability 1. For example, if the previous position has value 7, the word value is 2. When the transition value is 1, there are two cases:

    1. The previous position value did not correspond to the end of a word. This case is the same as the last.
    2. The previous position value *did* correspond to the end of a word. In this case, note that the identity of the previous word is implicit in the previous state. The distribution over word values is given by bigram language model probabilities, given the previous word.

The decoded word sequence can be read off from the sequence of word-variable values in frames that follow occurrences of a transition in a word-terminal state.

The training and decoding networks that we have discussed are some of the simplest possible, and more sophisticated ones can be created. Rather than enumerating more of these, we note that these all use the same principles illustrated above, and turn now to a discussion of the variety of other phenomena that can be modeled in an integrated way with Bayesian networks.

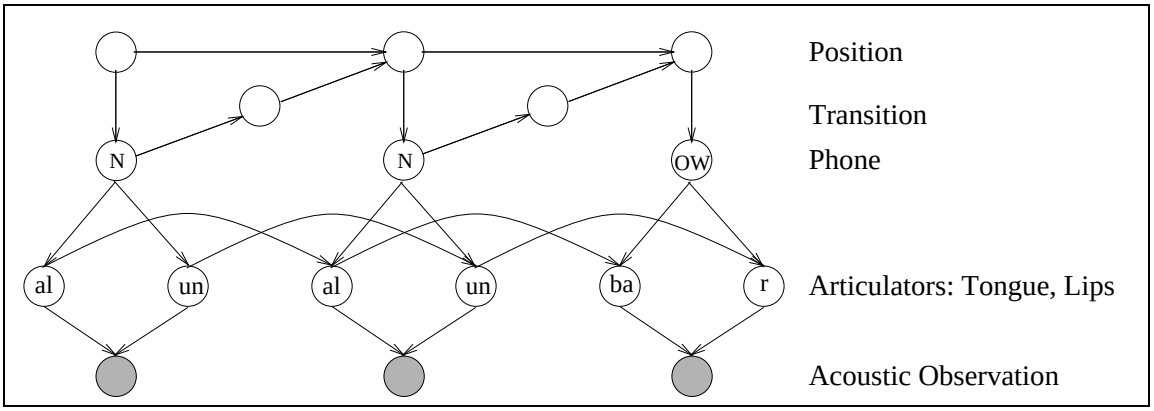

Figure 8: An articulatory Bayesian network. Tongue and lips are explicitly represented, and values are assigned that are characteristic of an utterance of the word "no." The tongue begins in the alveolar position (al) and moves to the back of the mouth (ba). The lips start unrounded (un) and become rounded (r).

2.4 The Range of Graphical Models

In sections 2.1 through 2.3, we saw how to represent a variety of standard speech recognition models in the Bayesian network framework. Now we show how new and interesting phenomena can be easily captured with Bayesian networks.

2.4.1 Articulatory Models

Speech is of course generated by the motion of a variety of articulatory organs: the lungs, glottis, diaphragm, tongue, lips, and jaw, to name a few. Traditionally, there has not been any explicit representation of the physical bodies involved in speech generation, with the complete state of the system being implicitly represented in the value of a HMM state. In the Bayesian network framework, however, it is quite easy to make models with explicit representations of the speech articulators. Figure 8 shows the basic idea behind an articulatory Bayesian network.

The main innovation of this model is an additional layer of articulatory variables intermediate between the phone and observation variables. The intuition behind the model is that the value of the phone variable represents the speaker's intention at some particular time, and that the values of the articulatory variables represent the state of the actual physical system producing the sound. The articulators are conditioned both on the value of the phone - their state will depend to some extent on what the speaker wants to say - and also on their own previous values - inertia will place constraints on the articulators regardless of the speaker's intentions.

The main issue in models such as these is to ensure that the variables represent what we want them to, even after training. This can be done by either placing strong constraints on the initial parameter values, or training with data that is fully observed (Richardson *et al.* 2000). Similar HMM approaches can be found in (Deng & Erler 1992; Erler & Deng 1993; Erler & Freeman 1996).

2.4.2 Noise Modeling

In many current speech recognition applications, the robust handling of noisy environments is crucial, and here, too, the expressive power of Bayesian networks is useful. For example, in an automotive application, one might have variables representing the speed of the car, the road condition, the status of the air-conditioner, whether or not the windows are open, and other factors affecting the acoustic environment. Clearly, some factors are independent of others, e.g. road condition is independent of air-conditioning, while others are dependent, e.g. whether the windows are open and the status of the air-conditioner. Thus, a sparse conditioning structure is likely to be desirable.

While very complex models are possible, quite simple ways of addressing noise are also available, as Figure 9 illustrates in a Bayesian network representation of parallel model combination (Varga & Moore 1990; Gales & Young 1992). In the top half of this figure there is a standard representation of an HMM. This produces a stream of hidden acoustic vectors that represent the speech alone. The bottom half of the figure represents a noise HMM that produces a stream of pure (but unobserved) noise. The actual acoustic observations, in the center of the figure, represent the addition of the noise and speech streams.

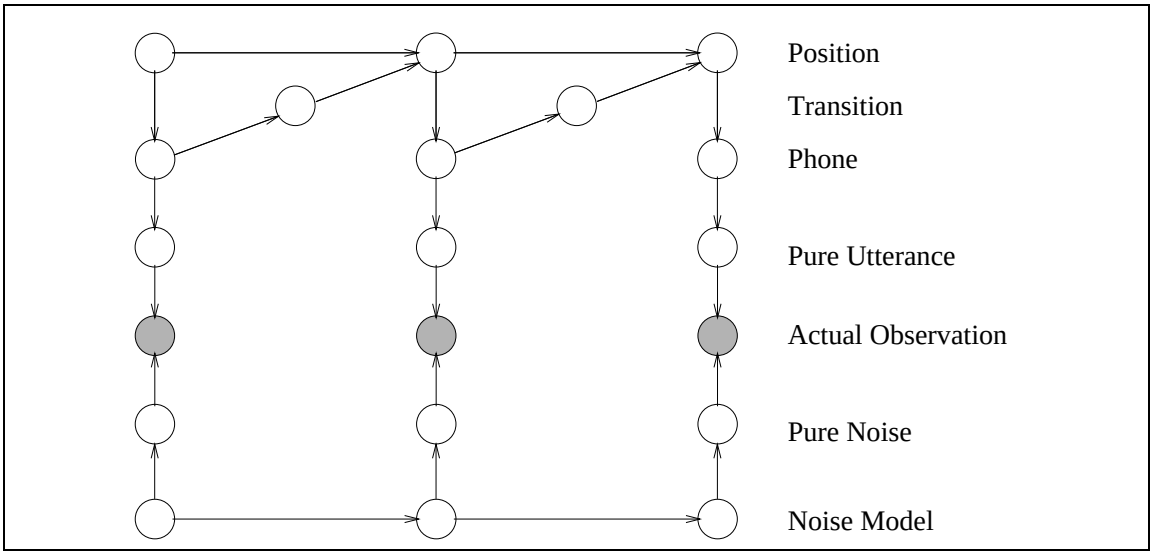

Figure 9: Parallel model combination with a Bayesian network.

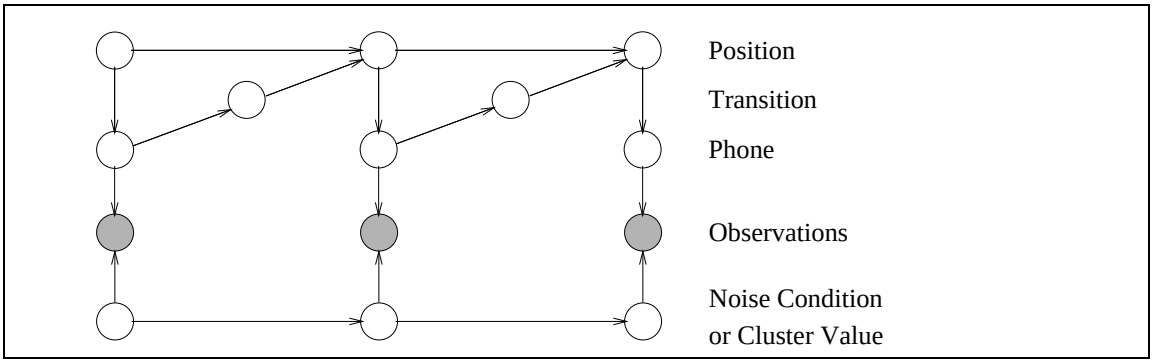

Figure 10: A very simple noise model. The same structure can be used for speaker or utterance clustering.

The main complexity of this model is that it requires a computational specification of how two hidden acoustic streams combine to produce the actual observations. Even this can be avoided with the model of Figure 10, where the "noise condition" variable takes a small number of discrete values, e.g. "high" and "low." In this case, different output distributions will be used depending on the noise condition.

2.4.3 Speaker Clustering

To improve model specificity, it is not unusual to build models for male and females separately, and then to do recognition in a two-step process in which there is a gender-determination step followed by recognition with the appropriate models. Alternatively, the clustering process can be done automatically, to an arbitrary number of clusters (with the side-effect that the cluster values no longer have a clear meaning).

In the Bayesian network context, the structure of Figure 10 can be recycled to do utterance clustering rather than noise modeling. In this case, the noise variable is used instead to denote cluster value. To prevent the value from "flip-flopping" within the utterance, the conditional probabilities of the cluster variables from time frame 2 forward are set to implement the deterministic operation of copying the previous value. Training the cluster model can be done either in a supervised way with the cluster value set during training, or in an unsupervised way with the cluster variable hidden in training.

2.4.4 Speech Rate Modeling

As a final example of the range of effects that can be represented with a Bayesian network, Figure 11 shows a model for tracking rate-of-speech. In this model, there is an explicit rate-of-speech variable, and the transition variable is conditioned on it. The rate variable is conditioned on its previous occurrence in order to model

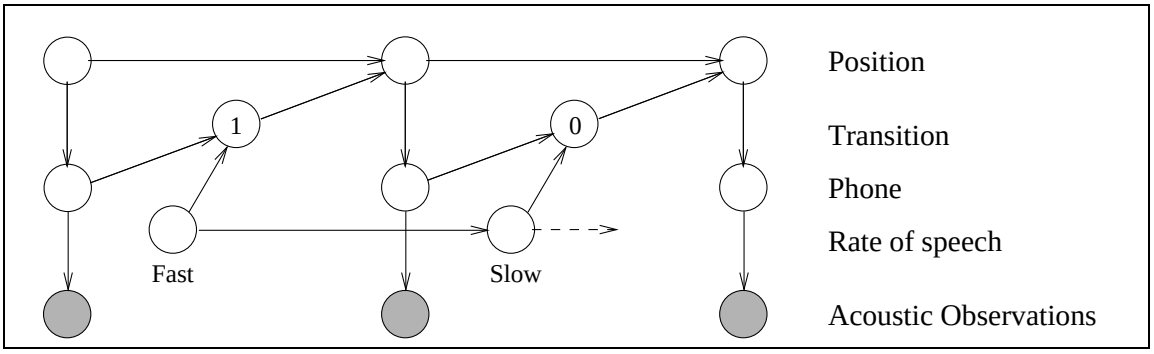

Figure 11: A rate-of-speech model.

