Dynamic Bayesian Networks:
Representation, Inference and Learning

by

Kevin Patrick Murphy

B.A. Hon. (Cambridge University) 1992
M.S. (University of Pennsylvania) 1994

A dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA, BERKELEY

Committee in charge:

Professor Stuart Russell, Chair
Professor Michael Jordan
Professor Peter Bickel
Professor Jeffrey Bilmes

Fall 2002

The dissertation of Kevin Patrick Murphy is approved:

Chair

Date

Date

Date

Date

University of California, Berkeley

Fall 2002

Dynamic Bayesian Networks:

Representation, Inference and Learning

by

Kevin Patrick Murphy

ABSTRACT

Dynamic Bayesian Networks:

Representation, Inference and Learning

by

Kevin Patrick Murphy

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Stuart Russell, Chair

Modelling sequential data is important in many areas of science and engineering. Hidden Markov models (HMMs) and Kalman filter models (KFMs) are popular for this because they are simple and flexible. For example, HMMs have been used for speech recognition and bio-sequence analysis, and KFMs have been used for problems ranging from tracking planes and missiles to predicting the economy. However, HMMs and KFMs are limited in their "expressive power". Dynamic Bayesian Networks (DBNs) generalize HMMs by allowing the state space to be represented in factored form, instead of as a single discrete random variable. DBNs generalize KFMs by allowing arbitrary probability distributions, not just (unimodal) linear-Gaussian. In this thesis, I will discuss how to represent many different kinds of models as DBNs, how to perform exact and approximate inference in DBNs, and how to learn DBN models from sequential data.

In particular, the main novel technical contributions of this thesis are as follows: a way of representing Hierarchical HMMs as DBNs, which enables inference to be done in $O(T)$ time instead of $O(T^3)$, where T is the length of the sequence; an exact smoothing algorithm that takes $O(\log T)$ space instead of $O(T)$; a simple way of using the junction tree algorithm for online inference in DBNs; new complexity bounds on exact online inference in DBNs; a new deterministic approximate inference algorithm called factored frontier; an analysis of the relationship between the BK algorithm and loopy belief propagation; a way of applying Rao-Blackwellised particle filtering to DBNs in general, and the SLAM (simultaneous localization and mapping) problem in particular; a way of extending the structural EM algorithm to DBNs; and a variety of different applications of DBNs. However, perhaps the main value of the thesis is its catholic presentation of the field of sequential data modelling.

DEDICATION

To my parents

for letting me pursue my dream

for so long

so far away from home

&

To my wife

for giving me

new dreams to pursue

ACKNOWLEDGMENTS

I would like to thank my advisor, Stuart Russell, for supporting me over the years, and for giving me so much freedom to explore and discover new areas of probabilistic AI. My other committee members have also been very supportive. Michael Jordan has long been an inspiration to me. His classes and weekly meetings have proved to be one of my best learning experiences at Berkeley. Jeff Bilmes proved to be a most thorough reviewer, as I expected, and has kept me honest about all the details. Peter Bickel brought a useful outsider's perspective to the thesis, and encouraged me to make it more accessible to non computer scientists (although any failings in this regard are of course my fault).

I would like to thank my many friends and colleagues at Berkeley with whom I have had the pleasure of working over the years. These include Eyal Amir, David Andre, Serge Belongie, Jeff Bilmes, Nancy Chang, Nando de Freitas, Nir Friedman, Paul Horton, Srini Narayanan, Andrew Ng, Mark Paskin, Sekhar Tatikonda, Yair Weiss, Erix Xing, Geoff Zweig, and all the members of the RUGS and IR groups.

I would like to thank Jim Rehg for hiring me as an intern at DEC/Compaq/HP Cambridge Research Lab in 1997, where my Bayes Net Toolbox (BNT) was born. I would like to thank Gary Bradski for hiring me as an intern at Intel in 2000 to work on BNT, and for providing me with the opportunity to work with people spanning three countries formerly known as superpowers — USA, China and Russia. In particular, I would like to thank Wei Hu and Yimin Zhang, of ICRC, for their help with BNT. I would also like to thank the many people on the web who have contributed bug fixes to BNT. By chance, I was able to work with Sebastian Thrun during part of my time with Intel, for which I am very grateful.

I would like to thank my friends in Jennie Nation and beyond for providing a welcome distraction from school. Finally, I would like to thank my wife Margaret for putting up with my weekends in the office, for listening to my sagas from Soda land, and for giving me the motivation to finish this thesis.

Contents

Chapter 1

Introduction

1.1 State-space models

Sequential data arises in many areas of science and engineering. The data may either be a time series, generated by a dynamical system, or a sequence generated by a 1-dimensional spatial process, e.g., bio-sequences. One may be interested either in online analysis, where the data arrives in real-time, or in offline analysis, where all the data has already been collected.

In online analysis, one common task is to predict future observations, given all the observations up to the present time, which we will denote by $y_{1:t} = (y_1, \dots, y_t)$. (In this thesis, we only consider discrete-time systems, hence t is always an integer.) Since we will generally be unsure about the future, we would like to compute a best guess. In addition, we might want to know how confident we are of this guess, so we can hedge our bets appropriately. Hence we will try to compute a probability distribution over the possible future observations; we denote this by $P(y_{t+h}|y_{1:t})$, where $h > 0$ is the horizon, i.e., how far into the future we want to predict.

Sometimes we have some control over the system we are monitoring. In this case, we would like to predict future outcomes as a function of our inputs. Let $u_{1:t}$ denote our past inputs, and $u_{t+1:t+h}$ denote our next h inputs. Now the task is to compute $P(y_{t+h}|u_{1:t+h}, y_{1:t})$.

"Classical" approaches to time-series prediction use linear models, such as ARIMA, ARMAX, etc. (see e.g., [Ham94]), or non-linear models, such as neural networks (either feedforward or recurrent) or decision trees [MCH02]. For discrete data, it is common to use n-gram models (see e.g., [Jel97]) or variable-length Markov models [RST96, McC95].

There are several problems with the classical approach. First, we must base our prediction of the future on only a finite window into the past, say $y_{t-\ell:t}$, where $\ell \geq 0$ is the lag, if we are to do constant work per time step. If we know that the system we are modelling is Markov with an order $\leq \ell$, we will suffer no loss of performance, but in general the order may be large and unknown. Recurrent neural nets try to overcome this problem by using internal state, but they are still not able to model long-distance dependencies [BF95]. Second, it is difficult to incorporate prior knowledge into the classical approach: much of our knowledge cannot be expressed in terms of directly observable quantities, and black-box models, such as neural networks, are notoriously hard to interpret. Third, the classical approach has difficulties when we have multi-dimensional (multi-variate) inputs and/or outputs. For instance, consider the problem of predicting (and hence compressing) the next frame in a video stream using a neural network. Actual video compression schemes (such as MPEG) try to infer the underlying "cause" behind what they see, and use that to predict the next frame. This is the basic idea behind state-space models, which we discuss next.

In a state-space model, we assume that there is some underlying hidden state of the world that generates the observations, and that this hidden state evolves in time, possibly as a function of our inputs.[1] In an online

[1] The term "state-space model" is often used to imply that the hidden state is a vector in $\mathbb{R}^K$, for some K; I use the term more generally to mean a dynamical model which uses any kind of hidden state, whether it is continuous, discrete or both. e.g., I consider HMMs an example of a state-space model. In contrast to most work on time series analysis, this thesis focuses on models with discrete and mixed discrete-continuous states. One reason for this is that DBNs have their biggest payoff in the discrete setting: combining multiple continuous variables together results in a polynomial increase in complexity (see Section 2.4.2), but combining multiple discrete

setting, the goal is to infer the hidden state given the observations up to the current time. If we let X_t represent the hidden state at time t, then we can define our goal more precisely as computing $P(X_t|y_{1:t}, u_{1:t})$; this is called the belief state.

Astrom [Ast65] proved that the belief state is a sufficient statistic for prediction/control purposes, i.e., we do not need to keep around any of the past observations.[2] We can update the belief state recursively using Bayes rule, as we explain below. As in the case of prediction, we maintain a probability distribution over X_t, instead of just a best guess, in order to properly reflect our uncertainty about the "true" state of the world. This can be useful for information gathering; for instance, if we know we are lost, we may choose to ask for directions.

State-space models are better than classical time-series modelling approaches in many respects [Aok87, Har89, WH97, DK00, DK01]. In particular, they overcome all of the problems mentioned above: they do not suffer from finite-window effects, they can easily handle discrete and multi-variate inputs and outputs, and they can easily incorporate prior knowledge. For instance, often we know that there are variables that we cannot measure, but whose state we would like to estimate; such variables are called hidden or latent. Including these variables allows us to create models which may be much closer to the "true" causal structure of the domain we are modelling [Pea00].

Even if we are only interested in observable variables, introducing "fictitious" hidden variables often results in a much simpler model. For example, the apparent complexity of an observed signal may be more simply explained by imagining it is a result of two simple processes, the "true" underlying state, which may evolve deterministically, and our measurement of the state, which is often noisy. We can then "explain away" unexpected outliers in the observations in terms of a faulty sensor, as opposed to strange fluctuations in "reality". The underlying state may be of much lower dimensionality than the observed signal, as in the video compression example mentioned above.

In the following subsections, we discuss, in general terms, how to represent state-space models, how to use them to update the belief state and perform other related inference problems, and how to learn such models from data. We then discuss the two most common kinds of state-space models, namely Hidden Markov Models (HMMs) and Kalman Filter Models (KFMs). In subsequent chapters of this thesis, we will discuss representation, inference and learning of more general state-space models, called Dynamic Bayesian Networks (DBNs). A summary of the notation and commonly used abbreviations can be found in Appendix D.

1.1.1 Representation

Any state-space model must define a prior, $P(X_1)$, a state-transition function, $P(X_t|X_{t-1})$, and an observation function, $P(Y_t|X_t)$. In the controlled case, these become $P(X_t|X_{t-1}, U_t)$ and $P(Y_t|X_t, U_t)$; we allow the observation to depend on the control so that we can model active perception. For most of this thesis, we will omit U_t from consideration, for notational simplicity.

We assume that the model is first-order Markov, i.e., $P(X_t|X_{1:t-1}) = P(X_t|X_{t-1})$; if not, we can always make it so by augmenting the state-space. For example, if the system is second-order Markov, we just define a new state-space, $\tilde{X}_t = (X_t, X_{t-1})$, and set

$$P(\tilde{X}_t = (x_t, x_{t-1})|\tilde{X}_{t-1} = (x'_{t-1}, x_{t-2})) = \delta(x_{t-1}, x'_{t-1})P(x_t|x_{t-1}, x_{t-2}).$$

Similarly, we can assume that the observations are conditionally first-order Markov: $P(Y_t|Y_{1:t-1}, X_t) = P(Y_t|X_t, Y_{t-1})$. This is usually further simplified by assuming $P(Y_t|Y_{t-1}, X_t) = P(Y_t|X_t)$. These conditional independence relationships will be explained more clearly in Chapter 2.

We assume that the transition and observation functions are the same for all time; the model is said to be time-invariant or homogeneous. (Without this assumption, we could not model infinitely long sequences.) If the parameters do change over time, we can just add them to the state space, and treat them as additional random variables, as we will see in Section 6.1.1.

variables results in an exponential increase in complexity, since the new "mega" state-space is the cross product of the individual variables' state-spaces; DBNs help ameliorate this combinatorial explosion, as we shall see in Chapter 2.

[2] This assumes that the hidden state space is sufficiently rich. We discuss some ways to learn the hidden state space in Chapter 6. However, learning hidden state representations is difficult, which has motivated alternative forms of sufficient statistics [LSS01].

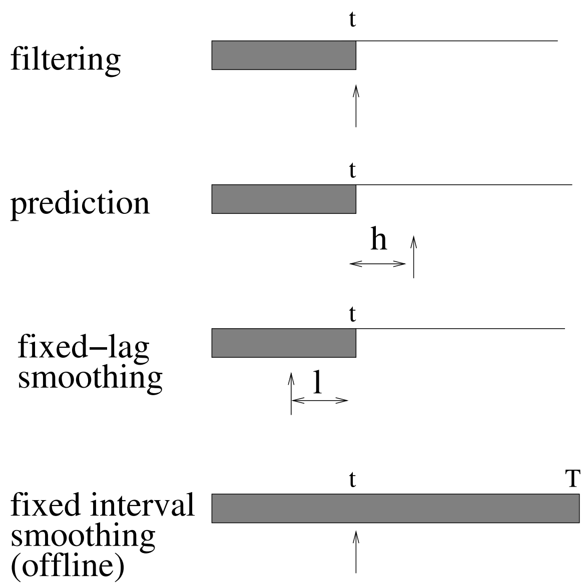


Figure 1.1: The main kinds of inference for state-space models. The shaded region is the interval for which we have data. The arrow represents the time step at which we want to perform inference. t is the current time, and T is the sequence length. See text for details.

There are many ways of representing state-space models, the most common being Hidden Markov Models (HMMs) and Kalman Filter Models (KFMs). HMMs assume X_t is a discrete random variable[3], $X_t \in \{1, \ldots, K\}$, but otherwise make essentially no restrictions on the transition or observation function; we will explain HMMs in more detail in Section 1.2. Kalman Filter Models (KFMs) assume X_t is a vector of continuous random variables, $X_t \in R^K$, and that $X_{1:T}$ and $Y_{1:T}$ are jointly Gaussian. We will explain KFMs in more detail in Section 1.3. Dynamic Bayesian Networks (DBNs) [DK89, DW91] provide a much more expressive language for representing state-space models; we will explain DBNs in Chapter 2.

A state-space model is a model of how X_t generates or "causes" Y_t and X_{t+1}. The goal of inference is to invert this mapping, i.e., to infer $X_{1:t}$ given $Y_{1:t}$. We discuss how to do this below.

1.1.2 Inference

We now discuss the main kinds of inference that we might want to perform using state-space models; see Figure 1.1 for a summary. The details of how to perform these computations depend on which model and which algorithm we use, and will be discussed later.

Filtering

The most common inference problem in online analysis is to recursively estimate the belief state using Bayes' rule:

$$\begin{aligned} P(X_t|y_{1:t}) &\propto P(y_t|X_t, y_{1:t-1})P(X_t|y_{1:t-1}) \\ &= P(y_t|X_t)\left[\sum_{x_{t-1}} P(X_t|x_{t-1})P(x_{t-1}|y_{1:t-1})\right] \end{aligned}$$

where the constant of proportionality is $1/c_t = 1/P(y_t|y_{1:t-1})$. We are licensed to replace $P(y_t|X_t, y_{1:t-1})$ by $P(y_t|X_t)$ because of the Markov assumption on Y_t. Similarly, the one-step-ahead pre-

[3] In this thesis, all discrete random variables will be considered unordered (cardinal), as opposed to ordered (ordinal), unless otherwise stated. (For example, $X_t \in \{\text{male}, \text{female}\}$ is cardinal, but $X_t \in \{\text{low}, \text{medium}, \text{high}\}$ is ordinal.) Ordinal values are sometimes useful for qualitative probabilistic networks [Wel90].

diction, $P(X_t|y_{1:t-1})$, can be computed from the prior belief state, $P(X_{t-1}|y_{1:t-1})$, because of the Markov assumption on X_t.

We see that recursive estimation consists of two main steps: predict and update; predict means computing $P(X_t|y_{1:t-1})$, sometimes written as $\hat{X}_{t|t-1}$, and update means computing $P(X_t|y_{1:t})$, sometimes written as $\hat{X}_{t|t}$. Once we have computed the prediction, we can throw away the old belief state; this operation is sometimes called "rollup". Hence the overall procedure takes constant space and time (i.e., independent of t) per time step.

This task is traditionally called "filtering", because we are filtering out the noise from the observations (see Section 1.3.1 for an example). However, in some circumstances the term "monitoring" might be more appropriate. For example, X_t might represent the state of a factory (e.g., which pipes are malfunctioning), and we wish to monitor the factory state over time.

Smoothing

Sometimes we want to estimate the state of the past, given all the evidence up to the current time, i.e., compute $P(X_{t-\ell}|y_{1:t})$, where $\ell > 0$ is the lag, e.g., we might want to figure out whether a pipe broke L minutes ago given the current sensor readings. This is traditionally called "fixed-lag smoothing", although the term "hindsight" might be more appropriate. In the offline case, this is called (fixed-interval) smoothing; this corresponds to computing $P(X_t|y_{1:T})$ for all $1 \leq t \leq T$.

Smoothing is important for learning, as we discuss in Section 1.1.3.

Prediction

In addition to estimating the current or past state, we might want to predict the future, i.e., compute $P(X_{t+h}|y_{1:t})$, where $h > 0$ is how far we want to look-ahead. Once we have predicted the future hidden state, we can easily convert this into a prediction about the future observations by marginalizing out X_{t+h}:

$$P(Y_{t+h} = y|y_{1:t}) = \sum_x P(Y_{t+h} = y|X_{t+h} = x)P(X_{t+h} = x|y_{1:t})$$

If the model contains input variables U_t, we must specify $u_{t+1:t+h}$ in order to predict the effects of our actions h steps into the future, since this is a conditional likelihood model.

Control

In control theory, the goal is to learn a mapping from observations or belief states to actions (a policy) so as to maximize expected utility (or minimize expected cost). In the special case where our utility function rewards us for achieving a certain value for the output of the system (reaching a certain observed state), it is sometimes possible to pose the control problem as an inference problem [Zha98a, DDN01, BMI99]. Specifically, we set Y_{t+h} to the desired output value (and leave $Y_{t+1:t+h-1}$ hidden), and then infer the values (if any) for $U_{t+1}, \ldots, U_{t+h}$ which will achieve this, where h is our guess about how long it will take to achieve the goal. (We can use dynamic programming to efficiently search over h.) The cost function gets converted into a prior on the control/input variable. For example, if the prior over U_t is Gaussian, $\mathcal{N}(U_t; 0, \Sigma)$, $U_t \sim \mathcal{N}(0, \Sigma)$, then the mode of the posterior $P(U_{t+1:t+h}|y_{1:t}, y_{t+h}, u_{1:t})$ will correspond to a sequence of minimal controls (minimal in the sense of having the smallest possible length, as measured by a Mahalanobis distance using Σ) which achieves the desired output sequence.

If U_t is discrete, inference amounts to enumerating all possible assignments to $U_{t+1:t+h}$, as in a decision tree; this is called receeding horizon control. We can (approximately) solve the infinite horizon control using similar methods so long as we discount future rewards at a suitably high rate [KMN99].

The general solution to control problems requires the use of influence diagrams (see e.g., [CDLS99, ch8], [LN01]). We will not discuss this topic further in this thesis.

Viterbi decoding

In Viterbi decoding (also called "abduction" or computing the "most probable explanation"), the goal is to compute the most likely sequence of hidden states given the data:

$$x^*_{1:t} = \arg\max_{x_{1:t}} P(x_{1:t}|y_{1:t})$$

(In the following subsection, we assume that the state space is discrete.)

By Bellman's principle of optimality, the most likely to path to reach state x_t consists of the most likely path to *some* state at time $t-1$, followed by a transition to x_t. Hence we can compute the overall most likely path as follows. In the forwards pass, we compute

$$\delta_t(j) = P(y_t|X_t = j) \max_i P(X_t = j|X_{t-1} = i)\delta_{t-1}(i)$$

where

$$\delta_t(j) \stackrel{\text{def}}{=} \max_{x_{1:t-1}} P(X_{1:t} = x_{1:t-1}, X_t = j|y_{1:t}).$$

This is the same as the forwards pass of filtering, except we replace sum with max (see Section B.2). In addition, we keep track of the identity of the most likely predecessor to each state:

$$\psi_t(j) = \arg\max_i P(X_t = j|X_{t-1} = i)\delta_{t-1}(i)$$

In the backwards pass, we can compute the identity of the most likely path recursively as follows:

$$x^*_t = \psi_{t+1}(x^*_{t+1})$$

Note that this is different than finding the most likely (marginal) state at time t.

One application of Viterbi is in speech recognition. Here, X_t typically represents a phoneme or syllable, and Y_t typically represents a feature vector derived from the acoustic signal [Jel97]. $x^*_{1:t}$ is the most likely hypothesis about what was just said. We can compute the N best hypotheses in a similar manner [Nil98, NG01].

Another application of Viterbi is in biosequence analysis, where we are interested in offline analysis of a fixed-length sequence, $y_{1:T}$. Y_t usually represents the DNA base-pair or amino acid at location t in the string. X_t often represents whether Y_t was generated by substitution, insertion or deletion compared to some putative canonical family sequence. As in speech recognition, we might be interested in finding the most likely "parse" or interpretation of the data, so that we can align the observed sequence to the family model.

Classification

The likelihood of a model, M, is $P(y_{1:t}|M)$, and can be computed by multiplying together all the normalizing constants that arose in filtering:

$$P(y_{1:t}) = P(y_1)P(y_2|y_1)P(y_3|y_{1:2}) \ldots P(y_T|y_{1:T-1}) = \prod_{t=1}^{T} c_t \tag{1.1}$$

which follows from the chain rule of probability. This can be used to classify a sequence as follows:

$$C^*(y_{1:T}) = \arg\max_C P(y_{1:T}|C)P(C)$$

where $P(y_{1:T}|C)$ is the likelihood according to the model for class C, and $P(C)$ is the prior for class C. This method has the advantage of being able to handle sequences of variable-length. By contrast, most classifiers work with fixed-sized feature vectors.[4]

[4] One could pretend that successive observations in the sequence are iid, and then apply a naive Bayes classifier: $P(y_{1:T}|C) = \prod_{t=1}^{T} P(y_t|C)$. However, often it matters in what order the observations arrive, e.g., in classifying a string of letters as a word. There has been some work on applying support vector machines to variable length sequences [JH99], but this uses an HMM as a subroutine.

Summary

We summarize the various inference problems in Figure 1.1. In Section 3.7, we will show how all of the above algorithms can be formulated in terms of a set of abstract operators which we will call forwards and backwards operators. There are many possible implementations of these operators which make different tradeoffs between accuracy, speed, generality, etc. In Sections 1.2 and 1.3, we will see how to implement these operators for HMMs and KFMs. In later chapters, we will see different implementations of these operators for general DBNs.

1.1.3 Learning

A state-space model usually has some free parameters θ which are used to define the transition model, $P(X_t|X_{t-1})$, and the observation model, $P(Y_t|X_t)$. Learning means estimating these parameters from data; this is often called system identification.