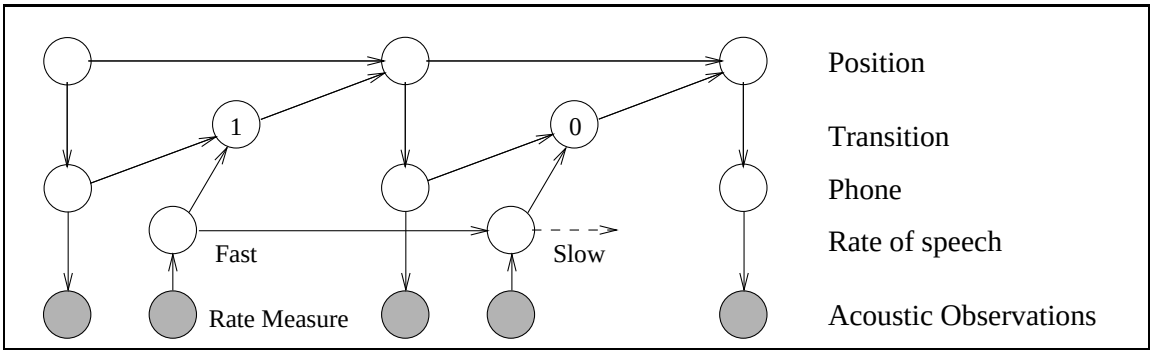

Figure 12: A rate-of-speech model, where the rate is conditioned on an acoustic observation.

continuity – presumably the rate does not flip-flop on a frame by frame basis. The intention is that when the rate-of-speech is fast, the probability of a transition will be higher than when the rate is slow, and this behavior can be ensured by doing supervised training where the rate is known.

Figure 12 shows a slightly more sophisticated model in which there is an acoustic observation that is strongly correlated with the speech rate. Here, the rate variable is actually conditioned on the observation, and in turn affects the transition variable.

3 Inference

In this section we turn from representation to computation, and outline the procedures for computing with Bayesian networks. There are three main computational tasks that can be performed with the standard algorithms:

1. Computation of the probability of the observed variable values: $P(\mathbf{o}) = \sum_{\mathbf{h}} P(\mathbf{o}, \mathbf{h})$
2. Computation of the likeliest hidden variable values: $argmax_{\mathbf{h}} P(\mathbf{o}, \mathbf{h})$
3. Parameter estimation from a set of observations $\{\mathbf{o}_k\}$ via EM: $argmax_\theta \prod_k P(\mathbf{o}_k|\theta)$

There are many ways of performing the tasks enumerated above, and the first major distinction is between exact and approximate inference. In an exact inference technique, the correct answer is obtained (to within numerical precision) regardless of the run-time. Approximate techniques, in contrast, provide approximate answers in significantly less time. To see why speed can be an issue with Bayesian networks, consider the model of Figure 13.

This figure presents a model that encodes an instance of the 3-satisfiability problem (Garey & Johnson 1979), a well-known NP-complete problem. In this problem, there is a set of binary variables $\{X_1 \ldots X_n\}$ and a collection of disjunctions of the form $(X_i \vee X_j \vee X_k)$, with the variables possibly negated. The object is to find an assignment of values to the variables such that each disjunction evaluates to true. Because values can be negated, this is non-trivial. In Figure 13, the top layer of variables are the binary variables X_i; the middle layer encodes "or" relationships between the top-level variables, and the bottom layer performs an "and" operation on

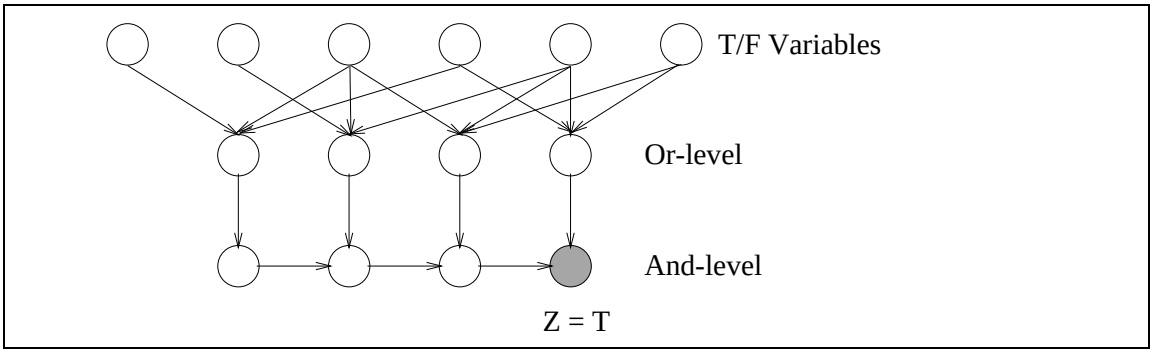

Figure 13: A model that encodes a 3-satisfiability problem.

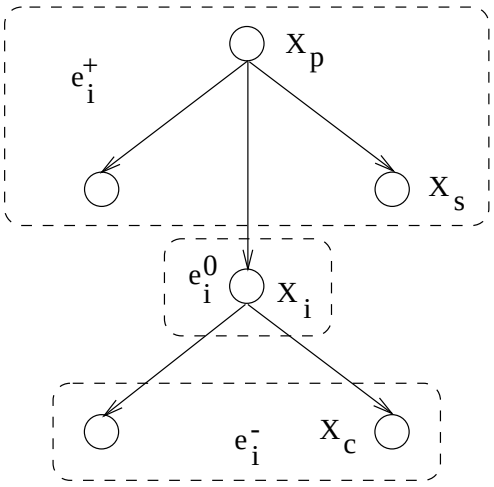

Figure 14: A tree of variables.

the output of the "or" variables. The bottom-rightmost variable is observed to have the value "true," and the query is posed, "what is the likeliest assignment of values to the hidden variables?" The SAT instance is satisfiable if a non-zero probability assignment is returned. All known methods for solving SAT problems have a compute time that increases exponentially on the number of variables, and it is widely believed that it is impossible to better. Thus it is quite likely that exact inference in Bayesian networks will require exponential time in the worst case. In fact, it has also been shown (Dagum & Luby 1993) that even *approximate* inference belongs to the class of NP-hard problems, and is therefore theoretically just as difficult as exact inference.

Fortunately, the worst case does not happen for many Bayesian network structures of practical interest, and both exact and approximate inference methods are feasible. In the remainder of this section, however, we will focus on exact inference. From the point-of-view of experimentally evaluating model structures, this ensures that the conclusions are not biased by potentially poor approximations.

In the following sections, we present an inference technique that is derived from (Peot & Shachter 1991) and similar to (Pearl 1988; Jensen *et al.* 1990; Spiegelhalter & Lauritzen 1990). However, it extends these methods by handling deterministic variables in a very efficient manner, and is cast purely in terms of simple dynamic programming recursions analogous to $\alpha - \beta$ computation in HMMs. The method is extremely straightforward in the case of tree-structured networks, and our presentation differs from others by casting the general case as a change-of-variables in which a complex network is transformed into an equivalent tree-structured network, where the original inference method is applied. We present a minimal set of conditions that must be met for this transformation to be valid.

3.1 Inference on Trees

The simplest case is when the Bayesian network is tree structured, i.e. there is no path that leads from a variable back to itself, even if directionality is ignored. In this case, the runtime is proportional to the number of variables in graph, and the actual inference procedure is quite straightforward.

Algorithm **Inference**()
for each variable X_i in postorder
  if X_i is a leaf
    $\lambda_j^i = 1,\ j \in CON(e_i^0);\ \lambda_j^i = 0,\ j \notin CON(e_i^0)$.
  else
    $\lambda_j^i = \prod_{c \in children(X_i)} \sum_f \lambda_f^c P(X_c = f | X_i = j), j \in CON(e_i^0);\ \lambda_j^i = 0,\ j \notin CON(e_i^0)$.

for each variable X_i in preorder
  if X_i is the root
    $\pi_j^i = P(X_i = j)$
  else
    let X_p be the parent of X_i
    $\pi_j^i = \sum_{v \in CON(e_p^0)} P(X_i = j | X_p = v) \pi_v^p \prod_{s \in siblings(X_i)} \sum_f \lambda_f^s P(X_s = f | X_p = v)$

Figure 15: Inference in a tree.

To see how tree-based inference works, consider the graph in Figure 14. The variables that are relevant for the subsequent discussion are X_i; X_p, which is X_i's parent; X_s, which is its sibling; and X_c, which is its child. In this graph, the values of some of the variables may be known, and we refer to these values as "evidence." Note that X_i partitions the remaining variables into two disjoint sets: the set of variables *rooted* in X_i, and all other variables. The set of evidence pertaining to the variables rooted in X_i is referred to as e_i^-, while the evidence pertaining to the variables not rooted in it is referred to as e_i^+. The evidence pertaining to X_i itself is referred to as e_i^0. We denote the set of values of X_i that are consistent with e_i^0 by $CON(e_i^0)$. In the case at hand, where X_i is a single variable, this means that when X_i is hidden, $CON(e_i^0)$ contains all its possible values, and when X_i is observed, $CON(e_i^0)$ contains only its observed value. In section 3.2, we will consider the more general case of composite variables whose values range over the Cartesian product of a set of constituent variables. In this case, $CON(e_i^0)$ refers to the set of cross-product values that are consistent with all known constituent values.

Note that the union of the evidence includes all the observations, and the probability of all the evidence and variable X_i taking value j is:

$$\begin{aligned} &P(e_i^+, e_i^-, e_i^0, X_i = j) = \\ &\quad P(e_i^+, X_i = j) P(e_i^-, e_i^0 | X_i = j, e_i^+) = \\ &\quad P(e_i^+, X_i = j) P(e_i^-, e_i^0 | X_i = j). \end{aligned}$$

In the case that $X_i = j$ contradicts e_i^0, $P(e_i^-, e_i^0 | X_i = j)$ is zero.

This leads to a natural factorization, and in the inference procedure the following two key quantities will be calculated for each variable X_i:

- $\lambda_j^i = P(\mathbf{e}_i^-, \mathbf{e}_i^0 | X_i = j)$
- $\pi_j^i = P(\mathbf{e}_i^+, X_i = j)$.

It follows from these definitions that for every variable X_i,

- $P(Observations) = \sum_j \lambda_j^i \pi_j^i$.
- $P(X_i = j | Observations) = \frac{\lambda_j^i \pi_j^i}{\sum_j \lambda_j^i \pi_j^i}$.

Figure 15 presents an algorithm for computing these values.

Before determining the runtime of this algorithm, it is useful to make a small simplification. If we define τ_v^i as

$$\tau_v^i = \sum_f \lambda_f^i P(X_i = f | X_p = v)$$

then we may rewrite λ_j^i for a non-leaf variable as:

$$\lambda_j^i = \prod_{c \in children(X_i)} \tau_j^c$$

Algorithm **Inference**()
for each variable X_i in postorder
  if X_i is a leaf
    $\lambda_j^i = 1,\ j \in CON(e_i^0); \lambda_j^i = 0,\ j \notin CON(e_i^0).$
  else
    $\lambda_j^i = \prod_{c \in children(X_i)} \tau_j^c,\ j \in CON(e_i^0); \lambda_j^i = 0,\ j \notin CON(e_i^0).$
  $\tau_v^i = \sum_f \lambda_f^i P(X_i = f | X_p = v)$
for each variable X_i in preorder
  if X_i is the root
    $\pi_j^i = P(X_i = j)$
  else
    let X_p be the parent of X_i
    $\pi_j^i = \sum_{v \in CON(e_p^0)} P(X_i = j | X_p = v) \frac{\pi_v^p \lambda_v^p}{\tau_v^i}$

Figure 16: Inference in a tree, with τ caching.

Algorithm **Viterbi** ()
for each variable X_i in postorder
  if X_i is a leaf
    $\lambda_j^{*i} = 1,\ \ j \in CON(e_i^0); \lambda_j^{*i} = 0,\ j \notin CON(e_i^0).$
  else
    $\lambda_j^{*i} = \prod_{c \in children(X_i)} \max_f \lambda_f^{*c} P(X_c = f | X_i = j),\ j \in CON(e_i^0); \lambda_j^{*i} = 0,\ j \notin CON(e_i^0).$

Figure 17: Viterbi inference in a tree. This produces the probability of the likeliest variable assignments. The actual variable values can be recovered by storing the child-values that lead to each λ_j^{*i}.

and π_j^i as

$$\sum_{v \in CON(e_p^0)} P(X_i = j | X_p = v) \pi_v^p \frac{\lambda_v^p}{\tau_v^i}$$

Hence if the τ-factors are stored in the bottom-up pass, the product over siblings in the original formula can be done with a single division. Interestingly, if $\tau_v^i = 0$, then any term involving division by it can be also set to 0, i.e. ignored in the sum. This is proved in Appendix 1. Figure 16 presents the steps explicitly.

Now we can calculate the runtime. The main computation in the bottom-up pass is the computation of the τ factors. This requires enumerating, for each parent-child-pair, all combinations of values for the two variables. If we assume a constant cardinality of k for the N variables in the tree, the runtime of the backward pass is thus $O(Nk^2)$. Similarly, the runtime of the top-down pass requires a loop over each parent-child pair. Thus the total runtime is $O(Nk^2)$. It is easy to see that the runtime cannot be improved on: in general, there are $O(Nk^2)$ conditional probabilities associated with the network, so it takes that long just to read in all the parameters.

To obtain the likeliest values for the variables, we denote an assignment to the hidden variables rooted in X_i as $\mathbf{x}_i^-$, and redefine the λ as follows:

λ_j^{*i} = Probability of the likeliest values for the variables rooted in X_i.

$= \max_{\mathbf{x}_i^-} P(e_i^-, e_i^0, \mathbf{x}_i^- | X_i = j)$

Similar definitions can be made for the π quantities, though they are unnecessary for the task at hand. The recursions for computing the λ_j^{*i} are given in Figure 17. The actual values $\mathbf{x_i}$ can be obtained by storing the child-values that led to each λ^*.

3.2 Inference in General Graphs

The algorithm presented in section 15 works only in tree-structured graphs. In this section, we present a technique for doing inference in graphs with arbitrary topology. The method of attack is to use a change-of variables. We will define a new tree-structured network in terms of a new set of variables in such a way that the new network represents exactly the same joint probability distribution as the old network. We will then be able to use the simple tree-inference algorithm.

3.2.1 Equivalent Representations of Probability Distributions

Suppose we have two Bayesian networks $\mathcal{N}$ and $\mathcal{N}'$ over the sets of variables $\mathbf{X}$ and $\mathbf{X}'$ respectively. Let $\mathbf{x}_i$ denote a joint assignment of values to the variables in $\mathbf{X}$, and let $\mathbf{x}_i^{'}$ denote a joint assignment of values to the variables in $\mathbf{X}'$. Let ξ be the set of all possible joint assignments of values to the variables in $\mathbf{X}$; i.e. $\xi = \cup_i \mathbf{x}_i$. Let $\xi^{'}$ be the set of all possible joint assignments of values to the variables in $\mathbf{X}'$; i.e. $\xi^{'} = \cup_i \mathbf{x}_i^{'}$. Assume without loss of generality that $\mid \xi^{'} \mid >= \mid \xi \mid$. Let ξ^* denote a subset of $\xi^{'}$ such that $\mid \xi^* \mid = \mid \xi \mid$. We will use P to represent probabilities associated with $\mathcal{N}$, and $P^{'}$ to represent those associated with $\mathcal{N}'$.

Definition 3.1
$\mathcal{N}'$ represents the same distribution as $\mathcal{N}$ if the following conditions are met:

1. *There is a one-to-one correspondence between the members of ξ and the members of ξ^*. For notational convenience, let $\mathbf{x}_i$ be associated with $\mathbf{x}_i^*$.*
2. *For each such pairing, $P(\mathbf{x}_i) = P'(\mathbf{x}_i^*)$.*
3. *For all joint assignments $\mathbf{x}_i^0 \in \xi^{'} \setminus \xi^*$, $P'(\mathbf{x}_i^0) = 0$.*

These conditions imply that the sum or max over any subset of ξ can be computed by performing the same operation on a well-defined subset of $\xi^{'}$.

3.2.2 Inference with Trees of Composite Variables

In this section, we will derive the clique tree inference algorithms (Lauritzen & Spiegelhalter 1988; Jensen *et al.* 1990) from the principles set out in section 3.2.1. We will structure the derivation by satisfying each requirement of Definition 3.1 in turn.

3.2.3 Correspondence Requirement

The variables in the tree will be defined to represent subsets of the variables in the original network. Each composite variable or clique ranges over the Cartesian product of the variables in the subset associated with it. This creates the one-to-one mapping from each joint assignment in the original network to a joint assignment in the tree.

3.2.4 Equal Probability Requirement

We will now add a further stipulation that each variable in the original network must occur together with its parents in at least one of the composite variables. We may thus "assign" each variable in the original network to a composite variable C_w in which it occurs with its parents. Let $\mathbf{F}_w$ represent the variables in the original network that are assigned to C_w. Let C_p and C_c denote two composite variables in a parent-child relationship in the tree. Define $P'(C_c = i|C_p = j)$ to be $\prod_{X_k \in \mathbf{F}_c} P(x_k|Parents(X_k))$, with x_k and $Parents(X_k)$ implied by i. In the case of a composite variable C_r with no parents, define $P'(C_r = i)$ in to be $\prod_{X_k \in \mathbf{F}_r} P(x_k|Parents(X_k))$, with x_k and $Parents(X_k)$ implied by i. In the case that $\mathbf{F}_c = \emptyset$, the conditional probabilities are defined to be 1, unless i and j imply inconsistent values for shared variables, in which case the conditional probability is 0. These definitions ensure equality between the individual factors used to compute the probability of a joint assignment in the two representations, and thus guarantee that $P'(\mathbf{x}_i^*) = P(\mathbf{x}_i)$ for all $\mathbf{x}_i \in \xi$. Operationally the required condition can be ensured by the process conventionally known as "moralization," and we refer to the property as **MORAL**.

3.2.5 Zero Probability Requirement

There are more possible joint assignments to the composite variables than to the original variables. However, each of the assignments to the composite variables for which there is no analog in the original network must imply the simultaneous assignment of inconsistent values to at least one of the original variables.

To satisfy the final requirement, we will impose two further stipulations.

1. We will define $P(C_c = i|C_p = j) = 0$ if i and j imply inconsistent values for shared variables.

2. We will require that the members of the composite variables satisfy the running intersection property (Pearl 1988): if two variables from the original network occur in any pair of composite variables, they also occur in every composite variable along the path connecting the two.

To see that this is sufficient, it suffices to realize that all inconsistencies must involve either:

1. two occurrences of an original variable in composite variables that are in a parent-child relationship, or

2. two occurrences of an original variable in composite variables that are separated by other composite variables.

The first requirement ensures that all inconsistencies of the first variety receive zero probability; the second requirement ensures that all inconsistencies of the second type imply an inconsistency of the first kind, and therefore receive zero probability. Operationally, the running intersection requirement can be satisfied by the process of triangulation (Rose 1970).

3.2.6 Summary of Inference in a Clique Tree

The variables in a clique tree represent subsets of the original variables. Algorithmically, the most natural way to view these subsets is as new variables. Each new clique-variable can take a distinct value for every possible assignment of values to its members. Each variable in the original network is assigned to a single clique in which it occurs with its parents (when there are several such cliques, and an arbitrary choice may be made). The inference procedure outlined in Section 15 works on clique trees with the following change of variables:

- $\mathbf{e}_i^0$ is the set of observed values for the evidence variables assigned to clique C_i.
- $\mathbf{e}_i^-$ is the set of observed values for the evidence variables assigned to cliques in the subtrees rooted in C_i.
- $\mathbf{e}_i^+$ is the set of observed values for all other evidence variables.
- $\lambda_j^i = P(\mathbf{e}_i^- | C_i = j)$
- $\pi_j^i = P(\mathbf{e}_i^+, C_i = j)$.
- $\mathbf{F}_c$ is the set of variables assigned to C_c.
- $P(C_c = i | C_p = j) = 0$ if i and j imply inconsistent values for shared variables. Otherwise, if $\mathbf{F}_c \neq \emptyset$,
  - $P(C_c = i | C_p = j) = \prod_{X_k \in \mathbf{F}_c} P(x_k | Parents(X_k))$, with x_k and $Parents(X_k)$ implied by i.
  - $P(C_r = i) = \prod_{X_k \in \mathbf{F}_r} P(x_k | Parents(X_k))$, with x_k and $Parents(X_k)$ implied by i.
- Any remaining conditional probability is defined to be 1.

3.2.7 Separators

In an implementation, it is also beneficial to introduce separator variables between each pair of parent-child cliques. These separator variables represent the variables in $\{C_p \cap C_c\}$. The value of a separator's parent uniquely defines the separator's value, and the conditional probability of a separator taking its one allowed value given its parent's value is always 1. The inference algorithm remains unchanged; the separators simply represent extra variables which have the same status as any other variables. This view of separator cliques is different from that taken in (Lauritzen & Spiegelhalter 1988; Jensen *et al.* 1990) where separators play a fundamental role in representing the probability distribution.

The use of separators can have a dramatic effect on running time since the work done for each clique is proportional to the number of values it can take multiplied by the number of values its parent can take, subject to the consistency constraints imposed by shared variables. For example, suppose there is a network consisting of variables A, B, C and D, which take a, b, c and d values respectively. In a clique tree consisting of the cliques $\{A, B, C\}$ and $\{B, C, D\}$, the introduction of a separator-variable representing $\{B, C\}$ reduces the work from $a * b * c * d$ to $a * b * c + b * c * d$. Runtime is further discussed in Section 3.4.

3.3 Fast Inference with Deterministic Variables

As we have seen, modeling word pronunciations with a Bayesian network requires the extensive use of deterministic variables. In contrast to regular stochastic variables, deterministic ones impose significant constraints, and in order to achieve efficient inference, these must be addressed.

There is the further complication that the deterministic relationships change on a word-by-word basis. Essentially, the same network structure is "reprogrammed" on a word-by-word basis to do the dynamic programming that is appropriate for that word model. Thus, it is not possible to determine the repercussions of the deterministic relationships just once in a network-compilation stage, as is done typically, e.g. in the *HUGIN* system (A/S 1995). The *HUGIN* approach also suffers from the flaw that it first tabulates and stores every possible clique value, and then throws away the ones that can be proven to have zero probability. The space required for this approach can be prohibitive in networks with a large number of deterministic relationships.

A final point is that the possible clique values will in general vary depending on the pattern of evidence observed. Thus any static compilation scheme is bound to miss some efficiencies that a dynamic scheme will be able to identify and exploit on a case-by-case basis. Therefore, our approach is to structure the network so that the possible clique values can be swiftly enumerated on demand.

3.3.1 Approach

The basic approach is to do a preorder traversal of the tree and recursively identify the values that are legal - i.e. that have nonzero probability - for a child clique, given the just-computed legal values of the parent. Since neighboring cliques typically share variables, it is usually the case that knowing the possible values of the parent clique significantly reduces the set of possible values for a child clique.

Examination of the inference procedures shows that all the loops are of the form "for each value of a parent clique and for each value of a child clique, do a handful of multiplications and additions." Thus, once the legal values of a child are identified for each of the possible parent values, the inference loops can be made considerably more efficient by iteration just over the subset of values that are possible.

Deterministic variables impose a constraint on the feasibility of this approach: in order to resolve the legal values in a single pass, it is necessary that the first time a deterministic variable appears in a clique, its parents also be present. Without this constraint, the (unique) value of a deterministic variable cannot necessarily be resolved when it is first encountered. This property is a fundamental requirement, and we restate it as follows:
Immediate Resolution Property **IRP**: In a preorder traversal of a clique tree, each deterministic variable first appears in a clique with all its parents. This is equivalent to the Strong Clique Tree property of (Jensen *et al.* 1994) for influence diagrams with decision variables, and can be guaranteed with minor modifications to standard clique-tree building techniques (Jensen *et al.* 1994; Pearl 1988; Lauritzen & Spiegelhalter 1988; Zweig 1998).