The usual criterion is maximum-likelihood (ML), which is suitable if we are doing off-line learning with large data sets. Suppose, as is typical in speech recognition and bio-sequence analysis, that we have N_{train} iid sequences, $Y = (y^1_{1:T}, \dots y^{N_{\text{train}}}_{1:T})$, where we have assumed each sequence has the same length T for notational simplicity. Then the goal of learning is to compute

$$\theta^*_{ML} = \arg\max_\theta P(Y|\theta) = \arg\max_\theta \log P(Y|\theta)$$

where the log-likelihood of the training set is

$$\log P(Y|\theta) \quad = \quad \log \prod_{m=1}^{N_{\text{train}}} P(y^m_{1:T}|\theta) = \sum_{m=1}^{N_{\text{train}}} \log P(y^m_{1:T}|\theta)$$

A minor variation is to include a prior on the parameters and compute the MAP (maximum a posteriori) solution

$$\theta^*_{MAP} = \arg\max_\theta \log P(Y|\theta) + \log P(\theta)$$

This can be useful when the number of free parameters is much larger than the size of the dataset (the prior acting like a regularizer to prevent overfitting), and for online learning (where at timestep t the dataset only has size t).

What makes learning state-space models difficult is that some of the variables are hidden. This means that the likelihood surface is multi-modal, making it difficult to find the globally optimal parameter value.[5] Hence most learning methods just attempt to find a locally optimal solution.

The two standard techniques for ML/MAP parameter learning are gradient ascent[6], and EM (expectation maximization), both of which are explained in Appendix C. Note that both methods use inference as a subroutine, and hence efficient inference is a prerequisite for efficient learning. In particular, for offline learning, we need need to perform fixed-interval smoothing (i.e., computing $P(X_t|y_{1:T}, \theta)$ for all t): learning with filtering may fail to converge correctly. To see why, consider learning to solve murders: hindsight is always required to infer what happened at the murder scene.[7]

For online learning, we can use fixed-lag smoothing combined with online gradient ascent or online EM. Alternatively, we can adopt a Bayesian approach and treat the parameters as random variables, and just add them to the state-space. Then learning just amounts to filtering (i.e., sequential Bayesian updating) in the augmented model, $P(X_t, \theta_t|y_{1:t})$. Unfortunately, inference in such models is often very difficult, as we will see.

A much more ambitious task than parameter learning is to learn the structure (parametric form) of the model. We will discuss this in Chapter 6.

---

[5] One trivial source of multi-modality has to do with symmetries in the hidden state-space. Often we can permute the labels of the hidden states without affecting the likelihood.

[6] Ascent, rather than descent, since we are trying to maximize likelihood.

[7] This example is from [RN02].

1.2 Hidden Markov Models (HMMs)

We now give a brief introduction to HMMs.[8] The main purpose of this section is to introduce notation and concepts in a simple and (hopefully) familiar context; these will be generalized to the DBN case later.

1.2.1 Representation

An HMM is a stochastic finite automaton, where each state generates (emits) an observation. We will use X_t to denote the hidden state and Y_t to denote the observation. If there are K possible states, then $X_t \in \{1, \ldots, K\}$. Y_t might be a discrete symbol, $Y_t \in \{1, \ldots, L\}$, or a feature-vector, $Y_t \in \mathbb{R}^L$.

The parameters of the model are the initial state distribution, $\pi(i) = P(X_1 = i)$, the transition model, $A(i,j) = P(X_t = j|X_{t-1} = i)$, and the observation model $P(Y_t|X_t)$.

$\pi(\cdot)$ represents a multinomial distribution. The transition model is usually characterized by a conditional mulitnomial distribution: $A(i,j) = P(X_t = j|X_{t-1} = i)$, where A is a stochastic matrix (each row sums to one). The transition matrix A if often sparse; the structure of the matrix is often depicted graphically, as in Figure 1.2 which depicts a left-to-right transition matrix. (This means low numbered states can only make transitions to higher numbered states or to themselves.) Such graphs should not be confused with the graphical models we will introduce in Chapter 2.

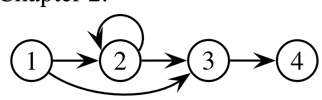


Figure 1.2: A left-to-right state transition diagram for a 4-state HMM. Nodes represent states, and arrows represent allowable transitions, i.e., transitions with non-zero probability. The self-loop on state 2 means $P(X_t = 2|X_{t-1} = 2) = A(2,2) > 0$.

If the observations are discrete symbols, we can represent the observation model as a matrix: $B(i,k) = P(Y_t = k|X_t = i)$. If the observations are vectors in $\mathbb{R}^L$, it is common to represent $P(Y_t|X_t)$ as a Gaussian:

$$P(Y_t = y|X_t = i) = \mathcal{N}(y; \mu_i, \Sigma_i)$$

where $\mathcal{N}(y; \mu, \Sigma)$ is the Gaussian density with mean μ and covariance Σ evaluated at y:

$$\mathcal{N}(y; \mu, \Sigma) = \frac{1}{(2\pi)^{L/2}|\Sigma|^{\frac{1}{2}}} \exp\left(-\tfrac{1}{2}(y-\mu)'\Sigma^{-1}(y-\mu)\right)$$

A more flexible representation is a mixture of M Gaussians:

$$P(Y_t = y|X_t = i) = \sum_{m=1}^{M} P(M_t = m|X_t = i)\mathcal{N}(y; \mu_{m,i}, \Sigma_{m,i})$$

where M_t is a hidden variable that specifies which mixture component to use, and $P(M_t = m|X_t = i) = C(i,m)$ is the conditional prior weight of each mixture component. For example, one mixture component might be a Gaussian centered at the expected output for state i and with a narrow variance, and the second component might be a Gaussian with zero mean and a very broad variance; the latter approximates a uniform distribution, and can account for outliers, making the model more robust.

In speech recognition, it is usual to assume that the parameters are stationary or time-invariant, i.e., that the transition and observation models are shared (tied) across time slices. This allows the model to be applied to sequences of arbitrary length. In biosequence analysis, it is common to use position-dependent observation models, $B_t(i,k) = P(Y_t = k|X_t = i)$, since certain positions have special meanings. These models can only handle fixed-length sequences. However, by adding a background state with a position-invariant distribution, the models can be extended to handle varying length sequences. In this thesis, we will usually assume time/position-invariant parameters, but this is mostly for notational simplicity.

[8] See [Rab89] for an excellent tutorial, and [Ben99] for a review of more recent developments; [MZ97] provides a thorough mathematical treatise on HMMs. The book [DEKM98] provides an excellent introduction to the application of HMMs to biosequence analysis, and the book [Jel97] describes how HMMs are used in speech recognition.

1.2.2 Inference

Offline smoothing can be performed in an HMM using the well-known forwards-backwards algorithm, which will be explained in Section 3.2. In the forwards pass, we recursively compute the filtered estimate $\alpha_t(i) = P(X_t = i|y_{1:t})$, and in the backwards pass, we recursively compute the smoothed estimate $\gamma_t(i) = P(X_t = i|y_{1:T})$ and the smoothed two-slice estimate $\xi_{t-1,t|T}(i,j) = P(X_{t-1} = i, X_t = j|y_{1:T})$ which is needed for learning.[9]

If X can be in K possible states, filtering takes $O(K^2)$ operations per time step, since we must do a matrix-vector multiply at every step. Smoothing therefore takes $O(K^2 T)$ time in the general case. If A is sparse, and each state has at most F_{in} predecessors, then the complexity is $(KF_{in}T)$.

1.2.3 Learning

In this section we give an informal explanation of how to do offline maximum likelihood (ML) parameter estimation for HMMs using the EM (Baum-Welch) algorithm. This will form the basis of the generalizations in Chapter 6.

If we could observe $X_{1:T}$, learning would be easy. For instance, the ML estimate of the transition matrix could be computed by normalizing the matrix of co-occurrences (counts):

$$\hat{A}_{ML}(i,j) = \frac{N(i,j)}{\sum_k N(i,k)}$$

where

$$N(i,j) = \sum_{t=2}^{T} I(X_{t-1} = i, X_t = j)$$

and $I(E)$ is a binary indicator that is 1 if event E occurs and is 0 otherwise. Hence $N(i,j)$ is the number of $i \to j$ transitions in a given sequence. (We have assumed there is a single training sequence for notational simplicity. If we have more than one sequence, we simply sum the counts across sequences.) We can estimate $P(Y_t|X_t)$ and $P(X_1)$ similarly.

The problem, however, is that $X_{1:T}$ is hidden. The basic idea of the EM algorithm, roughly speaking, is to estimate $P(X_{1:T}|y_{1:T})$ using inference, and to use the expected (pairwise) counts instead of the real counts to estimate the parameters θ. Since the expectation depends on the value of θ, and the value of θ depends on the expectation, we need to iterate this scheme.

We start with an initial guess of θ, and then perform the E (expectation) step. Specifically, at iteration k we compute

$$E[N(i,j)|\theta^k] = E\sum_{t=2}^{T} I(X_{t-1} = i, X_t = j|y_{1:T}) = \sum_{t=2}^{T} P(X_{t-1} = i, X_t = j|y_{1:T}) = \sum_{t=2}^{T} \xi_{t-1,t|T}(i,j)$$

ξ can be computed using the forwards-backwards algorithm, as discussed in Section 3.2. $E[N(i,j)|\theta^k]$ is called the expected sufficient statistic (ESS) for A, the transition matrix. We compute similar ESSs for the other parameters.

We then perform an M (maximization) step. This tries to maximize the value of the expected complete-data log-likelihood:

$$\theta^{k+1} = \arg\max_{\theta} Q(\theta|\theta^k)$$

where Q is the auxiliary function

$$Q(\theta|\theta^k) = E_{X_{1:T}}\left[P(y_{1:T}, X_{1:T}|\theta)|\theta^k\right]$$

[9] It is more common to define α as an unconditional joint probability, $\alpha_t(i) = P(X_t = i, y_{1:t})$. Also, it is more common to define the backwards pass as computing $\beta_t(i) = P(y_{t+1:T}|X_t = i)$; $\gamma_t(i)$ and $\xi_{t-1,t|T}$ can then be derived from $\alpha_t(i)$ and $\beta_t(i)$. These details will be explained in Section 3.2. The chosen notation is designed to bring out the similarity with inference in KFMs and general DBNs.

For the case of multinomials, it is easy to show that this amounts to normalizing the expected counts:

$$\hat{A}_{ML}^{k+1}(i,j) \propto E[N(i,j)|\theta^k]$$

[BPSW70, DLR77] proved that the EM algorithm is guaranteed to increase the likelihood at each step until a critical point (usually a local maximum) is reached. In practice, we declare convergence when the relative change in the log-likelihood is less than some threshold.[10] See Section C.4.2 for more details of the EM algorithm.

1.2.4 The problem with HMMs

Suppose we want to track the state (e.g., the position) of N objects in an image sequence. Let each object be in one of k possible states. Then $X_t = (X_t^1, \ldots, X_t^N)$ can have $K = k^N$ possible values, since we must form the Cartesian product of the state-spaces of each individual object. This means that we require an exponential number of parameters (exponential in the number of objects) to specify the transition and observation models, which means we will need a lot of data to learn the model (high sample complexity). In addition, inference takes exponential time, e.g., forwards-backwards takes $O\left(Tk^{2N}\right)$ (high computational complexity). DBNs will help ameliorate both of these problems.

1.3 Kalman Filter Models (KFMs)

We now give a brief introduction to KFMs, also known as linear dynamical systems (LDSs), state-space models, etc.[11] The main purpose of this section is to introduce notation and concepts in a simple and (hopefully) familiar context; these will form the basis of future generalizations.

1.3.1 Representation

A KFM assumes $X_t \in \mathbb{R}^{N_x}$, $Y_t \in \mathbb{R}^{N_y}$, $U_t \in \mathbb{R}^{N_u}$, and that the transition and observation functions are linear-Gaussian, i.e.,

$$P(X_t = x_t|X_{t-1} = x_{t-1}, U_t = u) = \mathcal{N}(x_t; Ax_{t-1} + Bu + \mu_X, Q)$$

and

$$P(Y_t = y|X_t = x, U_t = u) = \mathcal{N}(y; Cx + Du + \mu_Y, R)$$

In other words, $X_t = AX_{t-1} + BU_t + V_t$, where $V_t \sim \mathcal{N}(\mu_X, Q)$ is a Gaussian noise term. Similarly, $Y_t = CX_t + DU_t + W_t$, where $W_t \sim \mathcal{N}(\mu_Y, R)$ is another Gaussian noise term assumed independent of V_t. The noise terms are assumed to be temporally white, which means $V_t \perp V_{t'}$ (i.e., V_t is marginally independent of $V_{t'}$) for all $t \neq t'$, and similarly for W_t,

A is a $N_x \times N_x$ matrix, B is a $N_x \times N_u$ matrix, C is a $N_y \times N_x$ matrix, D is a $N_y \times N_u$ matrix, Q is a $N_x \times N_x$ positive semi-definite (psd) matrix called the process noise, and R is a $N_y \times N_y$ psd matrix called the observation noise. As for HMMs, we assume the parameters are time-invariant.

Without loss of generality, we can assume μ_X and μ_Y are 0, since we can always augment X_t with the constant 1, and add μ_X (mu_Y) to the first column of A (C) respectively. Similarly, we can assume Q or R is diagonal; see [RG99] for details.

Example

Suppose we are tracking an object as it moves through $\mathbb{R}^2$. Let $X_t = (x_t, y_t, \dot{x}_t, \dot{y}_t)$ represent the position and velocity of the object. Consider a constant-velocity model; however, we assume the object will get

[10] The fact that the likelihood stops changing does not mean that the parameters stop changing: it is possible for EM to cause the estimate to oscillate in parameter space, without changing the likelihood.

[11] See [RG99, Min99] for good tutorials on KFMs from the DBN perspective. There are many textbooks that give a more classical treatment of KFMs, see e.g., [AM79, BSF88].

"buffetted" around by some unknown source (e.g., the wind), which we will model as Gaussian noise. Hence

$$\begin{pmatrix} x_t \\ y_t \\ \dot{x}_t \\ \dot{y}_t \end{pmatrix} = \begin{pmatrix} 1 & 0 & \Delta & 0 \\ 0 & 1 & 0 & \Delta \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_{t-1} \\ y_{t-1} \\ \dot{x}_{t-1} \\ \dot{y}_{t-1} \end{pmatrix} + V_t$$

where Δ is the sampling period, $V_t \sim \mathcal{N}(0, Q)$ is the noise, and Q is the following covariance matrix

$$Q = \begin{pmatrix} Q_x & Q_{x,y} & 0 & 0 \\ Q'_{x,y} & Q_y & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Q_x is the variance of the noise in the x direction, Q_y is the variance of the noise in the y direction, and $Q_{x,y}$ is the cross-covariance. (If the bottom right matrix were non-zero, this would be called a "random acceleration model", since it would add noise to the velocities.)

Assume also that we only observe the position of the object, but not its velocity. Hence

$$\begin{pmatrix} x_t^o \\ y_t^o \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_t \\ y_t \\ \dot{x}_t \\ \dot{y}_t \end{pmatrix} + W_t$$

where $W_t \sim \mathcal{N}(0, R)$.

Intuitively we will be able to infer the velocity by taking successive differences between the observed positions; however, we need to filter out the noise first. This is exactly what the Kalman filter will do for us, as we will see below.

1.3.2 Inference

The equations for Kalman filtering/smoothing can be derived in an analogous manner to the equations for HMMs: see Section 3.6.1.

Example

We illustrate the Kalman filter and smoother by applying them to the tracking problem in Section 1.3.1. Suppose we start out at position $(10, 10)$ moving to the right with velocity $(1, 0)$. We sampled a random trajectory of length 15, and show the filtered and smoothed trajectories in Figure 1.3.

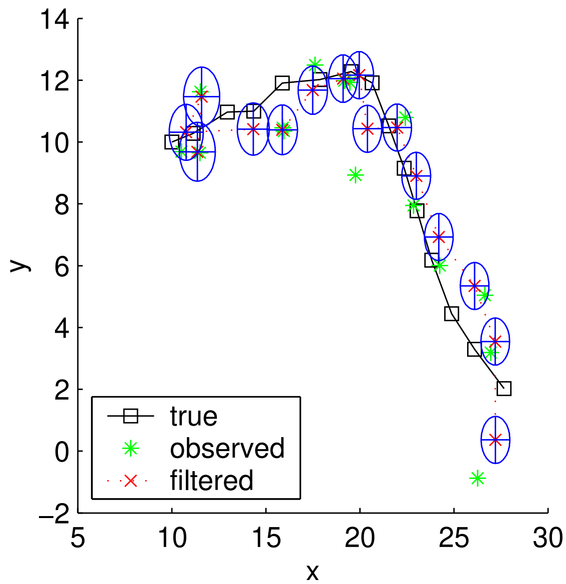
The mean squared error of the filtered estimate is 4.9; for the smoothed estimate it is 3.2. Not only is the smoothed estimate better, but we know that it is better, as illustrated by the smaller uncertainty ellipses; this can help in e.g., data association problems (see Section 2.4.6).

1.3.3 Learning

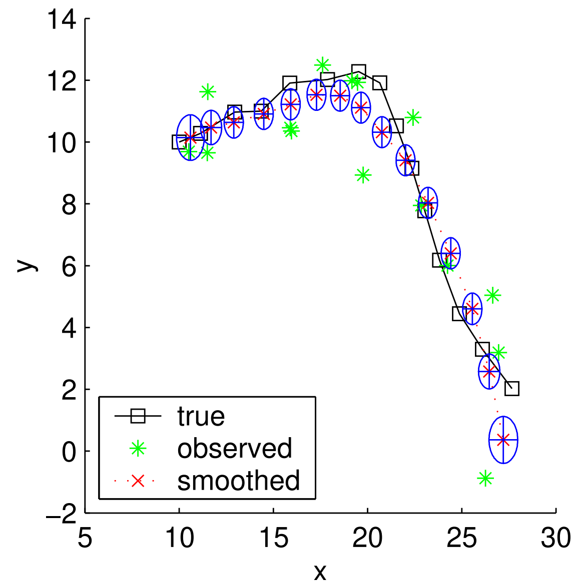
It is possible to compute ML (or MAP) estimates of the parameters of a KFM using gradient methods [Lju87] or EM [GH96b, Mur98]. We do not give the equations here because they are quite hairy, and in any case are just a special case of the equations we present in Section C.2.2. Suffice it to say that, conceptually, the methods are identical to the learning methods for HMMs.

1.3.4 The problem with KFMs

KFMs assume the system is jointly Gaussian. This means the belief state must be unimodal, which is inappropriate for many problems, especially those involving qualitative (discrete) variables. For example, some systems have multiple modes or regimes of behavior; an example is given in Figure 1.4: either the bird moves to the left, to the right, or it moves straight (which can be modelled as a equal mixture of the left and right



(a)



(b)

Figure 1.3: Results of inference for the the tracking problem in Section 1.3.1. (a) Filtering. (b) Smoothing. Boxes represent the true position, stars represent the observed position, crosses represented the estimated (mean) position, ellipses represent the uncertainty (covariance) in the position estimate. Notice that the smoothed covariances are smaller than the filtered covariances, except at $t = T$, as expected.

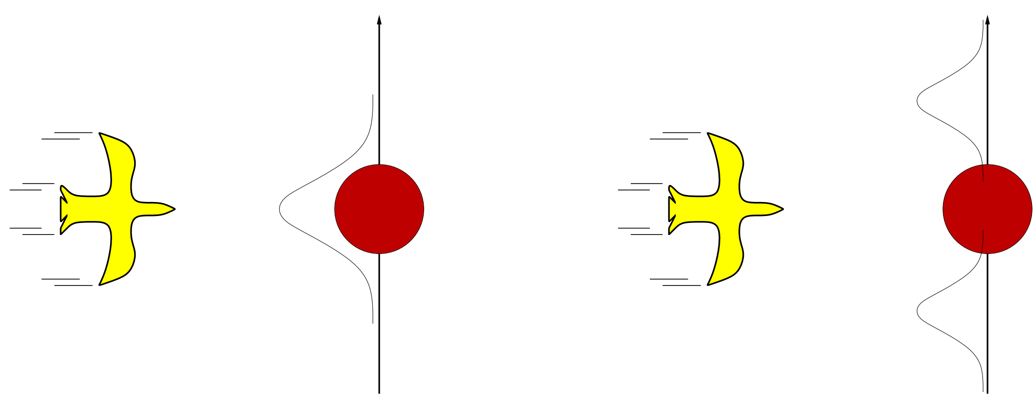


Figure 1.4: If a bird/plane is heading towards an obstacle, it is more likely to swerve to one side or another, hence the prediction should be multi-modal, which a KFM cannot do. This figure is from [RN02].

models), i.e., the dynamics is piece-wise linear. This model is called a switching KFM; we will discuss this and related models in Section 2.4.3. Unfortunately, the belief state at time t may have $O(K^t)$ modes; indeed, in general inference in this model is NP-hard, as we will see in Section 3.6.3. We will consider a variety of approximate inference schemes for this model.

Some systems have unimodal posteriors, but nonlinear dynamics. It is common to use the extended (see e.g., [BSF88]) or unscented Kalman filter (see e.g., [WdM01]) as an approximation in such cases.

1.4 Overview of the rest of the thesis

The rest of this thesis is concerned with representation, inference and learning in a class of models called dynamic Bayesian networks (DBNs), of which HMMs and KFMs are just special cases. By using DBNs, we are able to represent, and hence learn, much more complex models of sequential data, which hopefully are closer to "reality". The price to be paid is increased algorithmic and computational complexity.

In Chapter 2, we define what DBNs are, and give a series of examples to illustrate their modelling power. This should provide sufficient motivation to read the rest of the thesis. The main novel contribution of this chapter is a way to model hierarchical HMMs [FST98] as DBNs [MP01]. This change of representation means we can use the algorithms in Chapter 3, which take $O(T)$ time, whereas the original algorithm [FST98] takes $O(T^3)$ time. The reduction in complexity from cubic to linear allows HHMMs to be applied to long sequences of data (e.g., biosequences). We then discuss, at some length, the relationship between HHMMs and other models, including abstract HMMs, semi-Markov models and models used for speech recognition.

In Chapter 3, we discuss how to do exact inference in DBNs. The novel contributions are a new way of applying the junction tree algorithm to DBNs, and a way of trading time for space when doing (offline) smoothing [BMR97a]. In particular, we show how to reduce the space requirements from $O(T)$ to $O(\log T)$, where T is the length of the sequence, if we increase the running time by a $\log T$ factor. This algorithm enables us to learn models from very long sequences of data (e.g., biosequences).