3.3.2 Identifying the Legal Clique Values

Once a clique tree satisfying **IRP** is set up, there is a simple procedure for identifying the legal values of each clique. High-level pseudocode for the procedure is given in Figure 18. In the starred (*) steps, it is necessary to enumerate all the combinations of values for subsets of variables. This is done by recursively assigning values to the variables. By processing the variables in topological order, we are guaranteed that the parents of a deterministic variable will have values assigned *before* the variable is processed. This guarantees that its value can be determined. Via the **IRP** property, these legal values will be identified as soon as the variable is first seen, and then propagated down the tree.

3.4 Discussion of Time and Space Requirements

3.4.1 Identifying Possible Clique Values

Consider two cliques C_p and C_c in a parent-child relationship. The legal values of C_c can be found by recursively instantiating the variables in $\{C_c \setminus C_p\}$, for each of the legal values of C_p, in a depth-first manner. Clearly, the space required is proportional to the number of values which are finally stored (and some negligible stack overhead). In the worst case, the running time is proportional to the total number of possible assignments to the variables in C_c.

In general, however, the actual number of values enumerated will be smaller because the legal values discovered for C_p will be a subset of the total possible values. Furthermore, if we instantiate the variables in $\{C_c \setminus C_p\}$

```
Algorithm Identify_Legal_Values()
    for each clique C_i in preorder
        if C_i is the root C_r
            (*) Enumerate all the values j for which P(C_r = j) ≠ 0.
        else
            let C_p be C_i's parent
            if C_i is a separator
                For each legal value k of C_p which represents a distinct instantiation
                of the variables in {C_i ∩ C_p}, store a legal value j for C_i.
                Note that each parent value k maps onto a distinct child value
                j = mapping(k), and P(C_i = mapping(k)|C_p = k) = 1.
            else
                for each legal value k of C_p
                    (*) Enumerate the assignments to the variables in {C_i \ C_p}, and for each
                        Generate a value j corresponding to the now complete
                        assignment of values to C_i's members.
                        Store a legal value if P(C_i = j|C_p = k) ≠ 0.
```

Figure 18: Identifying the legal values of each clique.

in topological order, the enumeration procedure can be pruned as soon as there is some variable $X_i \in \{C_c \setminus C_p\}$ for which $P(x_i|Parents(X_i)) = 0$. This is true even for non-deterministic variables.

3.4.2 Clique Tree Inference

Let d_{C_i} denote the degree of clique i; i.e. the total number of edges incident on C_i including both incoming and outgoing edges. Let s_{C_i} denote the total number of possible values C_i can take. Once the possible clique values are enumerated, an examination of the inference loops reveals that the running time of the actual inference procedure is $O(\sum_{C_i \in Non-Separators} d_{C_i} s_{C_i})$. This work is about evenly distributed between λ and π calculations.

This time bound is slightly more precise than, but in the worst case the same as, the the bound presented in (Lauritzen & Spiegelhalter 1988). This bound cannot be improved on by any procedure in which each clique transmits information to all its neighbors.

3.5 Online Decoding

In many applications, it is desirable to process a continuing stream of data in an online fashion. For example, in speech recognition it is necessary to recognize words in real time, before a person has finished speaking. In these types of applications, it is not possible to construct a complete clique tree to represent the utterance, because the total number of frames is unknown and too long a delay would be imposed by waiting until the end. Under these circumstances, it is useful to do a Viterbi decoding in an online fashion.

Under some circumstances, there is an additional constraint: the number of possible clique values may be too large to handle in real time, or with the available memory. In speech recognition, this occurs when a Bayesian network is used to encode an entire language model, and tens of thousands of words are possible at any point in time. Under these circumstances, it is necessary to prune the set of hypotheses, and keep track only of ones that are reasonably likely. Typically a beam search is used, and this is straightforward to incorporate into the forward pass of HMM inference (see, e.g. (Jelinek 1997)).

With DBNs, however, the situation is more tricky. Recall that the likeliest assignment of values to the variables is computed with a *bottom-up* pass over the tree. Since the root of the tree must be at time 0 in order to handle deterministic variables, this creates a problem: the bottom of the tree is extended as each new frame becomes available. This means that the Viterbi decoding must be recomputed from scratch for the entire utterance after each frame. Clearly, this is prohibitive.

There are two obvious ways around this, neither of which works:

1. Compute the likeliest assignment of values in a top-down manner, rather than bottom-up. This does not work because the πs must be computed with reference to the λs, which are not available until the bottom-up pass is complete.

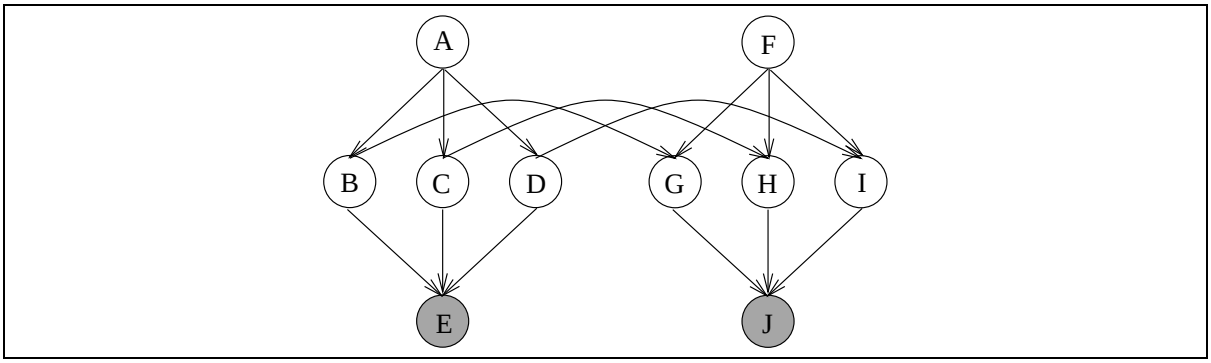

Figure 19: Two slices of a complex DBN; when reduced to a chain-structured tree, the computational requirements are significantly lower than if a cross-product of the state values were used in a an HMM.

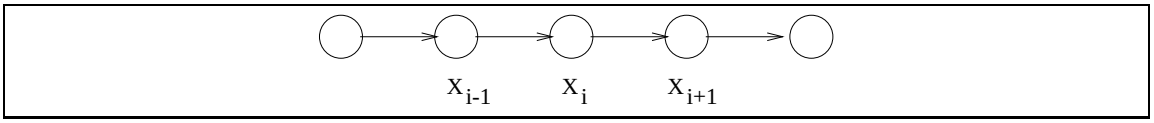

Figure 20: A chain-structured model.

2. Modify the tree-building procedure so that the bottom of the tree - rather than the root - is at time 0. This has the flaw that it becomes extremely cumbersome to keep track of the clique values that are possible, given the constraints of deterministic variables. Recall that this was done in a top-down manner, starting at the root.

However, there are at least two ways of overcoming these difficulties. The simpler of these methods, chain decoding, imposes a strong constraint on the clique tree, but is extremely simple and is discussed here. The more complex, "Backbone Decoding," imposes no constraints and is described in (Zweig 1998).

3.5.1 Chain Decoding

The first method comes from realizing that with chain-structured DBNs, it is possible to compute the likeliest values with a forward sweep. When the Bayesian network is a chain, the computation of π_j^i reduces to:

$$\pi_j^i = \sum_{v \in CON(e_p^0)} \pi_v^p P(X_i = j | X_p = v).$$

Since there is no reference to anything that must be computed in a bottom-up pass (specifically to any λs), this quantity can be computed online in a top-down fashion. By maximizing over v, rather than summing, and keeping track of the best v for each π_j^i, the likeliest assignment of values can also be recovered online. This is exactly analogous to online Viterbi decoding with HMMs (Rabiner & Juang 1993; Jelinek 1997). Finally, beam search can be implemented by pruning away the least likely π_j^is after each new frame. Since this is a top-down procedure and the root of the tree is still at time 0, it is straightforward to combine with the procedure for propagating deterministic constraints.

It is important to realize that in terms of computational requirements, a chain-structured DBN may be significantly more efficient than an HMM in which the state space represents the cross-product of all the variables in a timeslice (even though both can be represented by chain structures). This is because it is possible to "spread" the variables from a single slice across many cliques in the chain. Figure 19 shows an example of this. A valid chain of cliques is:

```
{A,B,C,D} {B,C,D} {B,C,D,E} {B,C,D} {B,C,D,F,G} {C,D,F,G}
{C,D,F,G,H} {D,F,G,H} {D,F,G,H,I} {G,H,I} {G,H,I,J}
```

Assuming that each of the hidden variables has f values, the compute time is proportional to f^5. This compares favorably to the f^8 requirement of a cross-product HMM, which results from the necessity of considering a transition from any of f^4 states at time t to any of f^4 states at time $t+1$. In general, if there are k hidden variables in the middle layer, the compute time is f^{k+2} as opposed to f^{2k+2}.

This particular form of inference is also quite well suited for pruning: one simply defines a threshold γ and modifies the forward pass to be:

$$\pi_j^i = \sum_{v \in Con(e_p^0), \pi_v^p > \gamma} \pi_v^p P(X_i = j | X_p = v)$$

The process for creating clique chains is also extremely simple. In particular, it is possible to use the *frontier algorithm* (Zweig 1996), which proceeds in the following steps:

1. Identify a topological ordering of the nodes.
2. Create an initial "frontier" clique containing the first variable in the ordering.
3. Repeat until all variables have been added and removed from a clique:
   (a) if there is a variable in the frontier whose children are also all present, then form a new frontier by removing it.
   (b) otherwise form a new frontier by adding the next variable in the ordering.
4. Remove any clique that is a subset of another.
5. Insert separator cliques.

For the example of Figure 1, the sequence of cliques would be:

```
{A} {A,B} {A,B,C} {B,C} {B,C,D} {C,D} {C}, {C,E} {E}
```

After step 4, the set of maximal cliques would be:

```
{A,B,C} {B,C,D} {C,E}
```

And after the separators are added in step 5, the final chain is:

```
{A,B,C} {B,C} {B,C,D} {C} {C,E}
```

3.6 EM

The λ and π quantities that are computed during inference can be used to do parameter estimation via EM. In the case of discrete conditional probability tables, the crux of the EM algorithm is extremely simple. It works in terms of the number of times families of variables are seen with specific values. If we let N_{ijk} be the number of times X_i is seen with value k and its parents have instantiation j, then we estimate θ_{ijk}, the probability that $X_i = k$ given that its parents have instantiation j, as $\frac{N_{ijk}}{\sum_k N_{ijk}}$ (Lauritzen 1991; Heckerman 1995). When both a variable and its parents have observed values, N_{ijk} is obtained simply by counting. More commonly, there are some unknown values in which case inference is necessary. The calculations can be summarized as follows.

Let C_i be the clique containing x_i and its parents. Because of the **MORAL** property, we know such a clique exists. Let $\mathbf{V}_{jk}^i$ be the set of C_i's clique values corresponding to underlying variable assignments that include $X_i = k, Parents(X_i) = j$. Recall that

$$P(C_i = w | Observations) = \frac{\lambda_w^i * \pi_w^i}{\sum_w \lambda_w^i * \pi_w^i}$$

Now, N_{ijk} can be found by summing over the appropriate clique values:

$$N_{ijk} = \sum_{w \in \mathbf{V}_{jk}^i} P(C_i = w | Observations)$$

Estimating N_{ijk} from a collection of examples simply requires summing the individual estimates for each example. By maintaining the appropriate data structures, the counts for every family can be computed in a single sweep over the cliques. When the observations are real-valued, this approach can be generalized, analogous to HMMs, to collect the sufficient statistics required for Gaussian re-estimation.

achieved	apex	arsenic	ashtray	ashwell
awe	barleycorn	beeswax	belgium	biff
bloodletting	boyish	breathes	broadview	cashways
chadwick	cheesecloth	colorfast	compromise	confession
cowling	craigs	disrespectful	echoes	egghead
exhaustion	festival	formalization	grisly	handlooms
haymarket	highman	humdinger	humphreys	immobilizing
impolite	indictments	inscribe	instincts	ivy
lavender	lawmakers	majorities	masks	mckane
mutually	mysteriously	nonstandard	noose	nothingness
overambitious	penguins	perm	plowing	porch
postmark	rustle	salesmen	sluggishness	soggy
sorceress	spokeswoman	staying	subgroups	sulfuric
swordcraft	theory	undeterred	unleashes	unnatural
vulgarity	watchful	windowless	windshield	youngman

Table 1: Typical words in the Phonebook database.

4 Speech Recognition Experiments

This section presents experimental results for a fully implemented Bayesian network speech recognition system. Early on, computational limitations forced a choice between experimenting on a fairly small database of digits or alphabet-letters, with the ability to test relatively complex network structures, or experimenting on a more challenging database with simpler network structures. In order to get more meaningful results, we chose the latter, and selected a large-vocabulary multi-speaker database of isolated words to use as a testbed. The database is challenging enough that results are significant, yet because it has isolated words, it avoids many issues that complicate continuous-speech ASR systems. In short, the database is just complex enough to test some basic issues relating to the use of factored state representations in ASR.

4.1 Database

This chapter presents results for the Phonebook database (Pitrelli *et al.* 1995). Despite its name, the database does not contain entries from a phonebook; instead, it consists of a collection of words chosen to exhibit all the coarticulatory effects found in the English language. Researchers at NYNEX compiled the list of words, and then contracted with an outside organization to obtain actual utterances. These were collected over the telephone, and thus contain a variety of transmission distortions. The database is divided into subsets, and the words in one of these subsets are reproduced in Table 4.1.

Word utterances were collected from a group of American speakers who were "balanced for gender, and demographically representative of geographic location, ..., income, age (over 18 years), education, and socio-economic status." (Pitrelli *et al.* 1995) Each speaker was asked to read 75 or 76 words, and the utterances were then screened for acceptable pronunciation; therefore, there are somewhat fewer than 75 words per speaker on average.

4.2 Acoustic Processing

The utterances were processed with relatively conventional acoustic processing. Initial and final silence was removed using the endpoints provided with the database. Then the utterances were divided into 25ms windows and MFCCs were calculated. The analysis windows overlapped by 2/3.

Smoothed MFCC derivatives (Rabiner & Juang 1993) were computed, and three data streams were created: one for the MFCCs, one for the derivatives, and one for the combination of C_0 and delta-C_0 (which were omitted from the first two streams). Mean cepstral-subtraction (Mammone *et al.* 1996) was performed for cepstral coefficients $C_1 - C_{10}$, and speaker normalization (Lee 1989) was done for C_0. The first process removes the effects of telephone transmission characteristics, and the second subtracts the maximum C_0 value in an utterance from each frame, in order to make the values comparable across speakers.

The data streams were vector-quantized to eight bits (256 values). The MFCCs and delta-MFCCs were quantized in separate codebooks. C_0 and delta-C_0 were quantized to four bits each, and then concatenated to form a single eight-bit stream.

4.3 Phonetic Alphabets

Results are presented for DBN models using both context-independent and context-dependent phonetic units. The Phonebook database provides phoneme-level transcriptions, and these formed the basis of both kinds of alphabet. To keep the number of parameters reasonable, and in common with other work (Dupont *et al.* 1997), we did not use the 3-way stress distinction for vowels. We used 41 basic phonemes, plus initial and final silence units.

4.3.1 Context Independent Alphabet

In the case of context-independent units, i.e. simple phonemes, each phoneme was replaced by a k-state left-to-right phone model. Most of the experiments report results for 4-state phone models; these have an initial and final state, and two interior states. Experimentation shows this to be a good number. In all cases, one-state models were used to represent silence.

4.3.2 Context Dependent Alphabet

To create a context-dependent alphabet, we used a variation on diphones (Schwartz *et al.* 1980). The basic idea is to create two new units for each phoneme in a transcription: one for the initial part of the phoneme in the left-context of the preceding phoneme, and one for the final part of the phoneme in the right-context of the following phoneme. Thus, for example, $/k\ ae\ t/$ becomes

$$(sil\ \mathbf{k})(\mathbf{k}\ ae)(k\ \mathbf{ae})(\mathbf{ae}\ t)(ae\ \mathbf{t})(\mathbf{t}\ sil).$$

This scheme has the advantage of addressing both left and right contexts, like triphones, while only squaring - rather than cubing - the number of potential phonetic units.

To prevent overtraining, a context-dependent unit was used only if it occurred a threshold number of times in the training data. Units that did not meet this criterion were replaced with context-independent units. We used thresholds of 250 and 125, which resulted in alphabets with sizes of 336 and 666 respectively, including a full set of context-independent units.

We found it beneficial to double the occurrence of each of the units in a context dependent transcription. Thus, the total number of phones is four times the original number of phonemes, the same number that results from four-state context-independent phoneme models. Repeating phonetic units has the effect of changing the minimum and expected state durations.

It is important to realize that context as expressed in a context-dependent alphabet is significantly different from that represented by a hidden context variable in a Bayesian Network. Context of the kind expressed in a context-dependent alphabet is based on an idealized and invariant pronunciation template; a word model based on context-dependent phones can be written down before ever seeing a sound wave, and therefore represents a-priori knowledge. The context-variable represents context as manifested in a specific utterance.

4.4 Experimental Procedure

4.4.1 Training, Tuning, and Testing

The database was divided into separate portions for training, tuning, and testing. All decisions concerning network structures and alphabets were made by training a system on the training data, and testing it on the tuning data. Decisions were not made on the basis of performance on the actual test data.

The training data consisted of the utterances found in the *a, *h, *m, *q, and *t subdirectories of the Phonebook distribution; the tuning utterances were from the *o and *y directories, and the test utterances from the *d and *r directories. This is the same partitioning used in (Dupont *et al.* 1997). This resulted in $19,421$ training utterances, $7,291$ tuning utterances and $6,598$ test utterances, with no overlap between the speakers or words in any of the partitions.

The words in the Phonebook vocabulary are divided into 75 and 76-word groups, and the recognition task consists of identifying each test word from among the other words in its subset. Hence, random guessing would result in a recognition rate of under 2%.

4.4.2 Models Tested

A baseline Bayesian network was constructed to emulate an unfactored HMM. Bayesian networks with one or more auxiliary state variables were then designed to answer the following questions:

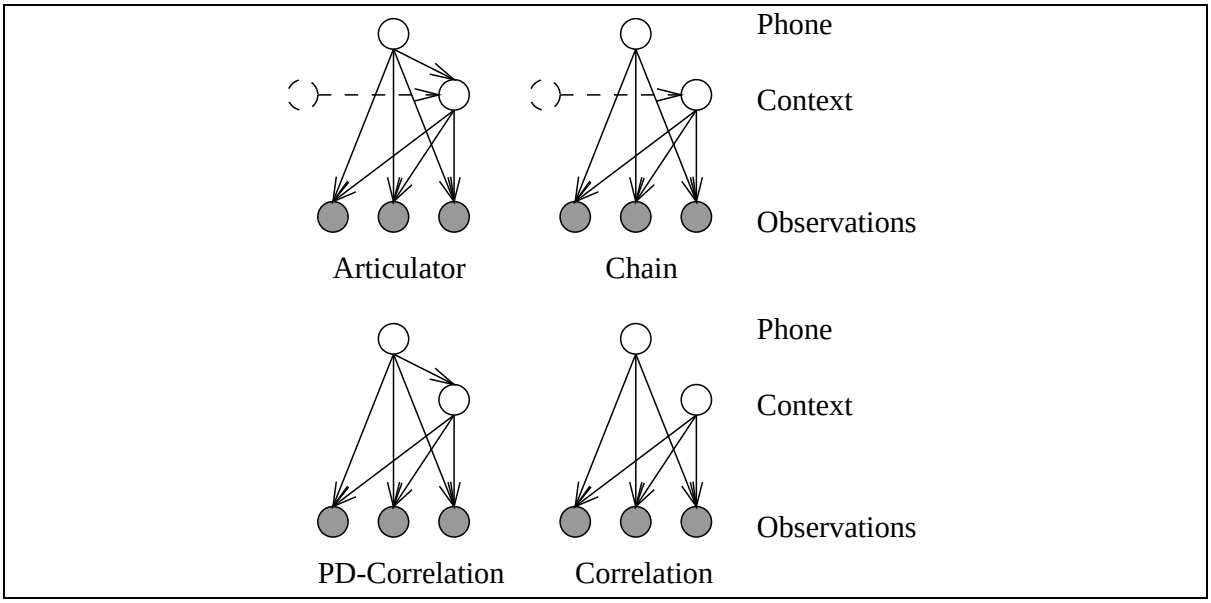

Figure 21: The acoustic models for four of the network topologies tested. The dotted lines indicate conditioning on the previous frame.

1. What is the effect of modeling correlations between observations within a single timeslice? Specifically,

   (a) What is the effect of modeling these correlations in a phone-independent way? This question was addressed with the "Correlation" network of Figure 21. This network is only capable of modeling intra-frame observation correlations in the most basic way.

   (b) What is the effect of modeling these correlations in a phone-dependent way? This question was addressed with the phone-dependent "PD-Correlation" network of Figure 21. This network can directly model phone-dependent intra-frame correlations among the acoustic features.

2. What is the effect of modeling temporal continuity? Specifically,

   (a) What is the effect of modeling temporal continuity in the auxiliary chain in a phone-independent way? This question was addressed with the "Chain" network of Figure 21. This network results from the addition of temporal links to the context variable of the correlation network, and can directly represent phone-independent temporal correlations. The network was initialized to reflect continuity in the value of the context variable.

   (b) What is the effect of modeling temporal continuity in the auxiliary chain in a phone-dependent way? This was addressed with the "articulator" network of Figure 21. In this network, the context variable depends on both the phonetic state and its own past value. This can directly represent phone-dependent articulatory target positions and inertial constraints. The network was initialized to reflect voicing.

3. How does the use of a context-dependent alphabet compare to context-modeling with an auxiliary variable? This was addressed by using a context-dependent alphabet in a network with no auxiliary variable. Additionally, we tested the combination of a context-dependent alphabet with a context variable.

4. What is the effect of increasing the number of values in the auxiliary chain, and how does increasing this number compare to increasing the number of context variables? This question was answered by making the proposed changes to the chain-network.

5. What is the effect of using an unfactored state representation to represent the same process? To answer this question, a Bayesian network was used to emulate an HMM with an unfactored representation.

6. What is the effect of doing unsupervised clustering?

Network	Parameters	Error Rate
Baseline-HMM	127k	4.8%
Correlation	254k	3.7%
PD-Correlation	254k	4.2%
Chain	254k	3.6%
Articulator	255k	3.4%

Table 2: Test set word error rate for systems using the basic phoneme alphabet. All the systems had slightly different numbers of parameters. The standard error is approximately 0.25%. Results from Zweig & Russell, 1998.

Network	Parameters	Error Rate
CDA-HMM	257k	3.2%
CDA-Articulator	515k	2.7%
CDA-HMM	510k	3.1%

Table 3: Test set word error rates for systems using context dependent alphabets. The first two results use an alphabet with 336 units, and the last result uses an alphabet with 666 units. The standard error is approximately 0.20%. Results from Zweig & Russell, 1998.

4.5 Results with a Single Auxiliary Variable

Answers to the first three questions are presented in this section. Table 2 shows the word-error rates with the basic phoneme alphabet, for the network structures shown in Figure 21. The results for the Bayesian networks with a context variable are consistently better than without a context variable. A large improvement results simply from modeling within-frame correlations, but for both the Correlation and the PD-Correlation networks, a further improvement results from the addition of temporal-continuity links. The Articulator network provided the best performance.