In Chapter 4, we discuss how to speed up inference using a variety of deterministic approximation algorithms. The novel contributions are a new algorithm, called the factored frontier (FF) [MW01], and an analysis of the relationship between FF, loopy belief propagation (see Section B.7.1), and the Boyen-Koller [BK98b] algorithm. We also compare these algorithms empirically on the problem of modeling freeway traffic using coupled HMMs [KM00]. We then survey algorithms for approximate inference in switching KFMs.

In Chapter 5, we discuss how to use Sequential Monte Carlo (sampling) methods for approximate filtering. The novel contributions are an explanation of how to apply Rao-Blackwellised particle filtering (RBPF) to general DBNs [DdFMR00, MR01], and the application of RBPF to a problem in mobile robotics called SLAM (Simultaneous Localization and Mapping) [Mur00]. This enables one to learn maps with orders of magnitude more landmarks than is possible using conventional (extended Kalman filter based) techniques. For completeness, we also discuss how to apply RBPF and Rao-Blackwellised Gibbs sampling to switching KFMs.

In Chapter 6, we explain how to do learning, i.e., estimate the parameters and structure of a DBN from data. The main novel contributions are an extension of the structural EM algorithm [Fri97, Fri98] to the DBN case [FMR98], plus various applications of DBNs, including discovering motifs from synthetic DNA sequences, predicting people's movements based on tracking data, and modelling freeway traffic data.

The appendices contain background material on graphical models that is not specific to DBNs. Appendix A defines various kinds of probabilistic graphical models, and introduces some conditional probability distributions that will be used throughout the thesis. Appendix B contains some novel material on ways of handling evidence in the junction tree algorithm (Section B.4.8), and a variational approximation for inference in BNs that have discrete nodes with continuous parents (Section B.5.3). Appendix C contains some novel material on computing ML estimates for tied conditional linear Gaussian distributions (Section C.2.2), and an experimental comparison of the speed of EM vs gradient methods (Section C.4.3). Although the remaining material is not novel, we do not know of any books or articles that provide such a broad treatment of the field; as such, we believe the appendices have merit in their own right. Appendix D defines some of the more frequently used notation and abbreviations.

1.5 A note on software

Many of the algorithms and examples in this thesis have been implemented using my Bayes Net Toolbox for Matlab (BNT) [Mur01b]. This is open-source and is freely available from `www.cs.berkeley.edu/~murphyk/Bayes/bnt.html`.

1.6 Declaration of previous work

This thesis is based on the following previously published material:

- “Space-efficient inference in dynamic probabilistic networks”, J. Binder, K. Murphy and S. Russell. IJCAI 1997. Section 3.7.1.
- “Learning the structure of dynamic probabilistic networks”, N. Friedman, K. Murphy and S. Russell. UAI 1998. Section 6.2.1.
- “A Variational Approximation for Bayesian Networks with Discrete and Continuous Latent Variables”, K. Murphy. UAI 1999. Section B.5.3.
- “Loopy Belief Propagation for Approximate Inference: an Empirical Study”, K. Murphy, Y. Weiss and M. Jordan. UAI 1999. Section B.7.1.
- “A Dynamic Bayesian Network Approach to Figure Tracking Using Learned Dynamic Models”, V. Pavlovic, J. Rehg, T-J. Cham, and K. Murphy. ICCV 1999. Section 4.3.
- “Bayesian Map Learning in Dynamic Environments”, K. Murphy. NIPS 2000. Section 5.3.2.
- “Rao-Blackwellised Particle Filtering for Dynamic Bayesian Networks”, A. Doucet, N. de Freitas, K. Murphy and S. Russell. UAI 2000. Section 5.3.
- “Rao-Blackwellised Particle Filtering for Dynamic Bayesian Networks”, K. Murphy and S. Russell. In *Sequential Monte Carlo Methods in Practice*, Doucet et al (eds), 2001. Section 5.3.
- “The Factored Frontier Algorithm for Approximate Inference in DBNs”, K. Murphy and Y. Weiss. UAI 2001. Section 4.2.
- “Linear time inference in hierarchical HMMs”, K. Murphy and M. Paskin. NIPS 2001. Section 2.3.9.
- “The Bayes Net Toolbox for Matlab”, K. Murphy. Computing Science and Statistics: Proceedings of the Interface, 2001. Appendices.
- “A Coupled HMM for Audio-Visual Speech Recognition”, A. Nefian, L. Liang, X. Pi, L. Xiaoxiang, C. Mao and K. Murphy. ICASSP 2002. Section 2.3.8.

Chapter 2

DBNs: Representation

2.1 Introduction

In this chapter, I define what DBNs are, and then give a "laundry list" of examples of increasing complexity. This demonstrates the versatility (expressive power) of DBNs as a modelling language, and shows that DBNs are useful for a wide range of problems. This should serve as motivation to read the rest of the thesis.

The unifying perspective of DBNs brings out connections between models that had previously been considered quite different. This, plus the existence of general purpose DBN software, such as BNT [Mur01b] and GMTK [BZ02], will hopefully discourage people from writing new code (and new papers!) everytime they make what often amounts to just a small tweak to some existing model.

The novel contribution of this chapter is a way to represent hierarchical HMMs (HHMMs) [FST98] as DBNs [MP01]; this is discussed in Section 2.3.9. Once an HHMM is represented as a DBN, any of the inference and learning techniques discussed in this thesis can be applied. In particular, exact inference using the junction tree algorithm (see Chapter 3) enables smoothing to be performed in $O(T)$ time, whereas the original algorithm required $O(T^3)$ time. This is just one example of the benefits of thinking in terms of DBNs.

Since DBNs are an extension of Bayes nets (BNs), the reader is assumed to already be familiar with BNs; read Appendix A for a refresher if necessary.

2.2 DBNs defined

A dynamic Bayesian network (DBN) [DK89] is a way to extend Bayes nets to model probability distributions over semi-infinite collections of random variables, $Z_1, Z_2, \ldots$. Typically we will partition the variables into $Z_t = (U_t, X_t, Y_t)$ to represent the input, hidden and output variables of a state-space model. We only consider discrete-time stochastic processes, so we increase the index t by one every time a new observation arrives. (The observation could represent that something has changed (as in e.g., [NB94]), making this a model of a discrete-event system.) Note that the term "dynamic" means we are modelling a dynamic system, not that the network changes over time. (See Section 2.5 for a discussion of DBNs which change their structure over time.)

A DBN is defined to be a pair, $(B_1, B_{\rightarrow})$, where B_1 is a BN which defines the prior $P(Z_1)$, and $B_{\rightarrow}$ is a two-slice temporal Bayes net (2TBN) which defines $P(Z_t|Z_{t-1})$ by means of a DAG (directed acyclic graph) as follows:

$$P(Z_t|Z_{t-1}) = \prod_{i=1}^{N} P(Z_t^i|\text{Pa}(Z_t^i))$$

where Z_t^i is the i'th node at time t, which could be a component of X_t, Y_t or U_t, and $\text{Pa}(Z_t^i)$ are the parents of Z_t^i in the graph. The nodes in the first slice of a 2TBN do not have any parameters associated with them, but each node in the second slice of the 2TBN has an associated conditional probability distribution (CPD), which defines $P(Z_t^i|\text{Pa}(Z_t^i))$ for all $t > 1$. The form of these CPDs is arbitrary: see Tables A.1 and A.2 for some examples.

The parents of a node, $\text{Pa}(Z_t^i)$, can either be in the same time slice or in the previous time slice, i.e., we assume the model is first-order Markov. However, this is mostly for notational simplicity: there is no fundamental reason why we cannot allow arcs to skip across slices. The arcs between slices are from left to right, reflecting the causal flow of time. If there is an arc from Z_{t-1}^i to Z_t^i, this node is called persistent. The arcs within a slice are arbitrary, so long as the overall DBN is a DAG.[1] Intuitively, directed arcs within a slice represent "instantaneous" causation. It is also useful to allow undirected arcs within a slice, which model correlation or constraints rather than causation; the resulting model is then called a (dynamic) chain graph [Dah00]. However, we will not consider such models in this thesis.

We assume the parameters of the CPDs are time-invariant, i.e., the model is time-homogeneous. If parameters can change, we can add them to the state-space and treat them as random variables. Alternatively, if there is only a finite set of possible parameter values (e.g., corresponding to different regimes), we can add a hidden variable which selects which set of parameters to use.

The semantics of a DBN can be defined by "unrolling" the 2TBN until we have T time-slices. The resulting joint distribution is then given by

$$P(Z_{1:T}) = \prod_{t=1}^{T} \prod_{i=1}^{N} P(Z_t^i|\text{Pa}(Z_t^i))$$

The difference between a DBN and an HMM is that a DBN represents the hidden state in terms of a set of random variables, $X_t^1, \ldots, X_t^{N_h}$, i.e., it uses a distributed representation of state. By contrast, in an HMM, the state space consists of a single random variable X_t. The difference between a DBN and a KFM is that a KFM requires all the CPDs to be linear-Gaussian, whereas a DBN allows arbitrary CPDs. In addition, HMMs and KFMs have a restricted topology, whereas a DBN allows much more general graph structures. The examples below will make this clearer.

Before diving into a series of DBN examples, we remark that some other ways of representing time in the context of BNs have been proposed in the UAI community, e.g., [AC95, DG95, SY99] and the references therein. However, very few of these formalisms have genuinely more expressive power (as opposed to just having nicer "syntactic sugar") than DBNs, and those that do are generally intractable, from the point of view of inference, learning or both. In the engineering community, DBNs have become the representation of choice because they embody a good tradeoff between expressiveness and tractability, and include the vast majority of models that have proved succesful in practice, as we will see below.

2.3 Representing HMMs and their variants as DBNs

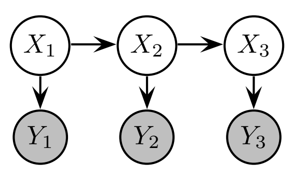


Figure 2.1: An HMM represented as an instance of a DBN, unrolled for 3 slices. Since the structure repeats, the model can be defined by showing just the first two slices.

We can represent an HMM as a DBN as shown in Figure 2.1. We follow standard convention and use shading to mean a node is observed; clear nodes are hidden. This graph represents the following conditional independence assumptions: $X_{t+1} \perp X_{t-1}|X_t$ (the Markov property) and $Y_t \perp Y_{t'}|X_t$, for $t' \neq t$. The latter assumption can be relaxed, as we discuss in Section 2.3.3.

[1] The intra-slice topology of the first slice may be different from the other slices, since the first slice may either represent the stationary distribution of the chain (if we assume the process started at $t = -\infty$), or the initial conditions of the chain (if we assume the process started at $t = 1$).

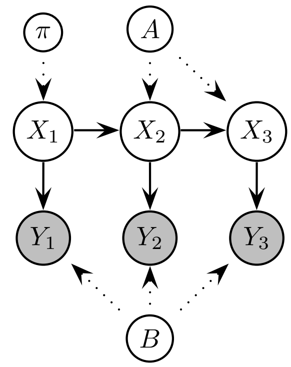


Figure 2.2: An HMM in which we explicitly represent the parameters (nodes with outgoing dotted arcs) and the fact that they are tied across time-slices. The parameters are $P(X_1 = i) = \pi(i)$, $P(X_t = j|X_{t-1} = i) = A(i, j)$, and $P(Y_t = j|X_t = i) = B(i, j)$. If the CPD for Y is a Gaussian, we would replace the B node with the mean and covariance parameters.

As is usual for Bayesian networks, we must define the conditional probability distribution (CPD) of each node given its parents. For the HMM in Figure 2.1, this means defining $P(X_1)$, $P(X_t|X_{t-1})$ and $P(Y_t|X_t)$. The CPD for $P(X_1)$ is usually represented as a vector, which represents a multinomial distribution, i.e., $P(X_1 = i) = \pi(i)$, where $0 \leq \pi(i) \leq 1$ and $\sum_i \pi(i) = 1$. The CPD for $P(X_t|X_{t-1})$ is usually represented as a stochastic matrix, i.e., $P(X_t = j|X_{t-1} = i) = A(i, j)$ where each row (which represents a conditional multinomial) sums to 1. The CPD for $P(Y_t|X_t)$ can take a variety of forms. If Y_t is discrete, we could use a conditional multinomial, represented as a stochastic matrix: $P(Y_t = j|X_t = i) = B(i, j)$. If Y_t is continuous, we could use a conditional Gaussian or a conditional mixture of Gaussians (see Section 2.3.1). We discuss more "exotic" types of CPDs, which use fewer parameters, in Section A.3.2.

If we assume that the parameters are time-invariant, we only need to specify $P(X_1)$, $P(X_2|X_1)$ and $P(Y_1|X_1)$; the CPDs for future slices are assumed to be the same as in the first two slices. This can massively reduce the amount of data needed to learn them. We can model this time-invariance assumption explicitly by viewing the parameters as random variables: see Figure 2.2.

One advantage of representing HMMs as DBNs is that it becomes easy to create variations on the basic theme. We discuss some of these models below.

2.3.1 HMMs with mixture-of-Gaussians output

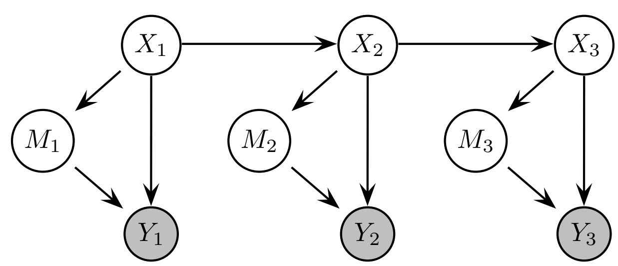


Figure 2.3: An HMM with mixture of Gaussians output.

In speech recognition, it is common to represent $P(Y_t|X_t = i)$ using a mixture of Gaussians (MoG) for each state i. We can either treat MoG as a primitive CPD type, or we can explicitly model the mixture variable as shown in Figure 2.3. The CPDs for the Y and M nodes are as follows:

$$
\begin{aligned}
P(Y_t|X_t = i, M_t = m) &= \mathcal{N}(y_t; \mu_{i,m}, \Sigma_{i,m}) \\
P(M_t = m|X_t = i) &= C(i, m)
\end{aligned}
$$

Suppose we modify the observation CPD so it first performs a linear projection of the data onto a subspace, i.e.,

$$P(Y_t|X_t = i, M_t = m) = \mathcal{N}(Ay_t; \mu_{i,m}, \Sigma_{i,m})$$

Maximum likelihood in this model (for a single t) is equivalent to heteroscedastic mixture discriminant analysis [KA96] (heteroscedastic since the covariance is not shared across classes (values of X_t), and mixture because of the M_t term). Unfortunately, there is no closed form solution for the M step in this case; if we had fully tied Σ, we could at least use standard (and fast) eigenvector methods (see [KA96] for details).

2.3.2 HMMs with semi-tied mixtures

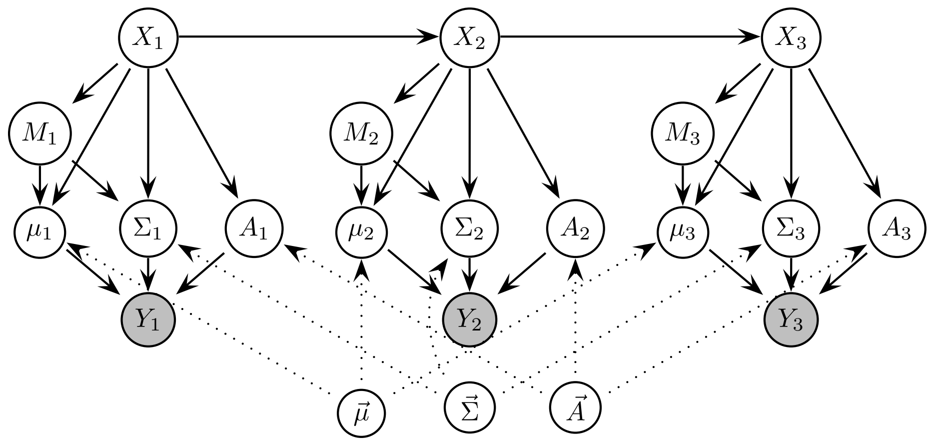


Figure 2.4: An HMM with semi-tied covariance matrices. The (set of) global parameters are the nodes at the bottom with outgoing dotted arcs. The local parameters for a slice, μ_t, Σ_t, and A_t, are determinsitically chosen from the global parameters using M_t and optionally X_t.

In speech recognition, it is common that $Y_t \in \mathbb{R}^{39}$, so estimating a full covariance matrix for each state of X_t and M_t requires a lot of data. An alternative is to assume there is a single global pool of Gaussians, and each state corresponds to a different mixture over this pool. This is called a semi-continuous or tied-mixture HMM [Gal99]. In this case, there is no arc from X_t to Y_t, so the CPD for Y becomes

$$P(Y_t|M_t = m) = \mathcal{N}(y_t; \mu_m, \Sigma_m)$$

The effective observation model becomes

$$P(Y_t|X_t = i) = \sum_m P(M_t = m|X_t = i)\mathcal{N}(y_t; \mu_m, \Sigma_m)$$

Hence all information about Y_t gets to X_t via the "bottleneck" M_t. The advantages of doing this are not only reducing the number of parameters, but also reducing the number of Gaussian density calculations, which speeds up inference dramatically. For instance, the system can calculate $\mathcal{N}(y_t; \mu_m, \Sigma_m)$ for each m, and then reuse these in a variety of different models by simple reweighting: typically $P(M_t = m|X_t = i)$ is non-zero for only a small number of m's [Jel97, p161]. The $M_t \to Y_t$ arc is like vector quantization, and the $X_t \to M_t$ arc is like a dynamic reweighting of the codebook.

More sophisticated parameter tying schemes are also possible. Figure 2.4 represents an HMM with semi-tied covariance matrices [Gal99]. The CPDs for this model are as follows:

$$\begin{aligned}
P(Y_t = y|\mu_t, \Sigma_t, A_t) &= \mathcal{N}(y_t; \mu_t, A_t\Sigma_t A_t') \\
P(\mu_t|\vec{\mu}, M_t = m, X_t = i) &= \delta(\mu_t, \mu_{i,m}) \\
P(A_t|\vec{A}, X_t = i) &= \delta(A_t, A_i) \\
P(\Sigma_t|\vec{\Sigma}, M_t = m, X_t = i) &= \delta(\Sigma_t, \Sigma_{m,i})
\end{aligned}$$

Typically each $\Sigma_{m,i}$ is assumed to be diagonal, but A_i converts this into a full matrix. In practice, we can "compile out" the deterministic CPDs, so that the overall graph looks like Figure 2.3, but where the observation CPD is defined as follows:

$$P(Y_t = y|M_t = m, X_t = i) = \mathcal{N}(y_t; \mu_{i,m}, A_i \Sigma_{i,m} A_i')$$

To compute ML parameter estimates in this model, we must use a nonlinear optimization method in the M step of EM (see Section C.4.2), since the covariance is bilinear in A_i and $\Sigma_{i,m}$. We can further reduce the number of parameters by associating an A matrix with a set of states of X_t, instead of having one matrix per state.

2.3.3 Auto-regressive HMMs

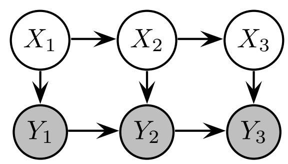


Figure 2.5: An auto-regressive HMM.

The standard HMM assumption that $Y_t \perp Y_{t'}|X_t$ is quite strong, and can be relaxed at little extra cost as shown in Figure 2.5. We will call this model an auto-regressive HMM (AR-HMM).[2] The AR-HMM model reduces the effect of the X_t "bottleneck", by allowing Y_{t-1} to help predict Y_t as well; this often results in models with higher likelihood.

If Y is discrete (e.g., for language modelling: see Section 2.3.5), the CPD for Y can be represented as a table. If Y is continuous, one possibility for the CPD for Y is

$$P(Y_t = y_t|X_t = i, Y_{t-1} = y_{t-1}) = \mathcal{N}(y_t; W_i y_{t-1} + \mu_i, \Sigma_i)$$

where W_i is the regression matrix given that X_t is in state i. This model is also known as a correlation HMM, conditionally Gaussian HMM switching regression model, switching Markov model [Ham90], or switching regression model.

2.3.4 Buried Markov Models

Buried Markov models [Bil98] generalize auto-regressive HMMs by allowing non-linear dependencies between the observable nodes. Furthermore, the nature of the dependencies can change depending on the value of X_t: see Figure 2.6 for an example. (Such a model is called a Bayesian "multi net".) [Bil00] introduces the EAR metric, which is a way to learn such structures in a discriminative way.[3]

2.3.5 Mixed-memory Markov models

As we mentioned in the introduction, one of the simplest approaches to modelling sequential data is to construct a Markov model of order $n-1$. These are often called n-gram models, e.g., when $n=2$, we get a bigram model, and when $n=3$, we get a trigram model, as shown in Figure 2.7.

When X_t is a discrete random variable with many possible values (e.g., if X_t represents words), then there might not be enough data to reliably estimate $P(X_t = k|X_{t-1} = j, X_{t-2} = i)$. A common approach is to create a mixture of lower-order Markov models:

$$P(X_t|X_{t-1}, X_{t-2}) = \alpha_3(X_{t-1}, X_{t-2}) f(X_t|X_{t-1}, X_{t-2}) + \alpha_2(X_{t-1}, X_{t-2}) f(X_t|X_{t-1}) + \alpha_1(X_{t-1}, X_{t-2}) f(X_t)$$

[2] Confusingly, the term "auto-regressive HMM" refers to two different models, the one being discussed here and the one in [Rab89], which is also called a linear predictive or hidden filter HMM, which looks like a regular HMM when represented as a DBN, but which uses a state-dependent auto-correlation function for the observation distribution.

[3] The basic idea is to avoid modelling features of the data which are common to all classes, but instead to focus on aspects of the structure that affect the decision boundaries.

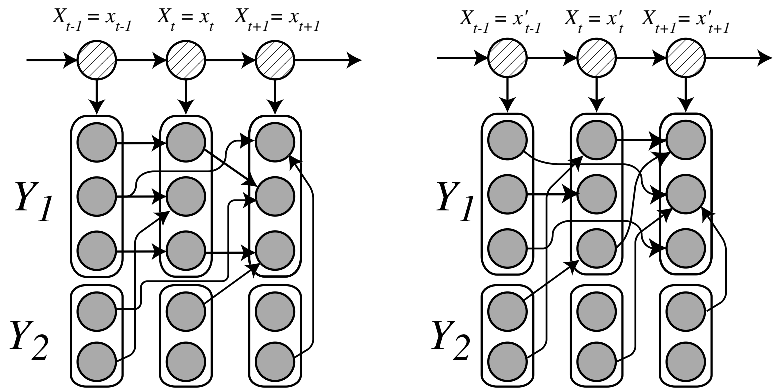


Figure 2.6: A buried Markov model. Depending on the value of the hidden variables, X_t, the effective graph structure between the components of the observed variables, Y_t, can change. Two different instantiations are shown. Thanks to Jeff Bilmes for this figure.

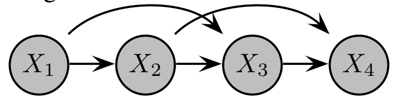


Figure 2.7: A trigram (second-order Markov) model, which defines $P(X_t|X_{t-1}, X_{t-2})$.

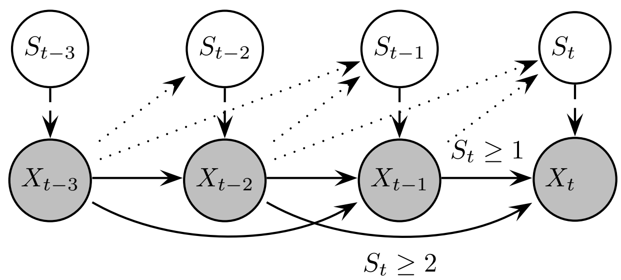


Figure 2.8: A mixed-memory Markov model. The dashed arc from S_t to X_t means S_t is a switching parent: it is used to decide which of the other parents to use, either X_{t-1} or X_{t-2} or both. The conditions under which each arc is active are shown for the X_t node only, to reduce clutter. The dotted arcs from X_{t-1} and X_{t-2} to S_t are optional. See text for details. Based on Figure 15 of [Bil01].

where the α coefficients may optionally depend on the history, and $f(\cdot)$ is an arbitrary (conditional) probability distribution. This is the basis of the method called deleted interpolation [Jel97, CG96].

As with mixtures of Gaussians, we can either represent mixtures of multinomials as a primitive CPD, or we can explicitely model the latent mixture variable as in Figure 2.8. Here $S_t = 1$ means use a unigram, $S_t = 2$ means use bi- and unigrams, and $S_t = 3$ means use tri-, bi- and uni-grams. S_t is called a switching parent, since it determines which of the other parents links are active, i.e., the effective topology of the graph changes depending on the value of the switch; the result is called a (dynamic) Bayesian multinet [Bil00]. Since S_t is hidden, the net effect is to use a mixture of all of these. The arcs from X_{t-2} and X_{t-1} to S_t model the fact that the coefficients can depend on the identity of the words in the window.

A very similar approach, called mixed-memory Markov models [SJ99], is to always use mixtures of bigrams: if $S_t = 1$, we use $P(X_t|X_{t-1})$, and if $S_t = 2$, we use $P(X_t|X_{t-2})$. This can be achieved by simply changing the "guard condition" on the X_{t-1} to X_t link to be $S_t = 1$ instead of $S_t \geq 1$; similarly, the guard on the X_{t-2} to X_t link becomes $S_t = 2$ instead of $S_t \geq 2$. Hence S_t acts like the input to a multiplexer, choosing one of the parents to feed into X_t; Using the terminology of [BZ02], S_t is the switching parent, and the other parents are the conditional parents. The overall effect is to define the CPD as follows:

$$P(X_t|X_{t-1}, \ldots, X_{t-n}) = \sum_{i=1}^{n} A^i(X_{t-i}, X_t)\pi(i)$$

An alternative is to cluster the words and then try to predict the next word based on the current cluster; this is called a class-based language model. The cluster identity is a hidden random variable, as in a regular HMM; the words are the observations.

Of course, the mixed-memory and class-based techniques can be combined by using an auto-regressive HMM, to get the best of both worlds.

2.3.6 Input-output HMMs

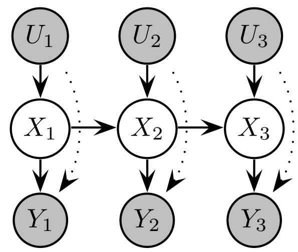


Figure 2.9: An input-output HMM. The dotted arcs are optional.

An input-output HMM [BF96] is a probabilistic mapping from inputs, $U_{1:T}$, to outputs, $Y_{1:T}$. (We relax the requirement that the input and output be the same length in Section 2.3.11.) This can be modelled as a DBN as shown in Figure 2.9. If the inputs are discrete, the CPD for X can be represented as a 3-dimensional array, $P(X_t = j|X_{t-1} = i, U_t = k) = A(i, j, k)$. If the input is continuous-valued, the CPD for X can be represented as a conditional softmax function (see Section A.3.2), or as a neural network.

2.3.7 Factorial HMMs

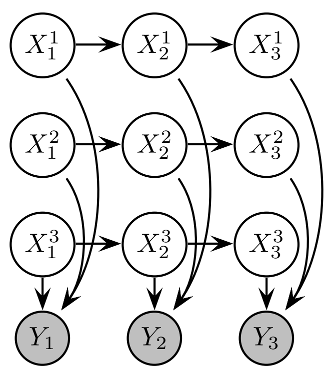


Figure 2.10: A factorial HMM with 3 chains.

Factorial HMMs [GJ97] use a single output variable but have a distributed representation for the hidden state, as shown in Figure 2.10. Note that, although all the chains are a priori independent, once we condition on the evidence, they all become coupled; this is due to the explaining away phenomenon [Pea88]. This makes inference intractable if there are too many chains, as we discuss in Chapter 3.

The CPDs for the hidden nodes, $P(X_t^{(d)}|X_{t-1}^{(d)})$, can be represented by $N_x \times N_x$ matrices (assuming each chain can be in one of N_x states). A naive representation for the CPD for the observed nodes, $P(Y_t|X_t^{(1)}, \ldots, X_t^{(D)})$, which we shall abbreviate to $P(Y_t|X_t^{(1:D)})$, would require $O({N_x}^D)$ parameters, for each possible combination of the parents. We discuss some more parsimonious representations in Section A.3.2.

A factorial HMM, like any DBN, can be converted to a regular HMM by creating a single "mega" variable, X_t, whose state space is the Cartesian product of the component state spaces. However, this is a bad idea: the resulting "flat" representation is hard to interpret, inference in the flat model will be exponentially slower (see Chapter 3), and learning will be harder because there may be exponentially many more parameters. In particular, although the entries of the transition matrix of the flat HMM would have constraints between them, it is hard to exploit these constraints either in inference or learning. For example, if $D = 2$, we have

$$P(X_t^1 = j_1, X_t^2 = j_2 | X_{t-1}^1 = i_1, X_{t-1}^2 = i_2) = P(X_t^1 = j_1 | X_{t-1}^1 = i_1) \times P(X_t^2 = j_2 | X_{t-1}^2 = i_2).$$

That is, the entries of the transition matrix for the flat HMM are computed by multiplying together the entries of the transition matrices for each chain in the factorial HMM.

If all CPDs are linear-Gaussian, graphical structure corresponds to sparse matrices in the traditional sense of having many zeros (see Section 2.4.2); in such a case, the compound DBN can be converted to a flat model without loss of efficiency or information.

2.3.8 Coupled HMMs

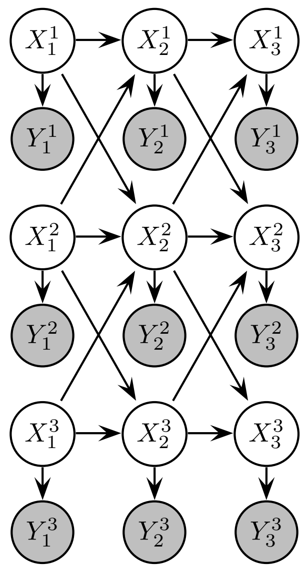


Figure 2.11: A coupled HMM with 3 chains.

In a coupled HMM [SJ95, Bra96], the hidden variables are assumed to interact locally with their neighbors. Also, each hidden node has its own "private" observation. This is shown in Figure 2.11.

I have used CHMMs for two different applications: audio-visual speech recognition (AVSR) and for modelling freeway traffic. In the AVSR application, there are two chains, one represents the audio stream and one representing the video stream; please see [NLP$^+$02] for details. [NY00] also used a two-chain CHMM for automatic speech recognitions (ASR). To reduce the number of parameters, they defined $P(X_t^1 | X_{t-1}^1, X_{t-1}^0)$ as a mixture of two bigram models (c.f., Section 2.3.5):

$$P(X_t^1 | X_{t-1}^1, X_{t-1}^0) = A^{11}(X_t^1 | X_{t-1}^1) P(S_t = 1) + A^{10}(X_t^1 | X_{t-1}^0) P(S_t = 0)$$

In the traffic application, X_t^d represents the hidden traffic status (free-flowing or congested) at location d on the freeway at time t. Y_t^d represents the reading returned by a magnetic loop detector, which (roughly speaking) is an indicator of traffic speed. The underlying traffic state at location d and time t is assumed to depend on its previous state and the previous states of its immediate spatial neighbors. Since the detectors are placed in a line down the middle of the highway, each location only has 2 neighbors, upstream and downstream. Please see [KM00] for details.

2.3.9 Hierarchical HMMs (HHMMs)

The Hierarchical HMM (HHMM) [FST98] is an extension of the HMM that is designed to model domains with hierarchical structure and/or dependencies at multiple length/time scales. In an HHMM, the states of the stochastic automaton can emit single observations or strings of observations. Those that emit single observations are called “production states”, and those that emit strings are termed “abstract states”. The strings emitted by abstract states are themselves governed by sub-HHMMs, which can be called recursively. When the sub-HHMM is finished, control is returned to wherever it was called from; the calling context is memorized using a depth-limited stack.

Example of an HHMM

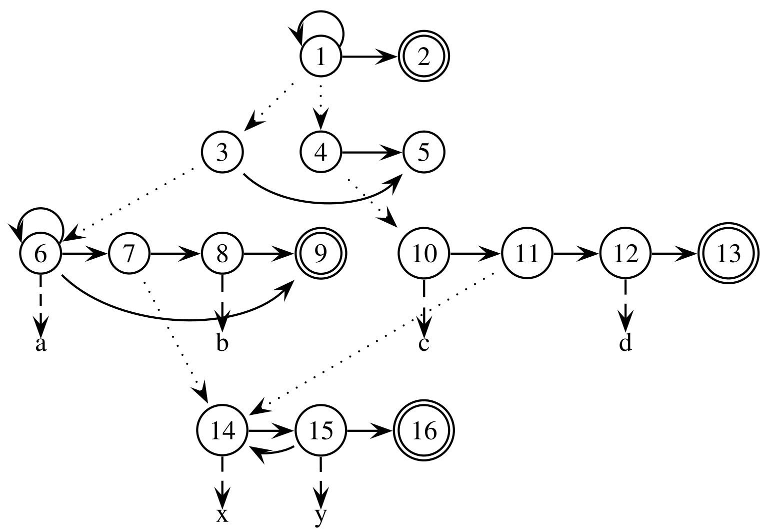


Figure 2.12: State transition diagram for a four-level HHMM representing the regular expression $a^+|a^+(xy)^+b|c(xy)^+d$. Solid arcs represent horizontal transitions between states; dotted arcs represent vertical transitions, i.e., calling a sub-HMM. Double-ringed states are accepting (end) states; we assume there is at least one per sub-HMM; when we enter such a state, control is returned to the parent (calling) state. Dahsed arcs are emissions from production (concrete) states; In this example, each production state emits a single symbol, but in general, it can have a distribution over output symbols.

We illustrate the generative process with Figure 2.12, which shows the state transition diagram of an example HHMM which models the regular expression $a^+|a^+(xy)^+b|c(xy)^+d$.[4] We start in the root state. Since this is an abstract state, it calls its sub-HHMM, entering into state 3 or 4; this is called a “vertical transition”. Suppose it enters state 3. Since 3 is abstract, it enters its child HMM via its unique entry point, state 6. Since 6 is a production state, it emits the symbol “a”. It may then loop around and exit, or make a horizontal transition to 7. State 7 then enters its sub-HMM (state 14), which emits “x”. State 14 makes a horizontal transition to state 15, and then emits “y”. Suppose at this point we make a horizontal transition to state 16, which is the end state for this sub-HMM. This returns control to wherever we were called from — in this case, state 7. State 7 then makes a horizontal transition to state 8, which emits “b”. State 8 is then forced to make a horizontal transition to the end state, which returns control to state 3. State 3 then enters its end state (5), and returns control to the root (1). The root can then either re-enter its sub-HMM, or enter its end state, which terminates the process.

An HHMM cannot make a horizontal transition before it makes a vertical one; hence it cannot produce the empty string. For example, in Figure 2.12, it is not possible to transition directly from state 1 to 2.

Related models

While HHMMs are less powerful than stochastic context-free grammars (SCFGs) and recursive transition networks (RTNs) [JM00], both of which can handle recursion to an unbounded depth, unlike HHMMs, they are sufficiently expressive for many practical problems, which often only involve tail-recursion (i.e., self

[4] This means the model must produce one or more ’a’s, or one or more ’a’s followed by one or ’x’s and ’y’s followed by a single ’b’, or a ’c’ followed by one or more ’x’s and ’y’s followed a ’d’.

transitions to an abstract state). Furthermore, HHMMs can be made much more computationally efficient than SCFGs. In particular, while the original inference algorithm for HHMMs was $O(T^3)$, as for SCFGs, once we have converted the HHMM to a DBN (see Section 2.3.9), we can use any of the techniques in Chapter 3 to do inference in $O(T)$ time [MP01].[5]

HHMMs are also closely related to cascades of finite state automata [Moh96, PR97b], but are fully probabilistic. A somewhat similar model, called a cascaded Markov model, was proposed in [Bra99b]; in this model, each HMM state represents a non-terminal in an SCFG, which can be used to generate a substring. Other HMM-SCFG hybrids have been proposed; for example, [IB00] suggest creating one HMM to recognize each terminal symbol, and then combing them using an SCFG. Unfortunately, inference and learning in such models is much more complicated than with an HHMM, not only because inference takes $O(T^3)$ time, but also because the output of the bank of HMMs is a probability distribution over symbols at each position, as opposed to a single symbol as the standard SCFG inference routines expect.

[PW96] discuss a way of combining static Bayes nets with SCFGs, but their goal is to represent a probability distribution over the parse trees obtained from an SCFG. [ND01] show how Bayes nets can be used as a model of how humans combine cues when parsing ambiguous sentences online.

Converting an HHMM to an HMM

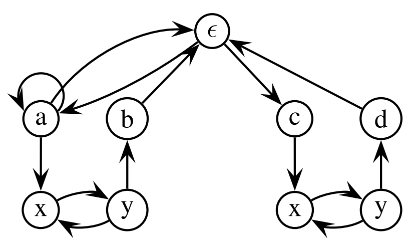


Figure 2.13: State transition diagram of the HHMM in Figure 2.12 flattened to a regular HMM. Each state is labelled by the symbol it emits; ϵ represents the empty string. The initial and final states are omitted.

Any HHMM can be converted to a regular HMM by creating an HMM state for every leaf in the HHMM state transition diagram (i.e., every legal HHMM stack configuration). The resulting state-space may be smaller, because it does not contain abstract states, but may also be larger, because shared sub-structure (such as the sub-expression $(xy)^+$) must be duplicated. See Figure 2.13 for an example.

Computing the parameters of the resulting flat HMM is not always trivial. The probability of an i to j transition in the flat model is the sum over all paths from i to j in the hierarchical model which only pass through abstract (non-emitting) states. For example, in Figure 2.13, the probability of a self-transition from state a to a is given by

$$P_{flat}(a \to a) = P_h(6 \to 6) + P_h(6 \to 9 \to 3 \to 6) = P_h(6|6) + P_h(9|6)P_h(6|3)$$

where P_h represents probabilities in the HHMM.

The HMMs used in speech recognition are essentially hierarchical (see Section 2.3.10), but are always flattened into a single level state space for speed and ease of processing. Simiarly, combinations of weighted transducers [Moh96, PR97b] are always flattened into a single transducer before use. Although it is always possible to convert an HHMM to an HMM, just as with any DBN, there are several disadvantages to doing so:

- Flattening loses modularity, since the parameters of the sub-HMMs get combined in a complex way, as we have just seen.

[5]The inference algorithm for SCFGs, which is called the inside-outside algorithm (see e.g., [JLM92, JM00]), takes $O(T^3)$ no matter what the grammar is.

- A flat HMM cannot easily provide a multi-scale interpretation of the data.[6]
- Training HMMs separately and combining them into a hierarchical model requires segmented data. (This is the approach adopted in [HIM$^+$00] for example.)
- The parameters in a flat HMM are constrained, but it is hard to exploit these constraints during inference and learning, because the constrained parameters are products of the free parameters, as we saw in the example above, and in the case of flattening factorial HMMs in Section 2.3.7. By contrast, it is easy to do parameter estimation in an HHMM. Furthermore, the fact that sub-models are re-used in different contexts can be represented, and hence learned, using an HHMM, but not using an HMM.

Representing the HHMM as a DBN

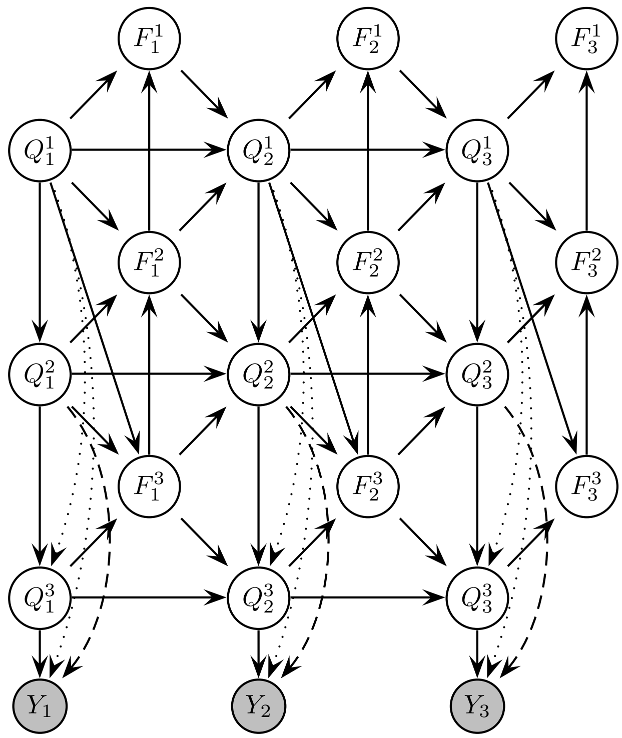


Figure 2.14: A 3-level HHMM represented as a DBN. Q_t^d is the state at time t, level d; $F_t^d = 1$ if the HMM at level d has finished (entered its exit state), otherwise $F_t^d = 0$. Shaded nodes are observed; the remaining nodes are hidden. In some applications, the dotted arcs from Q^1 and the dashed arcs from Q^2 may be omitted.

We can represent the HHMM as a DBN as shown in Figure 2.14. (We assume for simplicity that all production states are at the bottom of the hierarchy; this restriction is lifted in Section 2.3.9.) The state of the HMM at level d and time t is represented by Q_t^d. The state of the whole HHMM is encoded by the vector $\vec{Q}_t = (Q_t^1, \ldots, Q_t^D)$; intuitively, this encodes the contents of the stack, that specifies the complete "path" to take from the root to the leaf state in the state transition diagram.

F_t^d is a binary indicator variable that is "on" (has value 1) if the HMM at level d and time t has just "finished" (i.e., is about to enter an end state), otherwise it is "off" (has value 0). Note that if $F_t^d = 1$, then $F_t^{d'} = 1$ for all $d' > d$; hence the number of F nodes that are "off" represents the effective height of the "context stack", i.e., which level of the hierarchy we are currently on.

[6] It is also possible to interpret a single integer, representing the flat state, as a vector of integers, representing the HHMM state. However, sometimes it is useful to explicitly represent the different variables, instead of doing this post-processing interpretation: see Section 2.3.10 for example.

The downward arcs between the Q variables represent the fact that a state "calls" a sub-state. The upward arcs between the F variables enforce the fact that a higher-level HMM can only change state when the lower-level one is finished. This ensures proper nesting of the parse trees, and is the key difference between an HHMM and a hidden Markov decision tree [JGS96]. (An HMDT looks similar to Figure 2.14, except it lacks the F nodes and the upward going arcs.)

We now define the conditional probability distributions (CPDs) of each of the node types below, which will complete the definition of the model. We consider the bottom, middle and top layers of the hierarchy separately (since they have different local topology), as well as the first, middle and last time slices.

Bottom level ($d = D$, $t = 2 : T-1$): Q^D follows a Markov chain with parameters determined by which sub-HMM it is in, which is encoded by the vector of higher-up state variables $Q_t^{1:D-1}$, which we will represent by the integer k for brevity.[7] Instead of Q^D entering its end state, it "turns on" F^D instead, to mean it is finished (see Section 2.3.9 for more details); This will be a signal that higher-level HMMs can now change state. In addition, it will be a signal that the next value of Q^D should be drawn from its prior distribution (representing a vertical transition), instead of its transition matrix (representing a horizontal transition). Formally, we can write this as follows:

$$P(Q_t^D = j | Q_{t-1}^D = i, F_{t-1}^D = f, Q_t^{1:D-1} = k) = \begin{cases} \tilde{A}_k^D(i,j) & \text{if } f = 0 \\ \pi_k^D(j) & \text{if } f = 1 \end{cases}$$

where we have assumed $i, j \neq$ end, where end represents the end-state for this HMM. A_k^D is the transition matrix for level D given that the parent variables are in state k, and $\tilde{A}_k^D$ is just a rescaled version of this (see Section 2.3.9 for details on the rescaling). Similarly, π_k^D is the initial distribution for level D given that the parent variables are in state k. The equation for F_D is simply

$$P(F_t^D = 1 | Q_t^{1:D-1} = k, Q_t^D = i) = A_k^D(i, \text{end}).$$

Intermediate levels ($d = 2 : D-1$, $t = 2 : T-1$): As before, Q^d follows a Markov chain with parameters determined by $Q^{1:d-1}$, and F^d specifies whether we should use the transition matrix or the prior. The difference is that we now also get a signal from below, F^{d+1}, specifying whether the sub-model has finished or not; if it has, we are free to change state, otherwise we must remain in the same state. Formally, we can write this as follows:

$$P(Q_t^d = j | Q_{t-1}^d = i, F_{t-1}^{d+1} = b, F_{t-1}^d = f, Q_t^{1:d-1} = k) = \begin{cases} \delta(i,j) & \text{if } b = 0 \\ \tilde{A}_k^d(i,j) & \text{if } b = 1 \text{ and } f = 0 \\ \pi_k^d(j) & \text{if } b = 1 \text{ and } f = 1 \end{cases} \tag{2.1}$$

F^d should "turn on" only if Q^d is "allowed" to enter a final state, the probability of which depends on the current context $Q^{1:d-1}$. Formally, we can write this as follows:

$$P(F_t^d = 1 | Q_t^d = i, Q_t^{1:d-1} = k, F_t^{d+1} = b) = \begin{cases} 0 & \text{if } b = 0 \\ A_k^d(i, \text{end}) & \text{if } b = 1 \end{cases} \tag{2.2}$$

Top level ($d = 1$, $t = 2 : T-1$): The top level differs from the intermediate levels in that the Q node has no Q parent to specify which distribution to use. The equations are the same as above, except we eliminate the conditioning on $Q_t^{1:d-1} = k$. (Equivalently, we can imagine a dummy top layer HMM, which is always in state 1: $Q_t^0 = 1$. This is often how HHMMs are represented, so that this top-level state is the root of the overall parse tree, as in Figure 2.12.)