In terms of absolute error-rates, these results compare favorably with those reported in (Dupont *et al.* 1997). That paper reports an error rate of 4.1% for an ANN-HMM hybrid using the Phonebook transcriptions, and the same training and test sets. However, with phonetic transcriptions based on the CMU dictionary, (Dupont *et al.* 1997) achieved significantly improved results. Comparison with this work provides a useful check on the overall recognition rate, but it should be remembered that a vector-quantized system is being compared with a continuous-observation system. Thus differences are difficult to interpret.

4.5.1 Context Dependent Alphabet

Error rates with the context-dependent alphabets are reported in Table 3. These results improve significantly on the context-independent results, and (as expected) confirm the benefits of using a context-dependent alphabet. As discussed previously, context as represented in an alphabet is different from context as represented in a Bayesian network with a context variable. Therefore it makes sense to combine the two strategies, and this in fact produced the best overall results. Adding an extra context variable doubled the number of parameters, but as the last line in Table 3 indicates, doubling this number by using a bigger alphabet is not as effective.

4.6 Results With Two Auxiliary Variables

In order to evaluate the use of multiple context chains, the network shown in Table 22 was tested. Both of the context variables were binary. For comparison, a network with a single context variable with 3 and 4 values was tested. To cut down on memory usage and running time, and to save on the amount of disk-space needed to store conditional probabilities, 3-state phones were used in this set of experiments. The results are shown in Table 4.

These results indicate that increasing the amount of context state improves recognition performance. In addition, factoring the context state is beneficial. The results are somewhat worse than with the four-state phone models, which indicates that the combination of greater precision and longer minimum durations afforded by the four-state models is important.

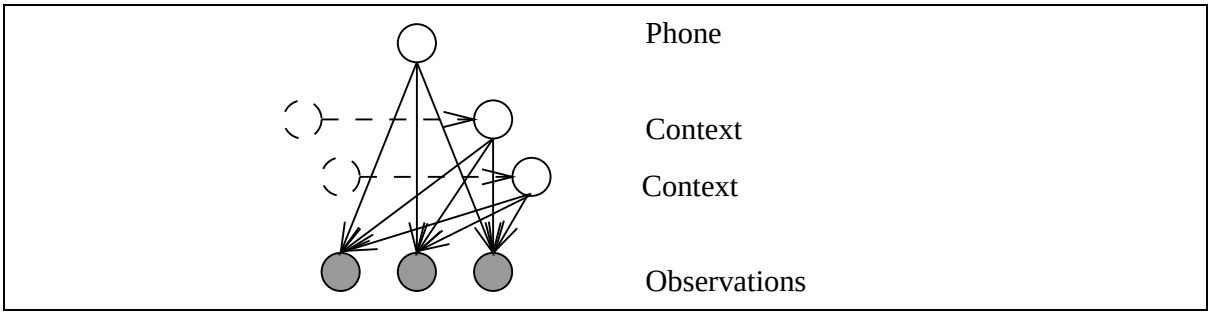

Figure 22: Network with two context variables.

Network	Parameters	Error Rate
Binary-Chain	191k	4.1%
Trinary-Chain	287k	4.0%
Quaternary-Chain	383k	3.8%
Double-Chain	383k	3.6%

Table 4: Test results with multi-valued and multi-chain context variables; the standard error is approximately 0.25%. The double-chain network used binary variables, and thus had a total of 4 possible context values.

4.7 Clustering Results

This section presents results for a network doing unsupervised clustering. The network structure is presented in Figure 10. A binary-valued context variable was used, with the restriction that its value not change over the course of an utterance. This was enforced by using a stochastic context variable in the first timeslice, and then using a deterministic context variable from the second timeslice on to copy the value determined in the first frame. The network was trained as usual, and then during testing the likeliest value for the cluster variable was determined.

There are at least two dimensions along which one might expect clustering to occur: the type of speaker (e.g. male vs. female, adult vs. child), and the type of word (e.g. consonant-initial vs. vowel initial). [1] The degree to which such clustering occurs can be measured by looking at the degree to which utterances with a particular characteristic are classified together in a single cluster. Figures 23 and 24 show that both speaker and word clustering are observed. Since there are more cross-validation utterances than test utterances, the histograms are based on that subset (no tuning was involved).

Figure 23 shows the consistency with which utterances from a single speaker were classified together, and what would be expected at random. Clearly, utterances from individual speakers are being grouped together with high frequency.

Figure 24 shows the same information for particular words. The fact that the occurrences of a single word tend to be clustered together indicates that word characteristics, as well as speaker characteristics, are being modeled by the auxiliary variable.

In terms of overall error-rate, the clustering technique did not do as well as the other augmented networks; the test-set error rate of 4.5% was midway between the 4.8% rate of the baseline network and the 3.6% score for the chain network. This is expected, since the cluster-network has more state, and therefore modeling power, than the baseline network, but not as much expressiveness as the chain-network, where the context variable can "flip-flop" between values.

It is possible to make sense of the value assigned to the cluster variable, both in terms of speaker-type and word-type. The mutual information between the cluster variable and the gender of the speaker is 0.24 bits, indicating a strong correlation between cluster and gender. To determine the word-characteristics associated with the two clusters, we examined the words that were very consistently assigned to a particular cluster. These are shown in Figure 5. One of the clusters is characterized by words beginning in liquid consonants, while the other is characterized by words ending in liquid consonants. The cluster with words beginning in liquid consonants also happens to be associated with female speakers; the cluster with terminal liquid consonants is associated with male speakers. Note, however, that since each word was spoken by approximately as many men as women, word-clustering comes at the expense of gender-clustering.

[1] Thanks to Jeff Bilmes for pointing out the duality between speakers and words.

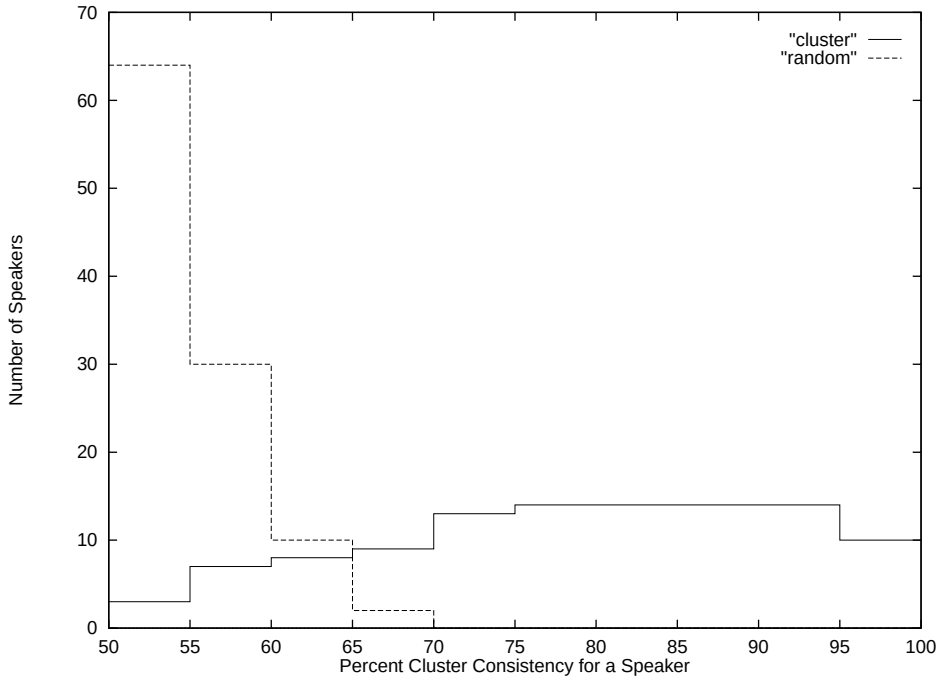

Figure 23: The frequency with which utterances from a single speaker were assigned to the same cluster. For example, about 15 speakers has their utterances clustered together with 85% consistency. On average, there are 68 utterances per speaker.

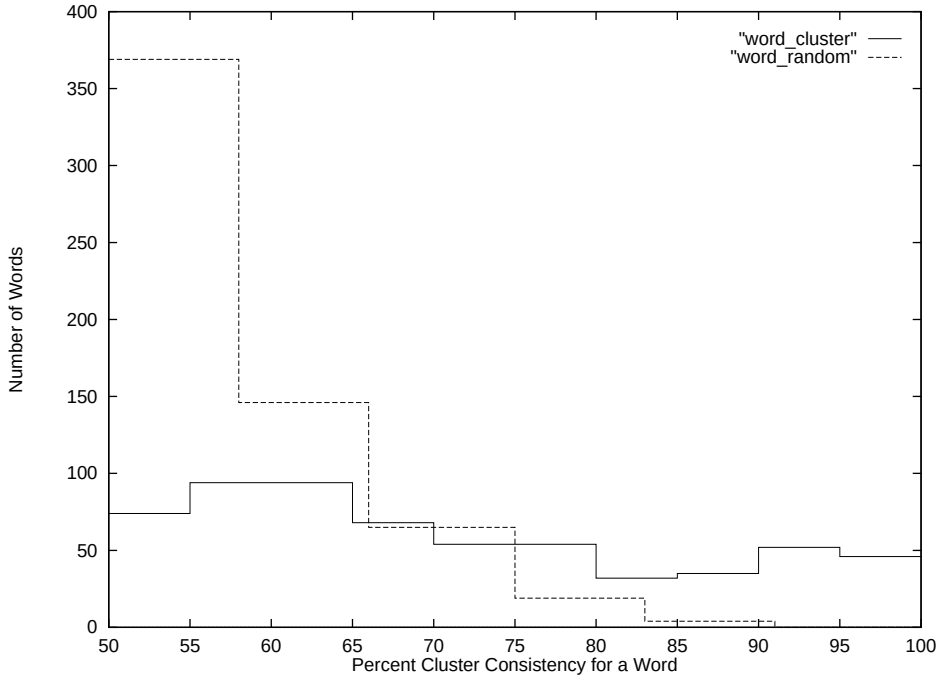

Figure 24: The frequency with which utterances of a single word were assigned to the same cluster.

Cluster 1			Cluster 2	
aboveboard	elsewhere	incapable	bathing	irving
mainville	melrose	oval	landberg	laundromat
store	unapproachable	ungovernable	lifeboat	livelihood
visual	whipples	arrivals	citizend	floodgates
gospels	salesroom	scarsdale	increasingly	motown
forced	starched	summerall	negligently	plaintiff
astronomical	bakeware	bridgeforth	redness	spacelink
fairchilds	geographical	gulps	implicitly	mcbee
mistrustful	pinwheel	quails	engagingly	heaves
torso	unforgivable	unusual	honda	nape
waffles	carlson	pathological	eighths	included
unborn	untraveled	westwall	lancelet	nat
strolls	totals	allies	peanut	lindsey
beagle	cashdrawer	dialed	cupcakes	woodlawn
reels	seldom	silverstone		
squirreled	tranquil	unethical		
isabell	spoilt	unquestionable		
foghorn	bale	unawares		
dimsdale	heartfelt	sparkled		
pebbles	seafowl	bulls		
baffled	dolphin	squabble		
westworld				

Table 5: The words that occurred in a particular cluster more than 90% of the time. About half the words in the first cluster end in liquid consonants (/l/ or /r/), even more if terminal /s/ is allowed. For example, "unapproachable" and "astronomical." None of the words in the second cluster end in liquid consonants. Instead, about a quarter of them *begin* with liquid consonants, e.g. "lifeboat" and "laundromat." Only one of the words in the first cluster, "reels," begins with a liquid consonant.

4.8 Discussion

4.8.1 Improvements

In every case that an auxiliary variable was used, there was a performance increase. Moreover, increasing the modeling power of the network by increasing the amount of context state produced further improvements. This indicates that context modeling with auxiliary state information is an effective way of decreasing speech-recognition error rates.