Initial slice ($t = 1$, $d = 1 : D$): The CPDs for the nodes in the first slice are as follows: $P(Q_1^1 = j) = \pi^1(j)$ for the top level and $P(Q_1^d = j | Q_1^{1:d-1} = k) = \pi_k^d(j)$, for $d = 2, \ldots, D$.

Final slice ($t = T$, $d = 1 : D$): To ensure that all sub-HMMs have reached their end states by the time we reach the end of sequence, we can clamp $F_T^d = 1$ for all d. (In fact, it is sufficient to clamp $F_T^1 = 1$, since this implies $F_T^d = 1$ for all d.) A similar trick was used in [Zwe98], to force all segmentations to be consistent with the known length of the sequence.

[7] If the state transition diagram is sparse, many of these joint configurations will be impossible; k will only index the valid configurations. Such sparse CPDs can be conveniently represented using decision trees (see Section A.3.2).

Observations: If the observations are discrete symbols, we may represent $P(O_t|\vec{Q}_t)$ as a table. If the observations are real-valued vectors, we can use a Gaussian for each value of $\vec{Q}_t$. We discuss more parsimonious representations in Section A.3.2. Alternatively, we can condition O_t only on some of its parents. This is often done in speech recognition: the acoustic observations depend only on the bottom two levels of the hierarchy (the sub-phones and phones), and not on the top level (the words): see Section 2.3.10 for details.

Handling end states

Unlike the automaton representation, the DBN never actually enters an end state (i.e., Q_t^d can never taken on the value "end"); if it did, what should the corresponding observation be? (Unlike an HMM, a DBN must generate an observation at every time step.) Instead, Q_t^d causes F_t^d to turn on, and then enters a new (non-terminal) state at time $t+1$, chosen from its initial state distribution. This means that the DBN and HHMM transition matrices are not identical. However, they satisfy the following equation:

$$\tilde{A}_k^d(i,j)\left(1 - A_k^d(i,\text{end})\right) = A_k^d(i,j)$$

where A represents the HHMM transition matrix, $\tilde{A}$ represents the DBN transition matrix, and $A_k^d(i,\text{end})$ is the probability of terminating from state i. The reason is that the probability of each horizontal transition in the DBN gets multiplied by the probability that $F_t^d = 0$, which is $1 - A_k^d(i,\text{end})$; this product should match the original probability. It is easy to see that the new matrix is stochastic (i.e., $\sum_{j=1}^K \tilde{A}_k^d(i,j) = 1$), since the original matrix satisfies $\sum_{j=1}^K A_k^d(i,j) + A_k^d(i,\text{end}) = 1$, where K is the number of states.

Allowing production states at different levels

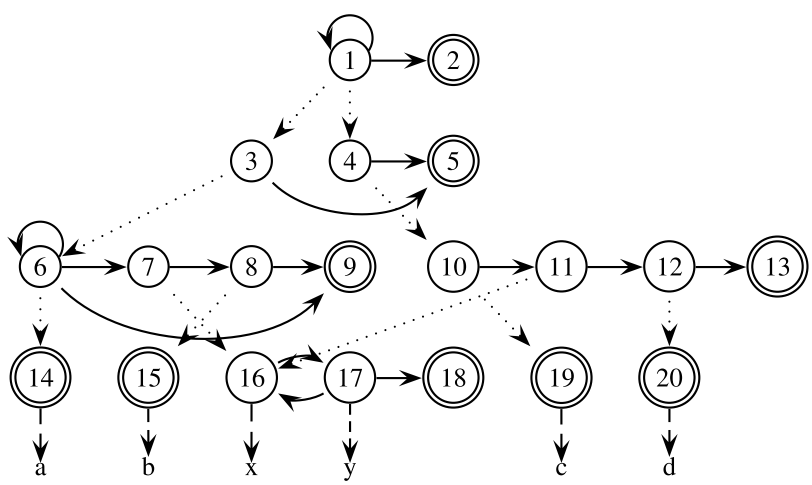


Figure 2.15: State transition diagram of the HHMM in Figure 2.12, except all production states are at the leaves.

In [FST98], the authors remark that putting all the production (concrete) states at the bottom level did not result in learning "interesting" hierarchical structure (i.e., the results were essentially the same as using a flat HMM). They therefore constructed the automaton topology by hand, and allowed production states at multiple levels of the hierarchy, at randomly chosen locations. (We discuss HHMM learning in Chapter 6; for now we assume that the parameters — and hence the topology of the state-transition diagram — are known.)

So far, the DBN representation has assumed that all the production states are at the bottom level. This can always be made the case, as illustrated in Figure 2.15. However, we now develop a way to directly encode in the DBN the fact that concrete states can occur at different levels in the hierarchy. We simply ensure that when Q_t^d enters a concrete state, it causes all the Q nodes below it to enter a special "uninformative" or "null" state. In addition, all the F nodes below level d will be forced on, so that level d is free to change state without waiting for its children to finish. The observation distribution will ignore null states, so the distribution simplifies from $P(O_t|Q_t^{1:D})$ to $P(O_t|Q_t^{1:d})$ if Q_t^d is a concrete state. It is straightforward to modify the CPD definitions to cause this behavior.

The above solution is somewhat inefficient since it introduces an extra dummy value, thus increasing the size of the state space of each level by one. A more elegant solution is to allow some arcs to "disappear" conditionally on the values of the Q nodes, i.e., let the Q nodes act both as regular nodes and as switching parents (c.f., Section 2.3.5). Specifically, when Q_t^d enters a concrete state, all the outgoing links from nodes $Q_t^{d'}$ and $F_t^{d'}$, for $d' > d$, would be rendered effectively (i.e., parametrically) redundant. It is straightforward to modify the CPD definitions to cause this behavior.

Advantages of representing the HHMM as a DBN

There are several advantages of representing the HHMM as a DBN rather than using the original formulation [FST98] in terms of nested state-transition diagrams. First, we can use generic DBN inference and learning procedures, instead of having to derive (fairly complicated) formulas by hand. Second, exact DBN inference takes $O(Q^D T)$ time, whereas the algorithm proposed in [FST98] takes $O(Q^D T^3)$ time. Third, it is easier to change the model once it is represented as a DBN, e.g., to allow factored hierarchical state spaces.

2.3.10 HHMMs for Automatic speech recognition (ASR)

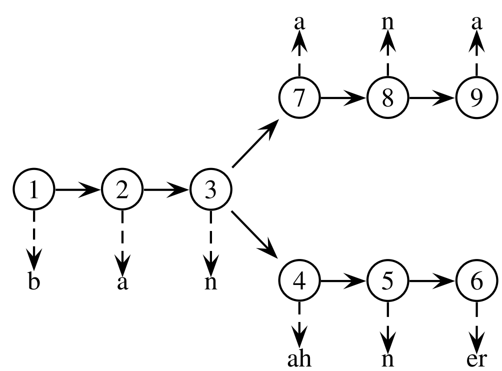


Figure 2.16: HMM state transition diagram for a model encoding two possible pronunciations of "banana", the British version (bottom fork: middle "a" is long, and final "a" has "r" appended), and the American version (top fork: all "a"s are equal). Dotted lines from numbers to letters represent deterministic emissions. The states may have self-loops (not shown) to model duration.

Consider modelling the pronunciation of a single word. In the simplest case, the can be described as a sequence of phones, e.g., "dog" is "d - o - g". This can be represented as a (left-to-right) finite state automaton. But consider a word like "yamaha": the first "a" is followed by "m", but the second "a" is followed by "h". Hence some automaton states need to share the same phone labels. (Put another way, given a phone label, we cannot always uniquely identify our position within the automaton/word.) This suggests that we use an HMM to model word pronunciation. Such a model can also cope with multiple pronunciations, e.g., the British or American pronunciation of "banana" (see Figure 2.16).[8]

Let Q_t^h be the hidden automaton state at time t, and Q_t be the observed phone. For the yamaha model, we have $Q_t^h \in \{1, \ldots, 9\}$ and $Q_t \in$ {/y/, /aa/, /m/, /a/, /h/}. Note that $P(Q_t = k|Q_t^h = q) = \delta(B(q), k)$ is a delta function (e.g., state 2 only emits "aa"). Also, the transition matrix, $P(Q_t^h = q'|Q_{t-1}^h = q) = A(q, q')$, can always be made upper diagonal by renumbering states, and will typically be very sparse.

Now suppose that we do not observe the phones directly, but that each phone can generate a *sequence* of acoustic vectors; we will model the acoustic "appearance" and duration of each phone with a phone HMM. The standard approach is to embed the appropriate phone HMMs into each state of the word HMM. However, we then need to tie the transition and observation parameters between different states.

[8]The "yamaha" example is from Jeff Bilmes, the "banana" example is from Mark Paskin.

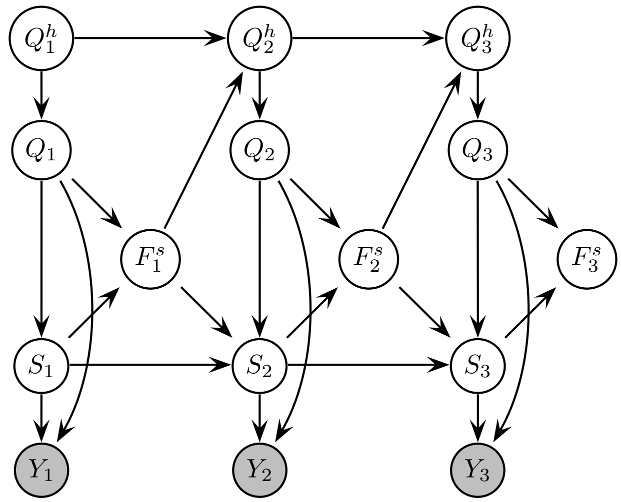


Figure 2.17: A DBN for modelling the pronunciation of a single word. Q^h is the state (position) in the word HMM; Q is the phone; S is the state (position) within the phone HMM, i.e., the subphone; Y is the acoustic vector. F^s is a binary indicator variable that turns on when the phone HMM has finished.

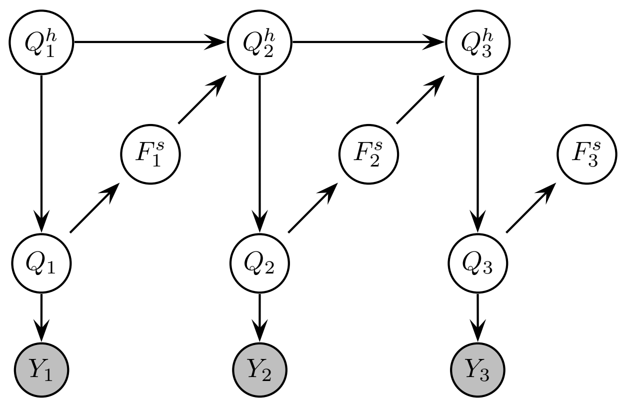


Figure 2.18: A DBN for modelling the pronunciation of a single word, where each phone is modelled by a single state. This is the simplification of Figure 2.17 if S has only one possible value. This corresponds to the model used in [Zwe98].

An alternative is to make the parameter tying graphically explicit by using HHMMs; this allows the tying pattern to be learned. Specifically, let each state of the word model, Q_t^h, emit Q_t which then "calls" the phone HMM. The state of the phone HMM at time t is S_t; this is called the subphone. Since we don't know how long each phone lasts, we introduce F^S which only turns on, allowing Q_t^h to make a transition, when the phone HMM has finished. This can be accomplished as shown in Figure 2.17. F^S is conditioned on Q_t, not Q_t^h, which represents the fact that the duration depends on the phone, not the automaton state (i.e., duration information is tied). Similarly, Y_t is conditioned on Q_t but not Q_t^h.

Normally the phone HMMs are 3-state left-to-right HMMs. However, if we use a single state HMM, we can simplify the model. In particular, if S_t can have only one possible value, it can be removed from the graph; the duration of the phone will be determined by the self-loop probability on the corresponding hidden state in the word HMM. The resulting DBN is shown in Figure 2.18, and corresponds to the model used in [Zwe98]. If the word HMM has a strict left-to-right topology, Q_t^h deterministically increases by 1 iff $F_t^S = 1$; hence $P(F_t^S = 1|Q_t = k) = 1 - A(i,i)$, where i is any state that (deterministically) emits the phone k. If the word HMM is not left-to-right, then F^S can specify which out-arc to take from state Q_{t-1}^h, so $P(Q_t^h|Q_{t-1}^h, F_{t-1}^s)$ becomes deterministic [ZR97].

The above was a model for a single word. If we have a sequence of words, we add another layer to the hierarchy, and condition the variables inside the word HMM (i.e., Q^h and Q) on the (hidden) identity of the word: see Figure 2.19. The fact that the duration and appearance are not conditioned on the word represents the fact that the phones are shared across words: see Figure 2.20. Sharing the phone models dramatically reduces the size of the state space. Nevertheless, with, say, 6000 words, 60 phones, and 3 subphones, there are still about 1 million unique states, so a lot of training data is required to learn such models. (In reality, things are even worse because of the need to use triphone contexts, although these are clustered, to avoid having 60^3 combinations.)

Note that the number of legal values for Q_t^h can vary depending on the value of W_t, since each word has a different-sized HMM pronunciation model. Also, note that silence (a gap between words) is different from silent (non-emitting) states: silence has a detectable acoustic signature, and can be treated as just another word (whose duration is of course unknown), whereas silent states do not generate any observations.

We now explicitly define all the CPDs for $t > 1$ in the model shown in Figure 2.19.

$$
\begin{aligned}
P(W_t = w'|W_{t-1} = w, F_{t-1}^W = f) &= \begin{cases} \delta(w, w') & \text{if } f = 0 \\ A(w, w') & \text{if } f = 1 \end{cases} \\
P(F_t^W = f|Q_t^h = q, W_t = w, F_t^S = b) &= \begin{cases} \delta(f, 0) & \text{if } b = 0 \\ 1 - A_w(q, \text{end}) & \text{if } b = 1 \text{ and } f = 0 \\ A_w(q, \text{end}) & \text{if } b = 1 \text{ and } f = 1 \end{cases} \\
P(Q_t^h = q'|Q_{t-1}^h = q, W_t = w, F_{t-1}^W = f, F_{t-1}^S = b) &= \begin{cases} \delta(q, q') & \text{if } b = 0 \\ A_w(q, q') & \text{if } b = 1 \text{ and } f = 0 \\ \pi_w(q') & \text{if } b = 1 \text{ and } f = 1 \end{cases} \\
P(Q_t = k|Q_t^h = q, W_t = w) &= \delta(B_w(q), k) \\
P(F_t^s = 1|S_t = j, Q_t = k) &= A_k(j, \text{end}) \\
P(S_t = j|S_{t-1} = i, Q_t = k, F_{t-1} = f) &= \begin{cases} A_k(i, j) & \text{if } f = 0 \\ \pi_k(j) & \text{if } f = 1 \end{cases}
\end{aligned}
$$

where $A(w, w')$ is the word bigram probability; $A_w(q, q')$ is the transition probability between states in word model w, $\pi_w(q)$ is the initial state distribution, and $B_w(q)$ is the phone deterministically emitted by state q in the HMM for word w, and similarly, $A_k(i, j)$ $\pi_k(j)$ and $B_k(j, y_t)$ are the parameters for the k'th phone HMM. We could model $P(Y_t|S_t = j, Q_t = k)$ using a mixture of Gaussians, for example.

Training HHMMs for ASR

When training an HMM from a known word sequence, we can use the model in Figure 2.21. We specify the sequence of words, $\vec{w}$, but not when they occur. If $W_t^h = k$, it means we should use the k'th word at time t; hence $P(W_t = w|W_t^h = k, \vec{w}) = \delta(w, w_k)$ is a delta function, c.f., the relationship between Q^h and Q. Also, W^h increases by 1 iff $F^W = 1$, to indicate the next word in the sequence.

In practice, we can "compile out" the $\vec{w}$ node, by changing the CPDs for the W nodes for each training sequence. If the mapping from words to phones is fixed, we can also compile out W^h, W, and F^W, resulting in the model shown in Figure 2.17 (or Figure 2.18 if we don't use subphones). Now Q^h acts as an index into the known (sub)phone sequence corresponding to $\vec{w}$, so its CPD becomes deterministic (increase Q^h by 1 iff $F_{t-1}^S = 1$, which occurs iff the phone HMM has finished), and the CPD for the phone becomes $P(Q_t = k|Q_t^h = i) = B_w(q, k)$, if i corresponds to state q in word HMM w. This is equivalent to combining all the word HMMs together, and giving all their states unique numbers, which is the standard way of training HMMs from a known phonetic sequence.

Note that the number of possible values for W^h in Figure 2.21 is now T_{in}, the number of words in the input training sequence. One concern is that this might make inference intractable. The simplest approach to inference is to combine all the interface variables (see Section 3.4) into one "mega" variable, and then use a (modified) forwards-backwards procedure, which has complexity $O(T_{out}N^2)$, where N is the number of states of the mega variable, and T_{out} is the length (number of frames) of the output (observed) sequence. The interface variables (the ones with outgoing temporal arcs) in Figure 2.21 are as follows: $W_t^h \in \{1, \ldots, T_{in}\}$, $Q_t^h \in \{1, \ldots, P\}$, $S_t \in \{1, \ldots, K\}$, $F_t^W \in \{0, 1\}$, $F_t^S \in \{0, 1\}$, where P is the max number of states in any word model, and K is the maximum number of states in each phone HMM (typically 3). Hence it would

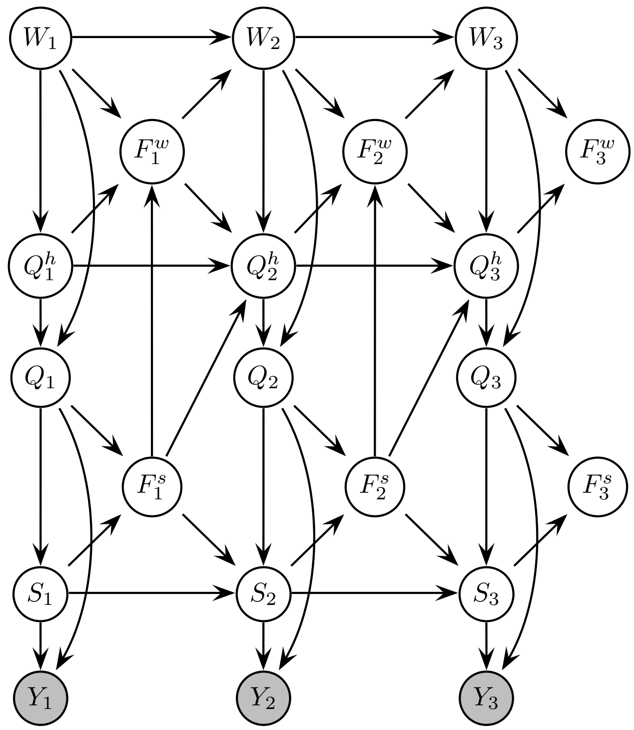


Figure 2.19: A DBN for continuous speech recognition. Note that the observations, Y_t, the transition probabilities, S_t, and the termination probabilities, F_t^S, are conditioned on the phone Q_t, not the position within the phone HMM Q_t^h, and not on the word, W_t.

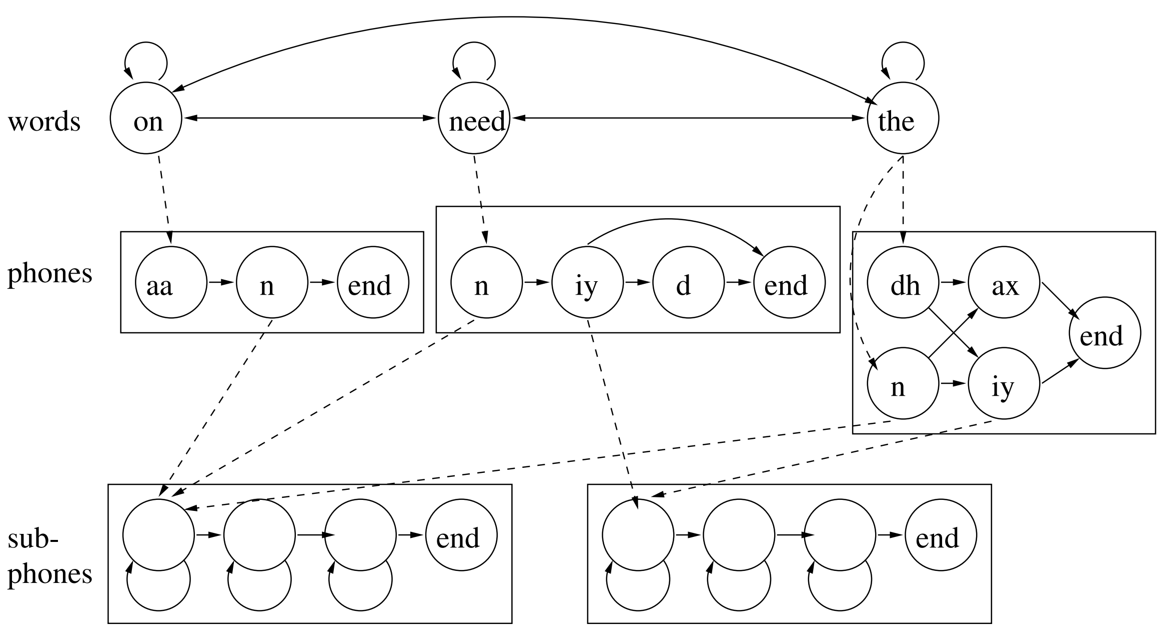


Figure 2.20: An example HHMM for an ASR system which can recognize 3 words (adapted from [JM00]). The phone models (bottom level) are shared (tied) amongst different words; only some of them are shown.

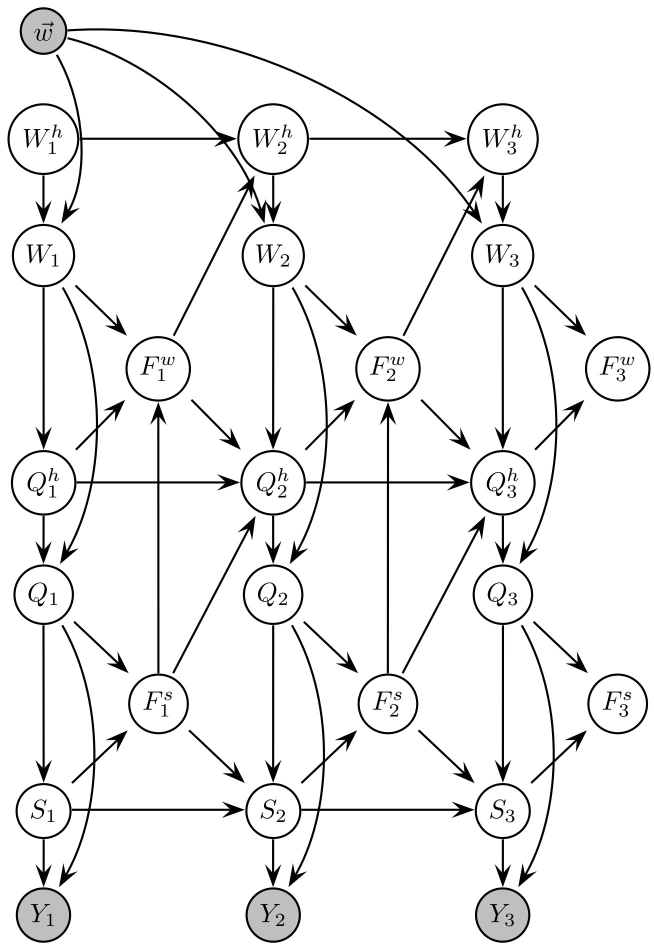


Figure 2.21: A DBN for training a continuous speech recognition system from a known word sequence $\vec{w}$. In practice, the $\vec{w}$ node can be absorbed into the CPDs for W. Also, if the mapping from words to phones is fixed, we can also compile out W^h, W, and F^W, resulting in the much simpler model shown in Figure 2.17 (or Figure 2.18 if we don't use subphones).

appear that exact smoothing requires $O(T_{out}T_{in}^2P^2K^2)$ time. However, because of the left-to-right nature of the transition matrices for W^h, Q^h and S, this can be reduced to $O(T_{out}T_{in}PK)$ time. This is the same as for a standard HMM (up to constant factors), since there are $N = T_{in}PK$ states created by concatenating the word HMMs for the whole sequence.

When training, we know the length of the sequence, so we only want to consider segmentations that "use up" all the phones. We can do this by introducing an extra "end of sequence" (eos) variable, that is clamped to some observed value, say 1, and whose parents are Q_T^h and F_T^s; we define its CPD to be $P(\text{eos} = 1|Q_T^h = i, F_T^s = f) = 1$ iff i is the last phone in the training sequence, and $f = 1$. (The reason we require $f = 1$ is that this represents the event that we are just about to transition to a (silent) accepting state.) This is the analog of only examining paths that reach the "top right" part of the HMM trellis, where the top row represents the final accepting state, and the right column represents the final time slice. This trick was first suggested in [Zwe98]. A simpler alternative is simply to set $Q_T^h = i$ and $F_T^s = 1$ as evidence; this has the advantage of not needing any extra variables, although it is only possible if there is only one state (namely i) immediately preceeding the final state (which will be true for a counting automaton of the kind considered here).

General DBNs for ASR

So far, we have focussed on HHMMs for ASR. However, it is easy to create more flexible models. For example, the observation nodes can be represented in factored form, instead of as a homogeneous vector-valued node. (Note that this only affects the computation of the conditional likelihood terms, $B_t(i,i) = P(y_t|Q_t = i)$, so the standard forwards-backwards algorithm can be used for inference.) More general hidden nodes can also be used, representing the state of the articulators, e.g., position of the tongue, size of mouth opening, etc [Zwe98]. If the training set just consists of standard speech corpora, there is nothing to force the hidden variables to have this "meaningful" intrepretation. [Row99, RBD00] trained on data where they were able to observe the articulators directly; the resulting models are much easier to intrepret. See [Bil01] for a more general review of how graphical models can be used for ASR.

2.3.11 Asynchronous IO-HMMs

When training an HHMM for ASR, we are essentially learning a probabilistic mapping from $\vec{w} = W_{1:T_{in}}$ to $Y_{1:T_{out}}$, where typically $T_{in} \ll T_{out}$. This is like an asychronous input-output HMM [BB96], which can handle input and output sequences of different lengths. (Note that the complexity of the inference algorithm for asychronous IO-HMMs in [BB96] is $O(T_{in}T_{out})$, just like the DBN method. By contrast, inference is a synchronous IO-HMM, where $T_{in} = T_{out} = T$, only takes time $O(T)$.)

Bengio [Ben99] suggested using IO-HMMs for ASR, treating the acoustic sequence as input and the words as output, the opposite of what we discussed above. There are two problems with this. First, we may be forced to assign high probability to unlikely observation sequences; this is because we are conditioning on the acoustics instead of generating them: if there is only one outgoing arc from a state, it must have probability one no matter what the input is, because we normalize transition probabilities on a per-state level. This has been called the "label bias problem" [MFP00]. A second and more important problem is that the "meaning" of the states in an IO-HMM is different than in an HMM. In particular, the states summarize past acoustics, rather than past words, so the kind of sharing of phone models we discussed above, that is vital to successful performance, cannot be used.[9]

2.3.12 Variable-duration (semi-Markov) HMMs

The self-arc on a state in an HMM defines a geometric distribution over waiting times. Specifically, the probability we remain in state i for exactly d steps is $p(d) = (1-p)p^{d-1}$, where $p = A(i,i)$ is the self-loop probability. To allow for more general durations, one can use a semi-Markov model. (It is called semi-Markov because to predict the next state, it is not sufficient to condition on the past state: we also need to know how long we've been in that state.) We can model this as a special kind of 2-level HHMM, as shown in Figure 2.22. The reason there is no Q to F arc is that the termination process is deterministic. In particular,

[9] Y. Bengio, personal communication, 3/18/02.

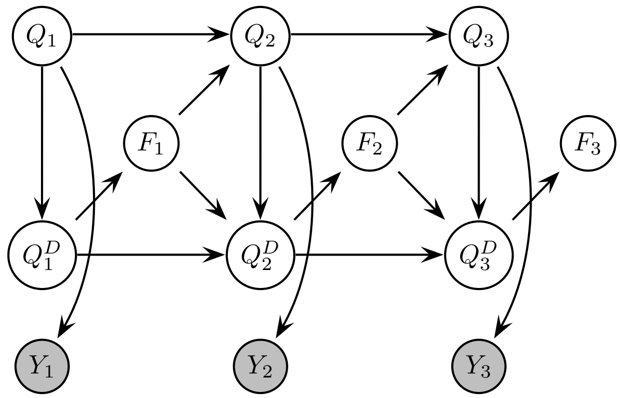


Figure 2.22: A variable-duration HMM modelled as a 2-level HHMM. Q_t represents the state, and Q_t^D represents how long we have been in that state (duration).

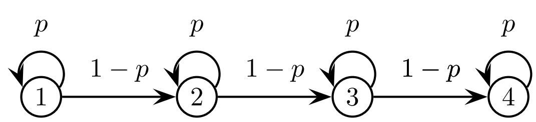
the bottom level counts how long we have been in a certain state; when the counter expires (reaches 0), the F node turns on, the parent node Q can change state, and the duration counter, Q^D, resets. Hence the CPD for the bottom level is as follows:

$$
\begin{aligned}
P(Q_t^D = d' | Q_{t-1}^D = d, Q_t = k, F_{t-1} = 1) &= p_k(d') \\
P(Q_t^D = d' | Q_{t-1}^D = d, Q_t = k, F_{t-1} = 0) &= \begin{cases} 1 & \text{if } d > 1 \text{ and } d' = d - 1 \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
$$

Note that $p_k(d)$ could be represented as a table (a non-parametric approach) or as some kind of parametric function. If $p_k(d)$ is a geometric distribution, this emulates a standard HMM.

The naive approach to inference in this HHMM takes $O(TD^2K^2)$ time, where D is the maximum number of steps we can spend in any state, and K is the number of HMM states; this is the same complexity as the original algorithm [Rab89]. However, we can exploit the fact that the CPD for Q^D is deterministic to reduce this to $O(TDK^2)$. This is faster than the original algorithm because we do dynamic programming on the joint state-space of Q and Q^D, instead of just Q. Also, we can apply standard inference and learning procedures to this model, without having to derive the (somewhat hairy) formulas by hand.

A more efficient, but less flexible, way to model non-geometric waiting times is to replace each state with n new states, each with the same emission probabilities as the original state [DEKM98, p69]. For example, consider this model.



Obviously the smallest sequence this can generate is of length $n = 4$. Any path of length l through the model has probability $p^{l-n}(1-p)^n$; the number of possible paths is $\binom{l-1}{n-1}$, so the total probability of a path of length l is

$$p(l) = \binom{l-1}{n-1} p^{l-n}(1-p)^n$$

This is the negative binomial distribution. By adjusting n and the self-loop probabilities of each state, we can model a wide range of waiting times.

2.3.13 Mixtures of HMMs

[Smy96] defines a mixture of HMMs, which can be used for clustering (pre-segmented) sequences. The basic idea is to add a hidden cluster variable as a parent to all the nodes in the HMM; hence all the distributions are conditional on the (hidden) class variable. A generalization of this, which I call a dynamical mixture of HMMs, adds Markovian dynamics to the mixture/class variable, as in Figure 2.23. This model has several

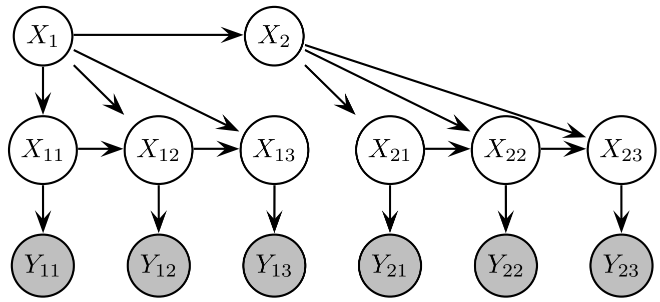


Figure 2.23: A dynamical mixture of HMMs, aka an embedded HMM. Here we have 2 sequences, both of length 3. Obviously we could have more sequences of different lengths, and multiple levels of the hierarchy. A static mixture of HMMs would combine X_2 and X_1.

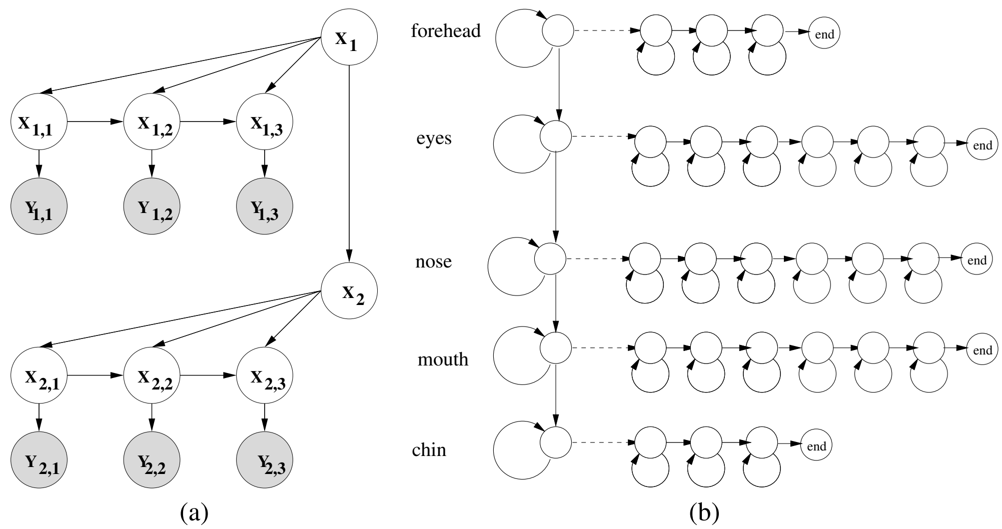


Figure 2.24: An embedded HMM for face recognition represented (a) as a DBN and (b) as the state transition diagram of an HHMM (dotted arcs represent calling a sub-HMM). We assume there are 2 rows and 3 columns in the image, which is clear from the DBN representation, but not from the HHMM representation. The number of states in each sub-model can be different, which is clear from the HHMM but not the DBN. The DBN in (a) is just a rotated version of the one in Figure 2.23.

names in the literature: "hierarchical mixture of Markov chains" [Hoe01], "embedded HMM" [NI00], and "pseudo-2D" HMM [KA94]. The reason for the term pseudo-2D is because this model provides a way of extending HMMs to 2D data such as images. The basic idea is that each row of an image is modelled by a row HMM, all of whose states are conditioned on the state of the column HMM. The overall effect is to allow dynamic warping in both the horizontal and vertical directions (although the warping in each row is independent, as noted by Mark Paskin).

Nefian [NI00] used embedded (2D) HMMs for face recognition. In this model, the state of each row could be forehead, eyes, nose, mouth or chin; each pixel within each row could be in one of 3 states (if the row was in the forehead or chin state) or in one of 6 states otherwise. The "meaning" of the horizontal states was not defined; they just acted as positions within a left-to-right HMM, to allow horizontal warping of each feature. See Figure 2.24.

The key difference between an embedded HMM and a hierarchical HMM is that, in an embedded HMM, the ending "time" of each sub-HMM is known in advance, since each horizontal HMM models exactly C observations, where C is the number of columns in the image. Hence the problem of deciding when to return control to the (unique) parent state is avoided. This makes inference in embedded HMMs very easy.

Embedded HMMs should not be confused with switching HMMs, shown in Figure 2.25. This is different from an embedded HMM because each substate emits a single observation, not a fixed-length vector; also, it is different from a 2-level HHMM, because there is nothing forcing the top level to change more slowly than the bottom level.

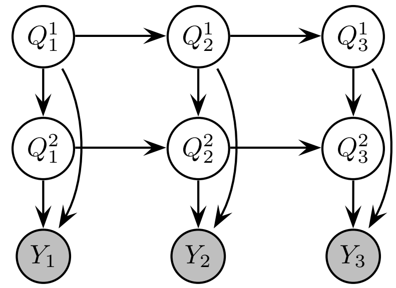


Figure 2.25: A switching HMM, aka a 2-level hidden Markov decision tree. This is different from a 2-level HHMM, because there is nothing forcing the top level to change more slowly than the bottom level.

2.3.14 Segment models

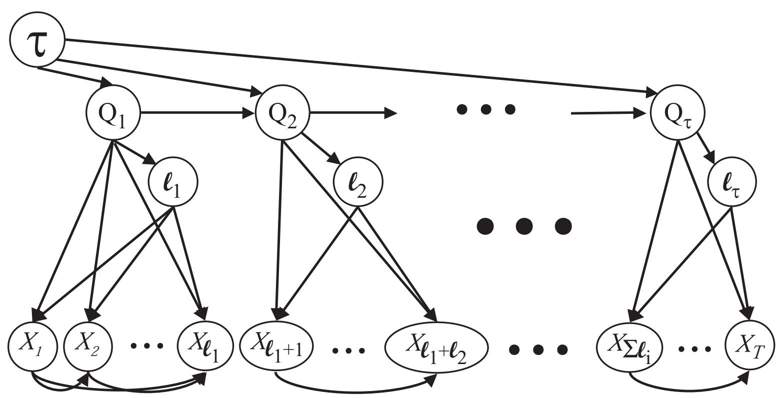


Figure 2.26: A schematic depiction of a segment model, from [Bil01]. The X_t nodes are observed, the rest are hidden.

The basic idea of the segment model [ODK96] is that each HMM state can generate a sequence of observations instead of just a single observation. The difference from an HHMM is that the length of the sequence generated from state q_i is explicitly determined by a random variable l_i, as opposed to being implicitly determined by when its sub-HHMM enters its end state. Furthermore, the total number of segments, τ, is a random variable. Hence the likelihood is given by

$$P(y_{1:T}) = \sum_{\tau} \sum_{q_{1:\tau}} \sum_{l_{1:\tau}} \prod_{i=1}^{l_i} P(\tau) P(q_i | q_{i-1}, \tau) P(l_i | q_i) P(y_{t_0(i):t_1(i)} | q_i, l_i)$$

where and $t_0(i) = \sum_{j=1}^{i-1} l_j + 1$ and $t_1(i) = t_0(i) + l_i - 1$ are the start and ending times of segment i. A first attempt to represent this as a graphical model is shown in Figure 2.26. This is not a fully specified graphical model, since the L_i's are random variables, and hence the topology is variable. Also, there are some dependencies that are not shown (e.g., the fact that $\sum_{i=1}^{\tau} l_i = T$), and some dependencies might not exist (e.g., the Y_i's in a segment may not be fully interconnected). We will give a more accurate DBN representation of the model below.

Let us consider a particular segment and renumber so that $t_0 = 1$ and $t_1 = l$. In a variable-length HMM,

$$P(y_{1:l} | q, l) = \prod_{k=1}^{l} P(y_k | q)$$

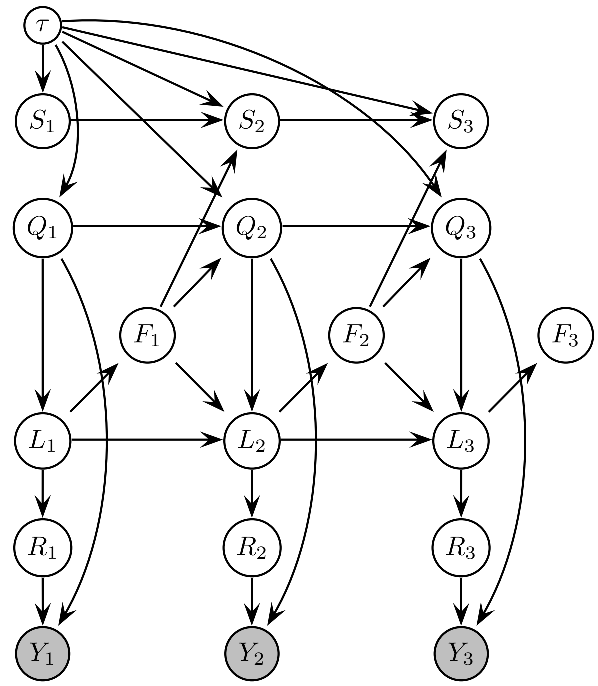


Figure 2.27: A stochastic segment model.

(If $p(l|q)$ is a geometric distribution, this becomes a regular HMM.) A simple generalization is to condition on some function of the position within the segment:

$$P(y_{1:l}|q,l) = \prod_{k=1}^{l} P(y_k|q,r_k)$$

where r_k is the region corresponding to position k within the segment. This can be modelled as shown in Figure 2.27. We have added a deterministic counter, S_t, which counts the number of segments. We constrain $S_T = \tau$, rather like the end-of-sequence trick for HHMMs (see Section 2.3.10). The L_t nodes is a deterministic down-counter, and turns on F when it reaches zero, as in a semi-Markov model (see Section 2.3.12).

Alternatively, we can define a conditional Gaussian segment model [DRO93]

$$P(y_{1:l}|q,l) = \prod_{k=1}^{l} P(y_k|y_{k-1},q,r_k)$$

We just modify Figure 2.27 by adding an arc from Y_{t-1} to Y_t; however, we want this arc to "disappear" when $F_{t-1} = 1$, since that indicates the end of a segment. Hence we need to add an arc from F_{t-1} to Y_t as well. We then define

$$P(Y_t|Y_{t-1}, R_t = r, Q_t = q, F_{t-1} = f) = \begin{cases} \mathcal{N}(y_t; A_{r,q}y_{t-1}, \Sigma_{r,q}) & \text{if } f = 0 \\ \mathcal{N}(y_t; \mu_{r,q}, \Sigma_{r,q}) & \text{if } f = 1 \end{cases}$$

In general, $P(y_{1:l}|q,l)$ can be an arbitrary joint distribution, e.g., consider an undirected graphical model connecting y_k to y_{k-1} and y_{k+1}. However, representing this graphically can get messy.

2.3.15 Abstract HMMs

[BVW00a] describes the "Abstract HMM" (AHMM), which is very closely related to HHMMs. (They have 3 slightly different models in [BVW00a, BVW00b, BVW01]; we consider the version in [BVW00b].) These

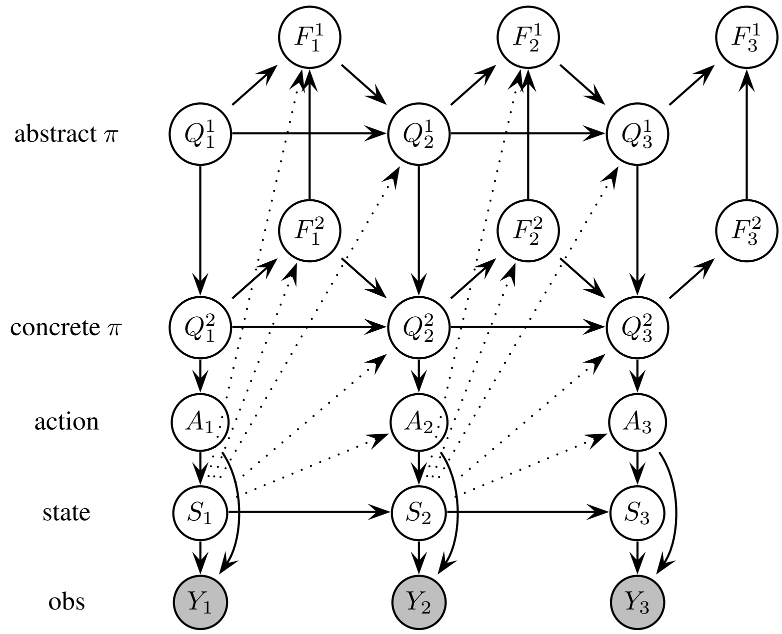


Figure 2.28: The DBN for a 3-level abstract HMM, from [BVW00b]. Note that we call the top layer level 1, they call it level D. Note also that we draw F_t^d above Q_t^d, they draw it below, but the topology is the same. The lines from the global world state S_t are shown dotted merely to reduce clutter. The top level encodes the probability of choosing an abstract policy: $P(Q_t^1 = \pi' | S_{t-1} = s) = \sigma(s, \pi')$. The second level encodes the probability of choosing a concrete policy: $P(Q_t^2 = \pi | S_{t-1} = s, Q_t^1 = \pi') = \sigma_{\pi'}(s, \pi)$. The third level encodes the probability of choosing a concrete action: $P(A_t = a | S_{t-1} = s, Q_t^2 = \pi) = \sigma_\pi(s, a)$. The last level encodes the world model: $P(S_t = s' | S_{t-1} = s, A_t = a)$.

authors are interested in inferring what policy an agent is following by observing its effects in the world. (We do not observe the world state, but we assume that the agent does.)