The context variable was able to capture several different phenomena, ranging from simple correlations between the observations within a single frame to gender and word-specific characteristics. In general, networks in which the context variables were linked across time did better than corresponding networks without temporal links. This is unsurprising because neither acoustic nor articulatory properties are expected to change rapidly over time; this is expected for the articulators because of physical inertia, and for acoustics both because they are generated by articulators, and because the acoustic features are generated from overlapping frames of speech.

A context-sensitive alphabet was an effective way of improving performance, but here too an auxiliary variable was beneficial. This makes sense because context-dependent alphabets encode prior knowledge about coarticulatory effects, but do not pay attention to the particulars of any specific utterance. An auxiliary variable has the ability to encode information on a case-by-case basis.

4.8.2 What Does it Mean?

In order to understand the meaning of the context variable, we examined its correlation with the different acoustic features, and found that it is highly correlated with the combined C_0 and delta-C_0 observation stream. This relation is graphed for four different network structures in Figure 25. Roughly speaking, C_0 is indicative of the overall energy in an acoustic frame. The maximum value in an utterance is subtracted, so the value is never greater than 0. Assuming that each frequency bin contributed an equal amount of energy, the C_0 range corresponds to an energy range of 50db. This figure shows that despite similar word-error rates, the different network structures worked by learning different kinds of patterns in the data.

A second important pattern that emerges is that in the networks with time-continuity arcs, the context variable

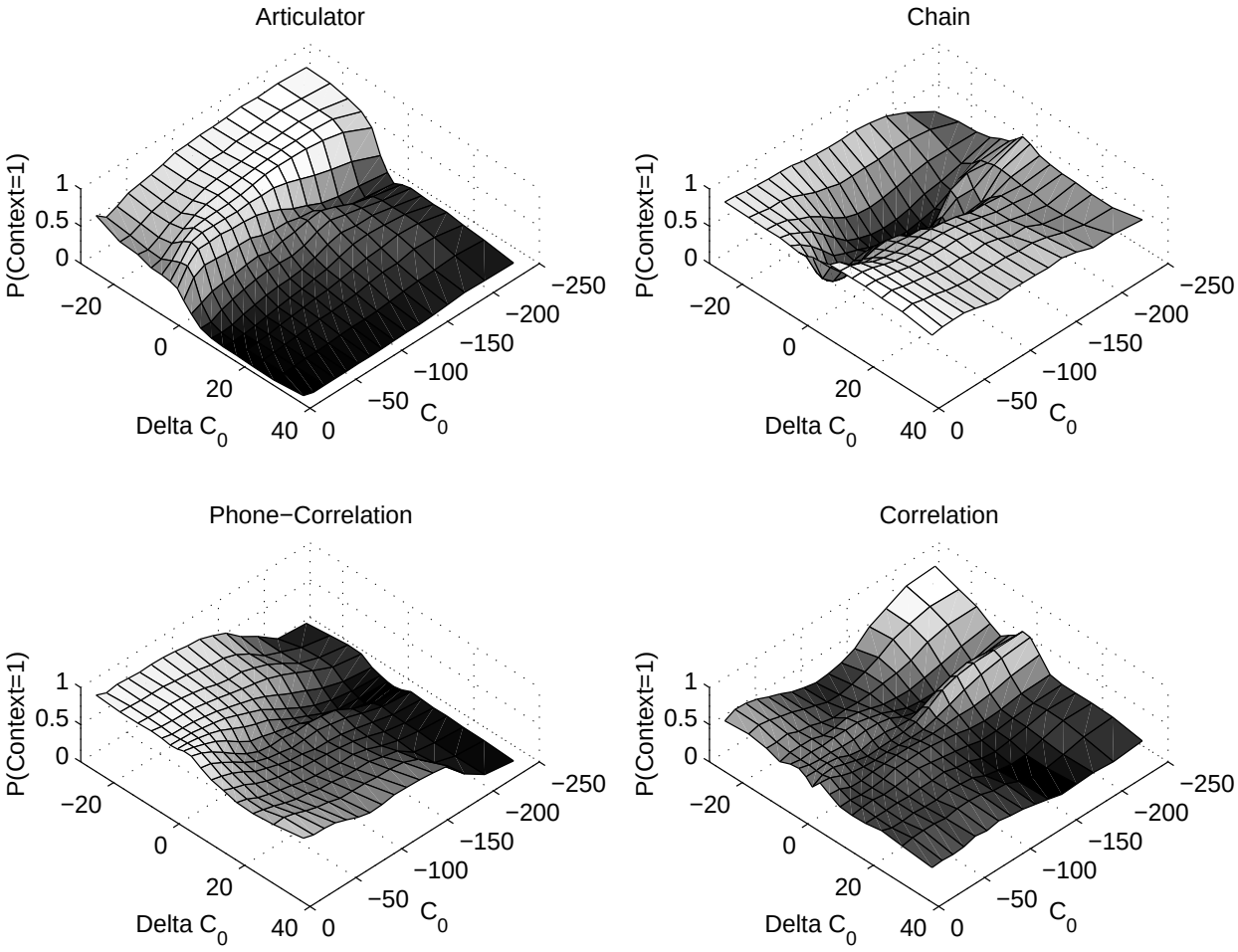

Figure 25: Probability that the context variable has the value 1 as a function of C_0 and delta-C_0.

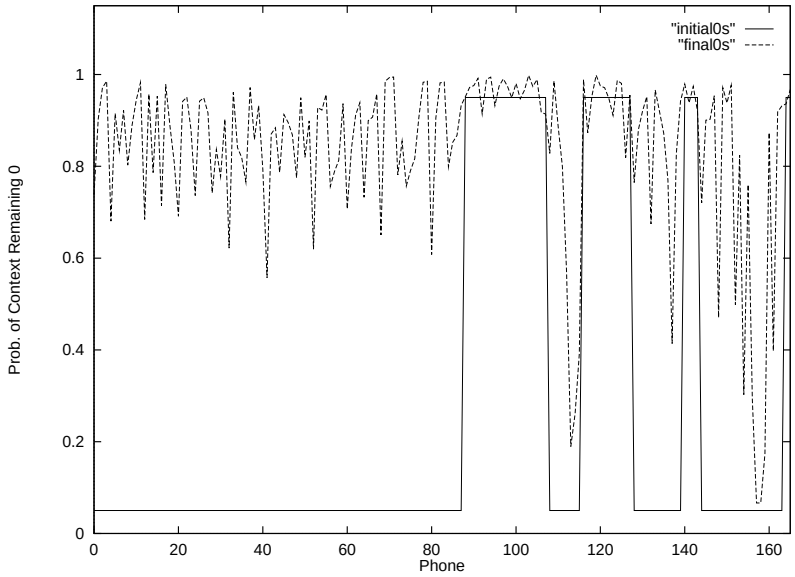

Figure 26: Learning continuity. The lines show $P(C_t = 0|C_{t-1} = 0, Q_t = p)$, i.e. the probability of the context value remaining 0 across two frames of speech, as a function of phone. The solid line is before training, and the dotted line is after training. The context variable represents voicing, so values close to 1.0 are for voiced phones. After training, the context value is unlikely to change, regardless of phone. This reflects temporal continuity.

Network	Corr.	PD-Corr.	Chain	Art.	CDA-257	CDA-Art	CDA-510
Base	65	69	49	50	35	32	29
Corr.		68	55	55	37	34	32
PD-Corr.			51	50	38	34	32
Chain				56	44	38	32
Art.					41	38	33
CDA-257						53	32
CDA-Art							44

Table 6: Percent similarity in the errors made by pairs of recognizers. If A and B are the sets of words the systems respectively got wrong, similarity is defined as $100\frac{|A \cap B|}{|A \cup B|}$.

is characterized by a high degree of continuity. To demonstrate that the networks will tend to learn continuity, an experiment was made in which the articulator network was initialized to reflect voicing in a very extreme way: the probability of a context value of 1 was set almost to 1 (regardless of its previous value) for voiced phones, and almost to 0 for unvoiced phones. The EM procedure was then applied, and in the learned parameters, the striking characteristic is that the context value is almost always unlikely to change. This is consistent with a physical model of a slowly changing inertial process. The initial and learned parameters are shown in Figures 26 and 27. For the results presented in previous sections, the context variable was initialized with less extreme values; this decreased the number of training iterations, and gave slightly better performance.

Although Figures 26 and 27 demonstrate that the context variable displays the continuity which is expected from something that models a physical object, we have been unable to associate it with any specific articulator. In particular, the figures illustrate that the conditional probabilities after training show no clear correlation with the voiced/unvoiced distinction between phones. The EM training procedure has apparently taken advantage of all the degrees of freedom available to it to maximize the data-probability, resulting in parameters that reflect a combination of many underlying factors. Training with data in which the articulator positions are known will likely lead to simpler interpretations of the learned parameters.

From the previous discussion of the context variable's acoustic correlates, it is clear that the different networks learned different patterns in the data. One way of measuring this is to see how similar the errors made by the different networks are. This is shown for several of the networks in Table 6. This indicates that the Correlation and PD-Correlation networks are relatively similar, as are the Chain and Articulator networks. The biggest difference is between networks using a basic phoneme alphabet and those using a context-dependent alphabet.

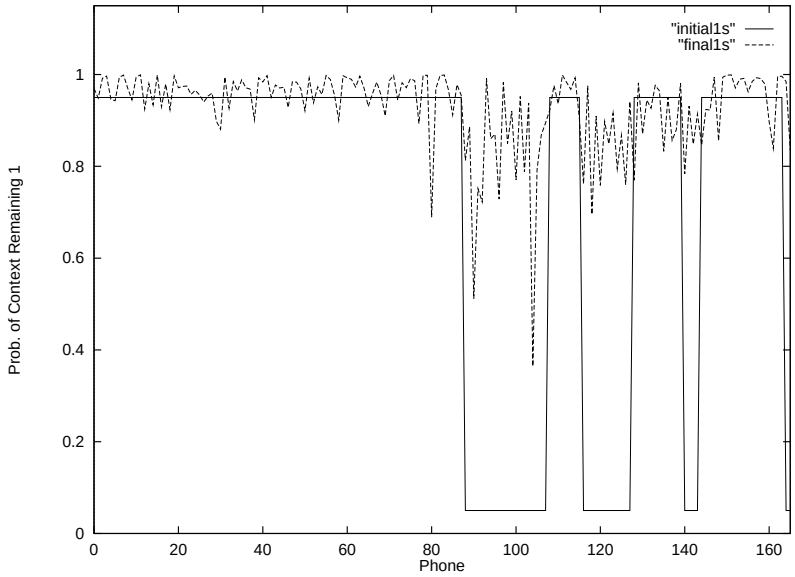

Figure 27: Learning continuity. This graph shows shows that a context value of 1 also shows continuity.

4.9 Perspective

An important tradeoff which is often discussed in the speech recognition literature is between error-rates and the number of parameters used; this is shown in Figure 28 for the networks studied in this chapter. The correlation between the number of parameters used and the error rate is striking, and in this context, the use of Bayesian networks can be understood as a directed and intelligent way of increasing the number of model parameters.

5 Conclusion

In this paper, we have seen that Bayesian networks can be used to model a wide variety of phenomena that occur in speech production and recognition. Using the two concepts of random variables and conditional probabilities, we are able to express many of the a-priori constraints that exist in a recognizer, e.g. baseform spellings, as well as physiologically-based phenomena such as coarticulation. In addition to expressive power, Bayesian networks are endowed with a set of general purpose algorithms that allow for parameter optimization and Viterbi decoding with arbitrary network structures. We have presented these algorithms in terms of extremely straightforward dynamic programming recursions, and extended them to efficiently handle the deterministic constraints that often occur amongst variables in a speech-recognition network.