A (stochastic) policy is a (probabilistic) mapping from (fully observed) states to actions: $\sigma_\pi(s, a) = P(\text{do action } a | \text{in state } s)$. An abstract (stochastic) policy is a (probabilistic) mapping from states to lower level abstract policies, or "macro actions". An abstract policy can call a sub-policy, which runs until termination, returning control to the parent policy; the sub-policy can in turn call lower-level abstract policies, until we reach the bottom level of the hierarchy, where a policy can only produce concrete actions. [BVW00a] consider abstract policies of the "options" kind [SPS99], which is equivalent to assuming that there are no horizontal transitions within a level. (HAMs [PR97a] generalize this by allowing horizontal transitions (i.e., internal state) within a controller.) This simplifies the CPDs, as we see below.

This can be modelled by the DBN shown in Figure 2.28. It is obviously very similar to an HHMM, but is different in several respects, which we now discuss.

- There is a global world state variable S_t, that is a parent of everything.
- The bottom level represents a concrete action, which always finishes in one step; hence there is no $F_t^A = F_t^3$ node, and no A_{t-1} to A_t arc.
- There is no arc from F_t^2 to Q_{t+1}^1, since levels cannot make horizontal transitions (so F_t^1 turns on as soon as F_t^2 does).
- There is no arc from Q_t^1 to A_t or Y_t. In general, they assume Q_t^d only depends on its immediate parent, Q_t^{d-1}, not its whole context, $Q_t^{1:d-1}$. This substantially reduces the number of parameters (and allows for more efficient approximate inference[10]). However, it may also lose some information. For example,

[10]We can apply Rao-Blackwellised particle filtering to sample the F and S nodes; if each Q node has a single parent, the Q nodes form a chain, and can be integrated out exactly.

if the bottom level controller determines how to leave a room via a door, we may be able to predict the future location of the agent (e.g., will it leave via the left or right door) more accurately if we take into account the complete calling context (e.g., the top-level goals).

- There is no arc from Q_t^1 to F_t^2. They assume the termination probability depends on the current policy only, not on the calling context.

Given these assumptions, the CPDs simplify as follows.

$$P(F_t^d = 1|Q_t^d = \pi, S_t = s, F_t^{d+1} = b) = \begin{cases} 0 & \text{if } b = 0 \\ \beta_\pi(s) & \text{if } b = 1 \end{cases}$$

where $\beta_\pi(s)$ is the probability of terminating in state s given that the current policy is π.

$$P(Q_t^d = j|Q_{t-1}^d = i, Q_t^{d-1} = \pi, S_{t-1} = s, F_{t-1}^d = f) = \begin{cases} \delta(i,j) & \text{if } f = 0 \\ \sigma_\pi(s,j) & \text{if } f = 1 \end{cases}$$

where $\sigma_\pi(s,j)$ is the probability that abstract policy π chooses to call sub-policy j in state s.

1-level AHMMs

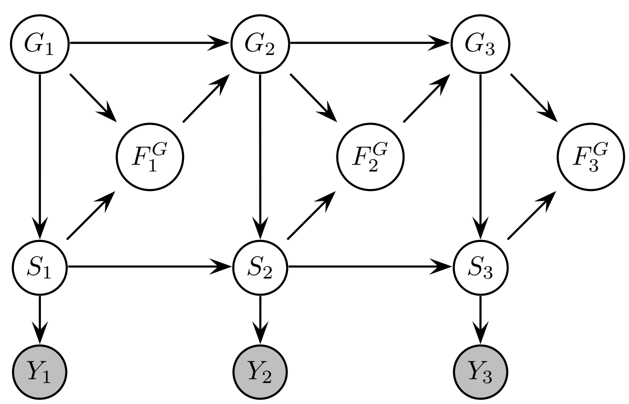


Figure 2.29: A 1-level AHMM. F_t^G turns on if state S_t statisfies goal G_t; this causes a new goal to be chosen. However, the new state may depend on the old state; hence there is no arc from F_{t-1}^G to S_t to “reset” the submodel.

In [BVW01], they consider the special-case of a 1-level AHMM, which is shown in Figure 2.29. This has a particularly simple interpretation: $Q_t^1 \equiv G_t$ represents the goal that the agent currently has, and $Q_t^2 \equiv S_t$ is the world state. It is similar to a variable duration HMM (see Section 2.3.12), except the duration depends on how long it takes to satisfy the goal.

The transition matrix for G is the same as for Q^1 in an HMMM, except we never reset.

$$P(G_t = j|G_{t-1} = i, F_{t-1}^G = f) = \begin{cases} \delta(i,j) & \text{if } f = 0 \\ A^G(i,j) & \text{if } f = 1 \end{cases}$$

([BVW01] consider the special case where $A^G(i,j)$ is a deterministic successor-state function for goals.)

The transition matrix for F^G is the same as for F^D in an HMMM:

$$P(F_t^G = 1|G_t = g, S_t = i) = A_g^S(i, \text{end})$$

The transition matrix for S is the same as for Q^D in an HHMM, except we never “reset”, since F^G is not a parent (since the next state always depends on the previous state, even if the goal changes):

$$P(S_t = j|G_{t-1} = g, S_{t-1} = i) = A_g^S(i,j)$$

2.4 Continuous-state DBNs

So far, all the DBNs we have looked at have only had discrete-valued hidden nodes. We now consider DBNs with continuous-valued hidden nodes, or mixed discrete-continuous systems (sometimes called hybrid systems). In this section, we adopt the (non-standard) convention of representing discrete variables as squares and continuous variables as circles.[11]

2.4.1 Representing KFMs as DBNs

The graph structure for a KFM looks identical to that for an HMM or an IO-HMM (see Figures 2.1 and Figure 2.9), since it makes the same conditional independence assumptions. However, all the nodes are continuous, and the CPDs are all linear-Gaussian, i.e.,

$$\begin{aligned} P(X_t = x_t | X_{t-1} = x_{t-1}, U_t = u) &= \mathcal{N}(x_t; Ax_{t-1} + Bu + \mu^X, Q) \\ P(Y_t = y | X_t = x, U_t = u) &= \mathcal{N}(y; Cx + Du + \mu^Y, R) \end{aligned}$$

In the linear-Gaussian case, but not the HMM case, there is a one-to-one correspondence between zeros in the parameter matrices and absent arcs in the graphical model, as we discuss in Section 2.4.2.

2.4.2 Vector autoregressive (VAR) processes

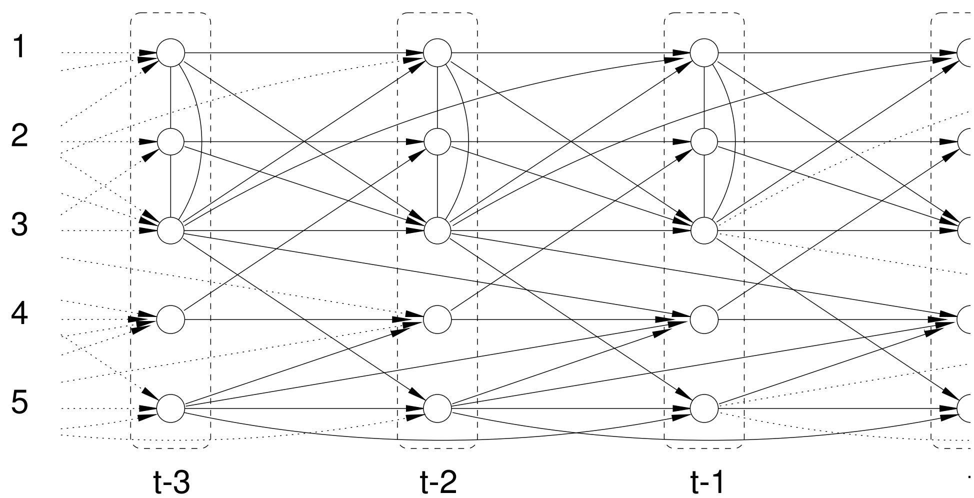


Figure 2.30: A VAR(2) process represented as a graphical model. From [DE00]. The link from $X^3_{t-3} \rightarrow X^4_{t-1}$ exists because $A_2(4,3) = \frac{1}{5} > 0$.

A vector autoregressive (VAR) processes of order L has the form

$$X_t = A_1 X_{t-1} + \cdots + A_L X_{t-L} + \epsilon_t$$

where $\epsilon_t \sim \mathcal{N}(0, \Sigma)$. This is equivalent to a DBN of order L where all CPDs are linear-Gaussian. There is a one-to-one correspondence between zeros in the regression matrices and absent inter-slice arcs of the DBN. Specifically, we can show that [DE00]

$$X^a_{t-u} \rightarrow X^b_t \text{ iff } A_u(b,a) \neq 0$$

[11] In an influence diagram [Sha88], square nodes represent decision or action variables; however, in this thesis, we will not consider influence diagrams. When we do have control variables, they will denoted as shaded round root nodes — shaded because we assume they are always known, round because we are agnostic about whether they are discrete or continuous, and roots because we assume they are exogeneous (we do not model what causes them).

for $1 \leq u \leq L$. If the intra-slice connections are modelled as undirected arcs (so the overall model becomes a time series chain graph), we can also show a one-to-one correspondence between zeros in the precision (concentration) matrix $K = \Sigma^{-1}$ and absent intra-slice arcs [Lau96]:

$$X_t^a \text{—} X_t^b \text{ iff } K(a,b) \neq 0$$

For example, consider the following VAR(2) process with parameters

$$A_1 = \begin{pmatrix} \frac{3}{5} & 0 & \frac{1}{5} & 0 & 0 \\ 0 & \frac{3}{5} & 0 & -\frac{1}{5} & 0 \\ \frac{2}{5} & \frac{1}{3} & \frac{3}{5} & 0 & 0 \\ 0 & 0 & 0 & -\frac{1}{2} & \frac{1}{5} \\ 0 & 0 & \frac{1}{5} & 0 & \frac{2}{5} \end{pmatrix}, \; A_2 = \begin{pmatrix} 0 & 0 & -\frac{1}{5} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{5} & 0 & \frac{1}{3} \\ 0 & 0 & 0 & 0 & -\frac{1}{5} \end{pmatrix}$$

and

$$\Sigma = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} & 0 & 0 \\ \frac{1}{2} & 1 & -\frac{1}{3} & 0 & 0 \\ \frac{1}{3} & -\frac{1}{3} & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \; \Sigma^{-1} = \begin{pmatrix} 2.13 & -1.47 & -1.2 & 0 & 0 \\ -1.47 & 2.13 & 1.2 & 0 & 0 \\ -1.2 & 1.2 & 1.8 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

This is illustrated in Figure 2.30.

Since sparse graphical structure is isomorphic to sparse matrices in a linear-Gaussian system, traditional techniques for representation (using block matrices), inference[12] and learning suffice: graphical models don't provide much added value. Hence in the rest of this section we focus on non-linear/ non-Gaussian and mixed discrete-continuous systems.

A model called a causality graph [DE00, Dah00, Eic01] can be derived from the time-series chain graph by aggregating X_t^i for all t into a single node X^i. The result is a graph in which nodes represent (components of) whole time series, and absence of edges correspond to what are called "Granger noncausality relationships" (see [DE00, Dah00, Eic01] for details). The causality graph can be further simplified to an undirected correlation graph by a procedure akin to moralization. In the resulting graph, the edge X^i — X^j is missing iff timeseries X^i and X^j are uncorrelated conditioned on all the remaining timeseries. If we have non-linear/ non-Gaussian relationships, uncorrelated must be replaced by the stronger notion of independent which, of course, is hard to measure in the non-discrete case (see [BJ01] for a promising kernel-based approach in the context of ICA).

2.4.3 Switching KFMs

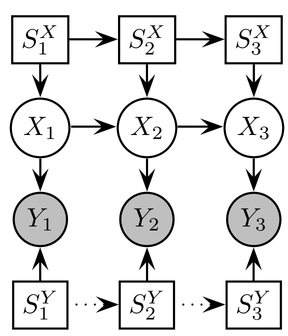


Figure 2.31: A switching Kalman filter model. The dotted arcs are optional. Square nodes are discrete, round nodes are continuous.

[12]In a VAR process, all the variables are assumed to be observed (this is the standard assumption in time-series analysis). Hence there is no need to do state estimation. Prediction is straightforward, since there are no future observations to condition on. The only challenging problem is structure learning (model selection), which we discuss in Chapter 6.

In Figure 2.31 we show a switching KFM, where we have omitted the input U for simplicity. (This model has various other names, including switching linear dynamical system (LDS), switching state-space model (SSM), jump-Markov model, jump-linear system, conditional dynamic linear model (DLM), etc. The term "jump" or "switch" implies S_t is discrete, whereas "conditional" leaves open the possibility that S_t is continuous.) Although it is traditional to have only one discrete switch variable, I have shown two, one for the dynamics and one for the observations, since their semantics is quite different (see below). The CPDs for this model are as follows:

$$\begin{aligned}
P(X_t = x_t | X_{t-1} = x_{t-1}, S_t^X = i) &= \mathcal{N}(x_t; A_i x_{t-1} + \mu_i^X, Q_i) \\
P(Y_t = y | X_t = x, S_t^Y = j) &= \mathcal{N}(y; C_j x + \mu_j^Y, R_j) \\
P(S_t^X = j | S_{t-1}^X = i) &= A^X(i,j) \\
P(S_t^Y = j | S_{t-1}^Y = i) &= A^Y(i,j)
\end{aligned}$$

Switching dynamics is useful for modelling piece-wise linear behavior (one way of approximating non-linear models), or multiple types or "modes" of behavior, e.g., I have succesfully used switching KFMs for visual tracking of people as they walk, run, skip, etc. [Mur98]; see also the manoeuvering plane example in Figure 1.4.

Switching observations can be used to approximate non-Gaussian noise models by a mixture, to model outliers (see Section 1.2.1), or to model data association ambiguity (see Section 2.4.6). Modelling outliers using mixtures of Gaussians was discussed in Section 1.2.1. Note that if the outlier is caused by the sensor being broken, we might expect such outliers to persist; this is modelled by the $S_{t-1}^Y \to S_t^Y$ arc (see Section 2.4.4). If we are just trying to approximate non-Gaussian noise, it would make more sense to remove the $S_{t-1}^Y \to S_t^Y$ arc.

Of course, we may have many discrete variables. For example, if X_t is vector valued, it is sometimes useful to associate one discrete variable for each component of X_t, so each can have its own piece-wise linear dynamics. Similarly, if Y_t is a vector, S^Y may be factored; this is useful for fault diagnosis applications (see Section 2.4.4). For another example, consider the problem of online blind deconvolution [LC95]. In this problem, we see

$$y_t = \sum_{i=0}^{q} \theta^i s_{t-i} + \epsilon_t$$

where the mixing coefficients θ are unknown, S_t is a hidden discrete random variable, and ϵ_t is a Gaussian noise term. For example, when $q = 2$,

$$\begin{pmatrix} s_t \\ s_{t-1} \\ s_{t-2} \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} s_{t-1} \\ s_{t-2} \\ s_{t-3} \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} s_t$$

We can represent this in matrix form as follows:

$$\begin{aligned}
\theta_t &= \theta_{t-1} \\
x_t &= H x_{t-1} + W s_t \\
y_t &= \theta_t' x_t + \epsilon_t
\end{aligned}$$

where $x_t = (s_{t:t-q})'$, $\theta_t = (\theta_t^{0:q})'$, $W = (1, 0, \ldots, 0)'$, and H is the matrix shown above (for the case $q = 2$): We can represent this as a DBN as shown in Figure 2.32. This is clearly a conditionally linear dynamical system.

Unfortunately, exact inference in switching KFMs is intractable, since the belief state at time t has $O(K^t)$ modes, where K is the number of discrete values, for reasons we explain in Section 3.6.3. We therefore need to use approximate inference. We discuss deterministic approximations in Section 4.3, and stochastic approximations in Section 5.3.1.

2.4.4 Fault diagnosis in hybrid systems

One of the most important applications of hybrid systems is fault diagnosis. Consider the model in Figure 2.33, which is a benchmark problem in the fault diagnosis community [MB99] (typically one considers

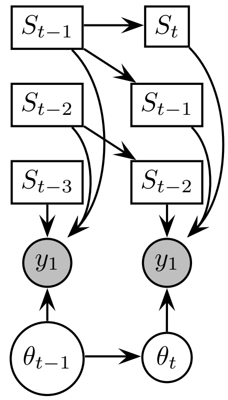


Figure 2.32: A DBN for blind deconvolution.

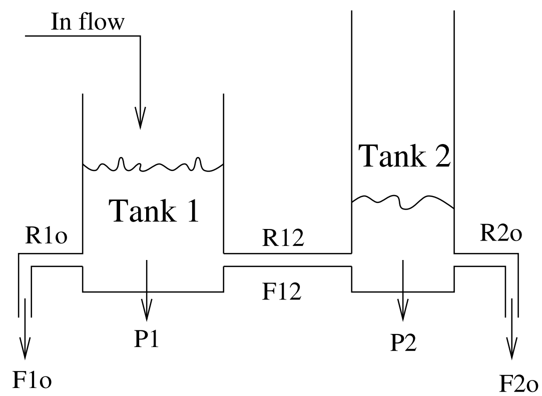


Figure 2.33: The two-tank system. The goal is to infer when pipes are blocked or have burst, or sensors have broken, from (noisy) observations of the flow out of tank 1, $F1o$, out of tank 2, $F2o$, or between tanks 1 and 2, $F12$. $R1o$ is a hidden variable representing the resistance of the pipe out of tank 1, $P1$ is a hidden variable representing the pressure in tank 1, etc. From Figure 11 of [KL01].

n tanks, for $n \gg 2$). This is a nonlinear system, since flow = pressure / resistance (or flow = pressure $\times$ conductance). More problematically, the values of the resistances can slowly drift, or change discontinuously due to burst pipes. Also, the sensors can fail intermittently and give erroneous results. We can model all of this using a DBN as shown in Figure 2.34. As in any switching KFM, inference in this model is intractable, but particle filtering [KL01] and other approximate inference algorithms [LP01] have been used successfully on this model.

2.4.5 Combining switching KFMs with segment models

Although inference in general switching KFMs is intractable, there are tractable special cases. A trivial one is where the switch nodes don't actually switch. More interesting is the model in Figure 2.35 (based on [DRO93]), where the switch nodes are piecewise constant: they remain fixed during a segment, which is modelled using a single KFM, but are allowed to switch at segment boundaries (whose locations are of course unknown). In addition, the model is assumed to reset once it crosses a segment boundary, i.e., the state in the first slice of a new segment is independent of the state in the last slice of the previous segment. (This is not evident from the graph structure, but is implicit in the CPDs.) Hence we can reason about segment independently (conditioned on knowing the boundaries), by running K Kalman filters per segment; we use the forwards-backwards algorithm to figure out boundary locations. For details, see [DRO93].

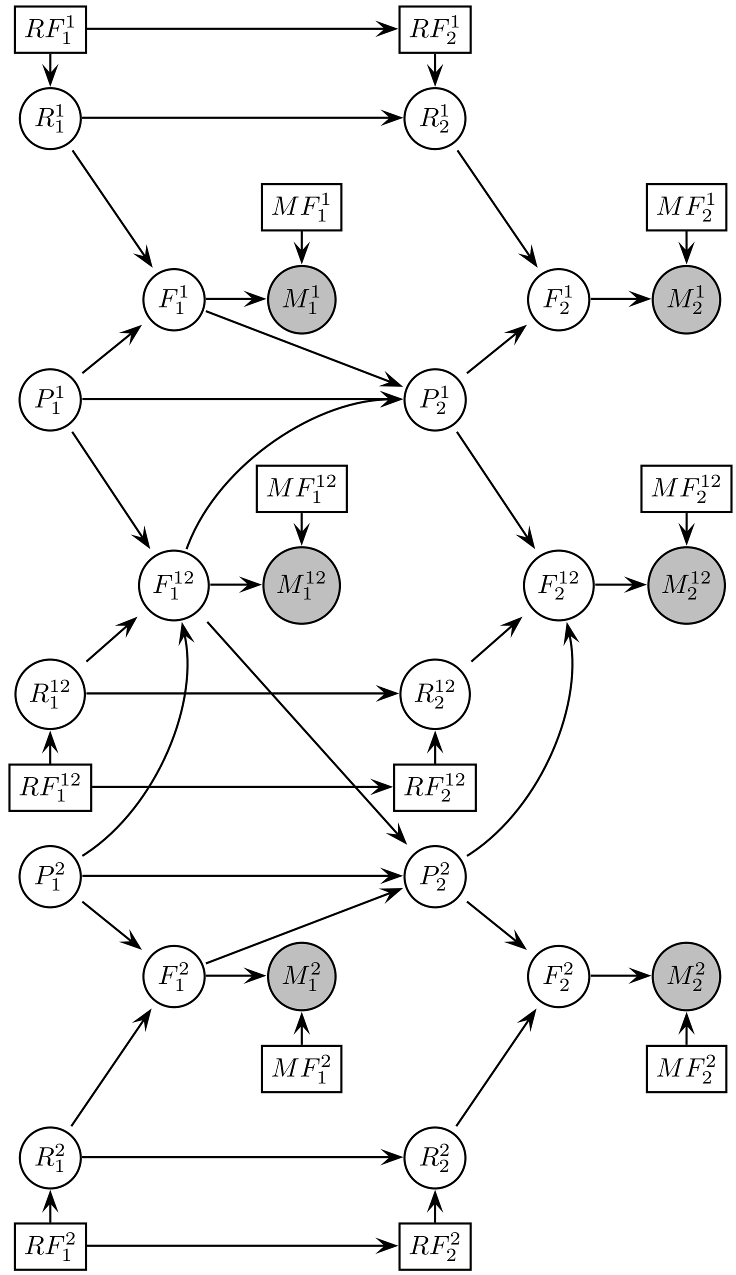


Figure 2.34: The two tanks system of Figure 2.33 modelled as a DBN. Discrete nodes are squares, continuous nodes are circles. Abbreviations: R = resistance, P = pressure, F = flow, M = measurement, RF = resistance failure, MF = measurement failure. Adapted from Figure 12 of [KL01].