6 Appendix: Proof of τ Speedup

The proof of correctness is inductive. We begin by noting that a π computed by X_i is only used by X_i and its children, and therefore only relevant to those variables. We will show that

1. The only way an error can be made is in the calculation of a child's π value.
2. Either the child's πs are correctly calculated, or the affected terms have τ-factors which themselves are 0. This implies an inductive chain in which πs can only be miscalculated for leaf variables. But since errors can only occur in a child's π, and a leaf variable has no children, the calculation is sound.

The proof is given in more detail below.

Theorem 6.1 *The omission of terms for which* $\tau_v^i = \sum_f \lambda_f^i * P(X_i = f | X_p = v) = 0$ *is irrelevant.*

Proof

Without loss of generality, consider the calculation of π_j^i:

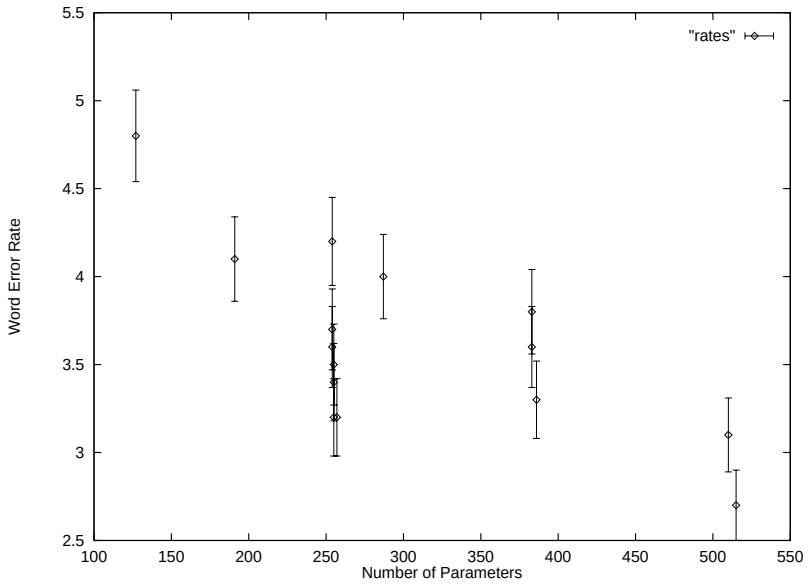

Figure 28: Error rate as a function of the number of network parameters. The errorbars represent one standard deviation in either direction.

$$\pi_j^i = \sum_{v \in CON(e_p^0)} P(X_i = j | X_p = v) * \pi_v^p * \prod_{s \in siblings(X_i)} \sum_f \lambda_f^s * P(X_s = f | X_p = v).$$

Suppose a value of v is encountered for which

$$\tau_v^i = \sum_f \lambda_f^i * P(X_i = f | X_p = v) = 0.$$

First note that $\sum_f \lambda_f^i * P(X_i = f | X_p = v) = 0$ implies $\lambda_f^i * P(X_i = f | X_p = v) = 0$, $\forall f$. In particular, $\lambda_j^i * P(X_i = j | X_p = v) = 0$, so either $\lambda_j^i = 0$ or $P(X_i = j | X_p = v) = 0$.

If $P(X_i = j | X_p = v) = 0$, the product over siblings is irrelevant because it will be multiplied by 0. Omitting the term does not result in error, and neither values associated with X_i nor its children will be affected.

If $\lambda_j^i = 0$ we must consider first the effects of miscomputing π_j^i on future calculations regarding both X_i and its children. There are no effects on future calculations regarding X_i because whenever π_j^i is used (e.g. to calculate marginals) it will be multiplied by λ_j^i.

Thus far we have shown that omitting the term can only affect future calculations regarding X_i's children, and never calculations regarding X_i itself. This implies that if X_i is a leaf, the omission is safe. Furthermore, the children can only be affected when $\lambda_j^i = 0$.

Now suppose $\lambda_j^i = 0$ and consider a child X_c computing some π value π_w^c.

$$\pi_w^c = \sum_{v \in CON(e_i^0)} P(X_c = w | X_i = v) * \pi_v^i * \prod_{s \in siblings(X_c)} \sum_f \lambda_f^s * P(X_s = f | X_i = v)$$

Since we are considering the results of a miscomputation of π_j^i, the only effect on the computation of π_w^c occurs when $v = j$. If $j \notin CON(e_i^0)$, there is no contribution to the sum, so a miscomputed π_j^i is irrelevant. Otherwise, the term involved is

$$\begin{aligned} P(X_c = w | X_i = j) * \pi_j^i * \prod_{s \in siblings(X_c)} \sum_f \lambda_f^s * P(X_s = f | X_i = j) \\ = P(X_c = w | X_i = j) * \pi_j^i * \lambda_j^i / \tau_j^c \\ \equiv P(X_c = w | X_i = j) * \pi_j^i * \lambda_j^i / (\sum_f \lambda_f^c * P(X_c = f | X_i = j)) \end{aligned}$$

Since $\lambda_j^i = 0$, we know by its definition that

$$(\sum_f \lambda_f^c * P(X_c = f|X_i = j)) \ * \prod_{s \in siblings(X_c)} \sum_f \lambda_f^s * P(X_s = f|X_i = j) \ = \ 0$$

Either one of the factors over siblings is 0 or the parenthesized factor is 0.

If one of the factors over siblings is 0, the miscomputed value of π_j^i is multiplied by 0, rendering it inconsequential. In the case that the parenthesized factor is 0, we have discovered a term in the calculation of π_w^c for which the τ-factor is 0. This term will be omitted, and we have already shown that omitting it can only affect X_c's children. This establishes an inductive chain in which harmless omissions are made; the chain must terminate at a leaf variable, where the omissions are again harmless.

References

A/S, HUGIN EXPERT. 1995. *HUGIN API Reference Manual*. Hugin Expert A/S.

BAKER, J. 1975. The Dragon system—an overview. *IEEE Transactions on Acoustics, Speech, and Signal Processing* 23.24–29.

BILMES, J. 2000. Factored sparse inverse covariance matrices. In *ICASSP 2000*.

DAGUM, P., & M. LUBY. 1993. Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial Intelligence* 60.141–153.

DENG, L., & K. ERLER. 1992. Structural design of hidden markov model speech recognizer using multivalued phonetic features: Comparison with segmental speech units. *Journal of the Acoustical Society of America* 92.3058–3067.

DUPONT, S., H. BOURLARD, O. DEROO, & J.-M. FONTAINE, V.AND BOITE. 1997. Hybrid HMM/ANN systems for training independent tasks: Experiments on PhoneBook and related improvements. In *ICASSP-97: 1997 International Conference on Acoustics, Speech, and Signal Processing*, 1767–1770, Los Alamitos, CA. IEEE Computer Society Press.

EIDE, E., B. MAISON, S. CHEN, P. OLSEN, D. KANEVSKY, & R.A. GOPINATH. 2000. Ibm's 10xrt system in the 1999 arpa broadcast news evaluation. In *ICSLP 2000*.

ERLER, K., & L. DENG. 1993. Hidden markov model representation of quantized articulatory features for speech recognition. *Computer Speech and Language* 7.265–282.

——, & G. FREEMAN. 1996. An hmm-based speech recognizer using overlapping articulatory features. *Journal of the Acoustical Society of America* 100.2500–2513.

GALES, M.J.F. 1998. Maximum likelihood linear transformations for hmm-based speech recognition. *Computer Speech and Language* 12.

——, & S. YOUNG. 1992. An improved approach to the hidden markov model decomposition of speech and noise. In *ICASSP-92*, I–233–I–236.

GAREY, MICHAEL R., & DAVID S. JOHNSON. 1979. *Computers and Intractability*. New York: W. H. Freeman.

HAIN, T., P.C. WOODLAND, G. EVERMANN, & D. POVEY. 2000. The cu-htk march 2000 hub5e transcription system. In *Proc. Speech Transcription Workshop, 2000*.

HECKERMAN, D. 1995. A tutorial on learning with Bayesian networks. Technical Report MSR-TR-95-06, Microsoft Research, Redmond, Washington. Revised June 1996.

JELINEK, FREDERICK. 1997. *Statistical Methods for Speech Recognition*. Cambridge, Massachusetts: MIT Press.

JENSEN, F., F. V. JENSEN, & S. L. DITTMER. 1994. From influence diagrams to junction trees. In *Uncertainty in Artificial Intelligence: Proceedings of the Tenth Conference*, 367–373, Seattle, Washington. Morgan Kaufmann.

JENSEN, FINN V., STEFFEN L. LAURITZEN, & KRISTIAN G. OLESEN. 1990. Bayesian updating in causal probabilistic networks by local computations. *Computational Statistics Quarterly* 5.269–282.

KNESER, N., & H. NEY. 1995. Improved backing-off for m-gram language modeling. In *ICASSP-95*.

LAURITZEN, STEFFEN L. 1991. The EM algorithm for graphical association models with missing data. Technical Report TR-91-05, Department of Statistics, Aalborg University.

——, & DAVID J. SPIEGELHALTER. 1988. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society* B 50.157–224.

LEE, KAI-FU. 1989. *Automatic speech recognition: The development of the SPHINX system*. Dordrecht, The Netherlands: Kluwer.

LEGGETER, C., & P.C. WOODLAND. 1995. Flexible speaker adaptation using maximum likelihood linear regression. In *Eurospeech-95*.

LUO, X., & F. JELINEK. 1999. Probabilistic classification of hmm states for large vocabulary continuous speech recognition. In *ICASSP 1999*.

MAMMONE, R.J., X. ZHANG, & R.P. RAMACHANDRAN. 1996. Robust speaker recognition. *IEEE Signal Processing Magazine* 58–71.

PEARL, JUDEA. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Mateo, California: Morgan Kaufmann.

PEOT, MARK, & ROSS SHACHTER. 1991. Fusion and propagation with multiple observations. *Artificial Intelligence* 48.299–318.

PITRELLI, J., C. FONG, S. WONG, J. SPITZ, & H. LEUNG. 1995. Phonebook: A phonetically-rich isolated-word telephone-speech database. In *ICASSP-95: 1995 International Conference on Acoustics, Speech, and Signal Processing*, 101–104, Los Alamitos, CA. IEEE Computer Society Press.

RABINER, L. R., & B.-H. JUANG. 1993. *Fundamentals of Speech Recognition*. Prentice-Hall.

RICHARDSON, M., J. BILMES, & C. DIORIO. 2000. Hidden-articulatory markov models for speech recognition. In *ICASSP-2000*.

ROSE, D. J. 1970. Triangulated graphs and the elimination process. *Journal of Mathematical Analysis and Applications* 597–616.

SCHWARTZ, R., J. KLOVSTAD, J. MAKHOUL, & J. SORENSEN. 1980. A preliminary design of a phonetic vocoder based on a diphone model. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 32–35.

SMYTH, P., D. HECKERMAN, & M. JORDAN. 1996. Probabilistic independence networks for hidden Markov probability models. Technical Report MSR-TR-96-03, Microsoft Research, Redmond, Washington.

SPIEGELHALTER, D. J., & S. L. LAURITZEN. 1990. Sequential updating of conditional probabilities on directed graphical structures. *Networks* 20.579–605.

VARGA, A.P., & R.K. MOORE. 1990. Hidden markov model decomposition of speech and noise. In *ICASSP-90*, 845–848.

ZHAN, PUMING, & ALEX WAIBEL. 1997. Vocal tract length normalization for large vocabulary continuous speech recognition. Technical Report CMU-CS-97-148, School of Computer Science, Carnegie Mellon University.

ZWEIG, GEOFFREY. 1996. A forward-backward algorithm for inference in bayesian networks and an empirical comparison with hmms. MS report, Computer Science Division, UC Berkeley.

——, 1998. *Speech Recognition with Dynamic Bayesian Networks*. Computer Science Division, University of California at Berkeley dissertation.