2.4.6 Data association

We now discuss a special kind of switching KFM which we will use later in the thesis. Consider a factorial HMM where all of the chains have linear-Gaussian CPDs except one, call it S_t, which is discrete and is used to "select" which of the hidden chains to "pass through" to the output, i.e., the CPD for $P(Y_t|X_t^{1:D}, S_t)$ is a (Gaussian) multiplexer:

$$P(Y_t = y|S_t = i, X_t^{(1)}, \ldots, X_t^{(D)}) = \mathcal{N}(y; W_i X_t^{(i)} + \mu_i, \Sigma_i)$$

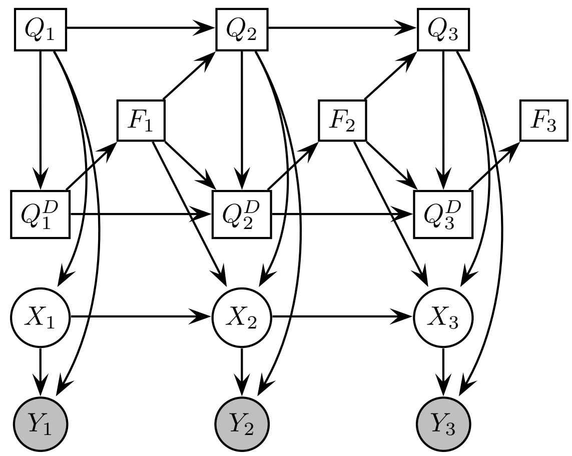


Figure 2.35: A switching KFM/ segment model hybrid. Square nodes are discrete, round nodes are continuous. We have omitted the segment counting control apparatus shown in Figure 2.27.

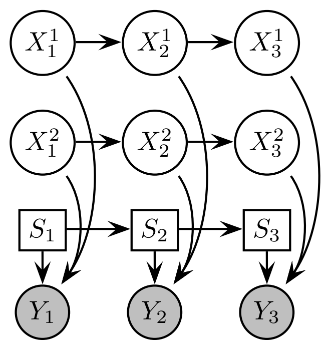


Figure 2.36: A DBN for the data association problem. The observation Y_t is from one of two independent KFMs, as determined by the multiplexer node S_t. Square nodes are discrete, round nodes are continuous.

This is shown in Figure 2.36. (This is what [GH98] call a "switching state-space model", although only the observations switch, not the dynamics.)

One important application for this model is for representing the data association (correspondence) problem [BSF88, BS90, BSL93]. In this case, one of the "objects" $X_t^1, \ldots, X_t^D$, is assumed to have generated the observation Y_t, but we just don't know which. S_t is a latent variable which specifies the identity of the source of the observation Y_t.

The data association problem is extremely common. For example, consider tracking a single target in clutter (e.g., one missile surrounded by decoys). If $S_t = 1$, we know the measurement was generated by the missile, so we update its state estimate using Y_t; if $S_t = 0$, we know the measurement was generated by the "background", so we ignore it. Of course, we do not know the value of S_t, so in principle we should consider both values; this is called multiple hypothesis tracking (MHT) [BSF88, CH96]. Unfortunately, the hypothesis tree at time t has 2^t leaves, corresponding to all possible assigments to $S_{1:t}$, so some pruning is necessary. Alternatively, we can use Monte Carlo techniques [PROR99].[13]

A simple alternative to MHT in the single target case is to see if an observation falls within the 95% confidence ellipse/ validation gate (given by Cov$[Y_t|X_{t-1}]$, which is computed using an (extended) Kalman filter); if it does we assume the observation was generated by the target, otherwise we assume it was generated by the background. If there is more than one observation inside the ellipse, we pick the nearest (most likely). Of course, since we using a 95% threshold, we expect this to fail once every 20 time steps; however, we can sometimes recover from incorrect associations using future observations.

[13] Note that, if all hidden variables are discrete, including the X^i's, exact inference (using constant time per update) is always possible; unfortunately, each step takes $O(K^D)$ time, where D is the number of objects and K is the number of states each object can be in (see Section 3.5)/ Data association and tracking using discrete-state DBNs is discussed in [MF92, NB94].

In the multi-target case, an observation might fall inside the validation gate (confidence ellipse) of several objects. In this case, we can either assign the observation to the nearest target (using Mahalanobis distance), or compute the likelihood of all possible joint assignments of observations to targets, and pick the most likely one [CH96]. Note that the nearest neighbor rule might assign the same measurement to multiple objects, which leads to inaccuracies.

2.4.7 Tracking a variable, unknown number of objects

When we are tracking multiple targets, the measurement could have been caused by any of them, the background, or perhaps a new target that has entered the "scene". (This problem also arises in mobile robotics, in particular in the SLAM (simultaneous localization and mapping) problem, which we will discuss in Section 5.3.2.) A standard way to detect the presence of new objects is if an observation arises that does not fall inside the validation gate (confidence ellipse) of any existing object; in this case, it could either be due to a new object, or due to background clutter, so we consider both hypotheses, and add the new object to a provisional list. Once the object on the provisional list receives a minimum number of measurements inside its validation gate, it is added to the state space. It is also possible for an object to be removed from the state space if it has not been updated recently (e.g., it left the area of interest). Hence in general we must allow the state space to grow and shrink dynamically. We discuss this further in Section 2.5.

2.5 First order DBNs

In Section 2.4.7, we discussed tracking an unknown, variable number of objects. In AI, such models are called "first order", as opposed to "propositional" models, which deal with a fixed set of attributes. See Section A.5 for a brief discussion of first-order probabilistic models, and [BRP01] for a discussion of first-order MDPs.

In a first order DBN, we must decide when to create a new object, and when to delete an old one (not necessarily because it ceased to exist (e.g., was blown up by a missile), but maybe just because we no longer want to reason about it, c.f., the frame problem in AI). In the tracking case, we used an heuristic (the provisional list) to decide when to add an object, triggered by "surprising" observations. In online model selection (see Section 6.1.1), we propose new basis functions at random, and check to see if they increased the model fit. (Obviously we could use smarter proposal distributions.)

In addition to creating a new object, we must decide how the new object relates to the existing ones. In the previous examples, we implicitly assumed the new object was unrelated to existing ones, i.e., had zero cross-covariance. However, in certain applications, the relationships between objects might be more important and/or easier to estimate than the object properties themselves [Gui02]. In general, we need to estimate both object properties and their relations, e.g., for image understanding [Gre93, BGP97], natural language understanding [GC92], etc.

Chapter 3

Exact inference in DBNs

3.1 Introduction

The goal of inference in a DBN is to compute the marginals $P(X_t^i|y_{1:\tau})$; if $\tau = t$ this is filtering, if $\tau > t$, this is smoothing, and if $\tau < t$ this is prediction. For learning, we must also be able to compute the family marginals $P(\text{Pa}(X_t^i), X_t^i|y_{1:\tau})$.

In this chapter, we show to perform these inferences using a pair of abstract forwards/ backwards operators c.f., [LBN96] and [RN02, ch.17]. We will then discuss increasingly efficient ways to implement these operators for general DBNs, culminating in a discussion of the lower bounds on complexity of exact inference in DBNs. For most of this chapter, we assume all hidden variables are discrete. We address inference in models with continuous (or mixed discrete-continuous) state-spaces in Section 3.6.

The main novel contributions are the island algorithm — a way of doing offline smoothing in $O(\log T)$ space instead of the usual $O(T)$ space — and the interface algorithm — a simple way to implement the forwards/ backwards operators using the junction tree algorithm.

Since inference in DBNs uses inference in BNs as a subroutine, you are strongly recommended to read Appendix B before reading this chapter.

3.2 The forwards-backwards algorithm

We will now review how to do (fixed-interval) smoothing in an HMM using the well-known forward-backwards (FB) algorithm. This algorithm can be applied to any discrete-state DBN, by converting the DBN to an HMM. We also present a number of variations on the standard algorithm, which will prove to be useful later in the thesis.

The basic idea of the FB algorithm is to recursively compute $\alpha_t(i) \stackrel{\text{def}}{=} P(X_t = i|y_{1:t})$ in the forwards pass[1], to recursively compute $\beta_t(i) \stackrel{\text{def}}{=} P(y_{t+1:T}|X_t = i)$ in the backwards pass, and then to combine them to produce the final answer $\gamma_t(i) \stackrel{\text{def}}{=} P(X_t = i|y_{1:T})$:

$$
\begin{aligned}
P(X_t = i|y_{1:T}) &= \frac{1}{P(y_{1:T})} P(y_{t+1:T}|X_t = i, y_{1:t}) P(X_t = i, y_{1:t}) \\
&= \frac{1}{P(y_{1:T})} P(y_{t+1:T}|X_t = i) P(X_t = i|y_{1:t})
\end{aligned}
$$

or

$$\gamma_t \propto \alpha_t . * \beta_t$$

where $.*$ denotes elementwise product, i.e., $\gamma_t(i) \propto \alpha_t(i)\beta_t(i)$.

[1] It is more common (see e.g., [Rab89]) to define $\alpha_t(i) = P(X_t = i, y_{1:t})$; the difference is discussed in Section 3.2.1.

3.2.1 The forwards pass

We compute α recursively as follows.

$$\alpha_t(j)P(X_t = j|y_{1:t}) = \frac{1}{c_t}P(X_t = j, y_t|y_{1:t-1})$$

where

$$P(X_t = j, y_t|y_{1:t-1}) = \left[\sum_i P(X_t = j|X_{t-1} = i)P(X_{t-1} = i|y_{1:t-1})\right] P(y_t|X_t = j) \tag{3.1}$$

and

$$c_t = P(y_t|y_{1:t-1}) = \sum_j P(X_t = j, y_t|y_{1:t-1}) \tag{3.2}$$

In vector-matrix notation, this becomes

$$\alpha_t \propto O_t A' \alpha_{t-1} \tag{3.3}$$

where A' denotes the transpose of A and $O_t(i,i) \stackrel{\text{def}}{=} P(y_t|X_t = i)$ is a diagonal matrix containing the conditional likelihood of the evidence at time t. (This is the only way in which the evidence affects the algorithm. Hence we can use any model for $P(Y_t|X_t = i)$.)

The base case is

$$\alpha_1(j) = P(X_1 = j|y_1) = \frac{1}{c_1}P(X_1 = j)P(Y_1|X_1 = j)$$

or

$$\alpha_1 \propto O_1 \pi$$

If we did not normalize at each step, then we would be computing $\alpha_t(i) = P(X_t = i, y_{1:t})$. However, this joint probability will become very small for large t, and hence this quantity will rapidly underflow. One solution is to work in the log-domain. An alternative is to normalize, i.e., to compute $\alpha_t(i) = P(X_t = i|y_{1:t})$, which always sums to one. Not only does this prevent underflow, but it is also a much more meaningful quantity (a filtered state estimate). We keep track of the normalizing constants so that we can compute the likelihood of the sequence:

$$P(y_{1:T}) = P(y_1)P(y_2|y_1)P(y_3|y_{1:2})\dots P(y_T|y_{1:T-1}) = \prod_{t=1}^{T} c_t$$

3.2.2 The backwards pass

We compute the β's recursively as follows. The base case is

$$\beta_T(i) = 1$$

since $\Pr(y_{T+1:T}|X_T = i) = \Pr(\emptyset|X_T = i) = 1$. The recursive step is

$$\begin{aligned} P(y_{t+1:T}|X_t = i) &= \sum_j P(y_{t+2:T}, X_{t+1} = j, y_{t+1}|X_t = i) \\ &= \sum_j P(y_{t+2:T}|X_{t+1} = j, y_{t+1}, X_t = i)P(X_{t+1} = j, y_{t+1}|X_t = i) \\ &= \sum_j P(y_{t+2:T}|X_{t+1} = j)P(y_{t+1}|X_{t+1} = j)P(X_{t+1} = j|X_t = i) \end{aligned}$$

or

$$\beta_t = A O_{t+1} \beta_{t+1}$$

Note that the backwards algorithm requires the evidence stream in the form of O_t, but can be run independently of the forwards algorithm. This is sometimes called the “two filter” approach to smoothing. However,

$\beta_t = P(y_{t+1:T}|X_t)$ is a conditional likelihood, not a filtered estimate; see Section 3.2.5 for a "proper" two-filter smoothing algorithm.

Since $\beta_t = P(y_{t+1:T}|X_t)$ is a conditional likelihood, it need not sum to one. To prevent underflow, we normalize at every step. The normalization constant is arbitrary, since it will cancel when we compute γ_t:

$$P(X_t|y_{1:T}) \propto P(X_t, y_{1:t})P(y_{t+1:T}|X_t) = (C_t\alpha_t)\beta_t \propto (C_t\alpha_t)(d_t\beta_t)$$

where $C_t = \prod_{i=1}^{t} c_t = P(y_{1:t})$ and d_t is any constant. Rabiner [Rab89] chooses $d_t = c_t$, but in BNT[2], I use $d_t = \sum_i \beta_t(i)$, which is computable without reference to α.

3.2.3 An alternative backwards pass

The fact that β_t is not a true probability distribution, besides being unintuitive, presents problems when we consider Kalman filter models.[3] Hence we now derive a way of computing γ_t recursively without first having to first compute β_t. I call this the α/γ algorithm instead of the α/β algorithm. This derivation turns out to be equivalent to the junction tree algorithm (see Section B.4) applied to this model [SHJ97].)

The α/γ algorithm exploits the fact that $X_{t-1} \perp y_{t+1:T}|X_t$ as follows.

$$\gamma_{t-1}(i) = \sum_j \xi_{t-1,t|T}(i,j)$$

where

$$\begin{aligned}
\xi_{t-1,t|T}(i,j) &\stackrel{\text{def}}{=} P(X_{t-1}=i, X_t=j|y_{1:T}) \\
&= P(X_{t-1}=i|X_t=j, y_{1:t})P(X_t=j|y_{1:T}) \\
&= P(X_{t-1}=i, X_t=j|y_{1:t})\frac{P(X_t=j|y_{1:T})}{P(X_t=j|y_{1:t})} \\
&= \xi_{t-1,t|t}(i,j)\frac{\gamma_t(j)}{\alpha_t(j)}
\end{aligned}$$

and

$$\begin{aligned}
\xi_{t-1,t|t}(i,j) &\stackrel{\text{def}}{=} P(X_{t-1}=i, X_t=j|y_{1:t}) \\
&\propto P(y_t|X_t=j)P(X_t=j|X_{t-1}=i)P(X_{t-1}=i|y_{1:t-1}) \\
&= O_t(j,j)A(i,j)\alpha_{t-1}(i)
\end{aligned}$$

We can think of the ratio $\frac{P(X_t=j|y_{1:T})}{P(X_t=j|y_{1:t})}$ as an update factor, or equivalently, as the backwards message, since

$$\begin{aligned}
\frac{P(X_t=j|y_{1:T})}{P(X_t=j|y_{1:t})} &= \frac{P(X_t=j, y_{1:T})}{P(X_t=j|y_{1:t})P(y_{1:T})} \\
&= \frac{P(X_t=j|y_{1:t})P(y_{t+1:T}|X_t=j)}{P(X_t=j|y_{1:t})P(y_{1:T})} \\
&\propto \beta_t(j).
\end{aligned}$$

This update factor is "absorbed" into the two-slice distribution $P(X_{t-1}, X_t|y_{1:t})$, which is then marginalized down to yield $P(X_{t-1}|y_{1:T})$. This is the standard way of performing inference in the junction tree algorithm (see Section B.4.4).

[2] Specifically, in the function `BNT/HMM/forwards-backwards.m`.

[3] In KFMs, α_t can be represented in moment form (i.e., in terms of $E[X_t|y_{1:t}]$ and Cov$[X_t|y_{1:t}]$), or in canonical (information) form (see Section B.5.1). However, β_t must be represented in canonical form, since it is not necessarily normalizable. Unfortunately, even in this form, the required inverses may not exist. Hence in the backwards (smoothing) pass, it is better to compute γ_t directly, which can always be represented in moment form, since it is a probability distribution. This is also essential for switching KFMs, since the standard moment-matching (weak marginalisation) approximation is only applicable if the message is in moment form (see Section B.5.2).

3.2.4 Two-slice distributions

When we do parameter estimation (see Section 1.2.3), we will also need to compute the two-slice distribution $\xi_{t-1,t|T}(i,j) \stackrel{\text{def}}{=} P(X_{t-1} = i, X_t = j|y_{1:T})$. If we use the α/γ (junction tree) algorithm, the two-slice distribution has already been computed. If we use the α/β algorithm, this can be computed using

$$P(X_{t-1} = i, X_t = j|y_{1:T}) \propto P(X_t = j|X_{t-1} = i)P(X_{t-1} = i|y_{1:t-1})P(y_t|X_t = j)P(y_{t+1:T}|X_t = j)$$

or

$$\xi_{t-1,t|T} \propto A. * \alpha_{t-1} O_t \beta_t'$$

where the normalizing constant ensures $\sum_{i,j} \xi_{t-1,t|T}(i,j) = 1$.

3.2.5 A two-filter approach to smoothing

We have already remarked that $\beta_t = P(y_{t+1:T}|X_t)$ is a conditional likelihood; it would seem more natural to use the backwards filtered estimate $\hat{\beta}_t \stackrel{\text{def}}{=} P(X_t|y_{t+1:T})$ (we start at $t+1$ so we don't double-count y_t when we combine with α_t). We now derive a novel recursive backwards update for $\hat{\beta}_t$. (An analogous result for KFMs can be found in [FP69].)

First we define the reverse transition matrix as follows:

$$A_r(j,i) \stackrel{\text{def}}{=} P(X_t = i|X_{t+1} = j) = \frac{P(X_{t+1} = j|X_t = i)P(X_t = i)}{P(X_{t+1} = j)}$$

Let $\Pi_t(i,i) \stackrel{\text{def}}{=} P(X_t = i)$ be a diagonal matrix. Then we can write

$$A_r(j,i) = \frac{A(i,j)\Pi_t(i,i)}{\Pi_t(j,j)}$$

This is only defined if $\Pi_t(j,j) > 0$ for all j and t. (This is not true for a left-to-right transition matrix, for example.)

Assuming A_r exists, we can derive the backwards filter as follows.

$$\begin{aligned}
P(X_t = i|y_{t+1:T}) &\propto P(X_t = i, y_{t+1}|y_{t+2:T}) \\
&= \sum_j P(X_t = i|y_{t+1}, X_{t+1} = j, y_{t+2:T})P(y_{t+1}, X_{t+1} = j|y_{t+2:T}) \\
&= \sum_j P(X_t = i|X_{t+1} = j)P(y_{t+1}|X_{t+1} = j)P(X_{t+1} = j|y_{t+2:T})
\end{aligned}$$

or, more concisely,

$$\hat{\beta}_t \propto A_r' O_{t+1} \hat{\beta}_{t+1}$$

where the constant of proportionality is $1/P(y_{t+1}|y_{t+2:T})$. This has an appealing symmetry with the forwards algorithm, but is only applicable if A_r exists.

Finally, can combine the two filters as follows.

$$\begin{aligned}
P(X_t = i|y_{1:T}) &\propto P(X_t = i|y_{1:t})P(y_{t+1:T}|X_t = i) \\
&= P(X_t = i|y_{1:t})\frac{P(y_{t+1:T}, X_t = i)}{P(X_t = i)} \\
&\propto P(X_t = i|y_{1:t})\frac{P(X_t = i|y_{t+1:T})}{P(X_t = i)}
\end{aligned}$$

or

$$\gamma_t(i) \propto \frac{\alpha_t(i)\hat{\beta}_t(i)}{\Pi_t(i,i)}$$

3.2.6 Time and space complexity of forwards-backwards

If X can be in K possible states, filtering takes $O(K^2)$ operations per time step, since we must do a matrix-vector multiply at every step. Smoothing therefore takes $O(K^2T)$ time. (If A is sparse, and each state has at most F_{in} predecessors, then the complexity is $(KF_{in}T)$, e.g., for a left-to-right HMM, $F_{in} = 2$ (predecessor state and self-arc), so the complexity is $O(KT)$.) Smoothing also needs $O(KT)$ space, since we must store α_t for $t = 1, \dots, T$ until we do the backwards pass. For complex models, in which K can be very large, and long sequences, in which T is large, we often find that we run out of space. Section 3.7.1 discusses a way to reduce the space complexity to $O(K \log T)$, while still performing exact inference (at the expense of a slowdown by a $O(\log T)$ factor).

3.2.7 Abstract forwards and backwards operators

We have now seen several different ways of performing filtering and smoothing in HMMs, depending on whether we use α/β, α/γ, etc. To hide these details from higher-level inference algorithms, we define abstract forwards and backwards operators.

The forward operator is denoted by

$$(f_{t|t}, L_t) = \text{Fwd}(f_{t-1|t-1}, y_t)$$

where $f_{t|t}$ is the forwards "message"[4], and $L_t = \log P(y_t|y_{1:t-1})$ is the conditional log-likelihood of the observation; this is a measure of surprise (the "innovation" probability). The backward operator is denoted by

$$b_{t|T} = \text{Back}(b_{t+1|T}, f_{t|t})$$

where $b_{t|T}$ is the backwards "message". The base cases will be denoted by $(f_{1|1}, L_1) = \text{Fwd1}(y_1)$ and $b_{T|T} = \text{backT}(f_{T|T})$.

The "meaning" of the forwards and backwards messages depends on which algorithm we use. In general all we can say is that $b_{t|T}$ is some "object" (in the programming language sense) from which we can compute $P(Z_t^i|y_{1:T})$ and $P(Z_t^i, \text{Pa}(Z_t^i)|y_{1:T})$ for any node Z_t^i and its parents.

3.3 The frontier algorithm

The forwards-backwards algorithm for HMMs works by exploiting the fact that X_t d-separates the past from the future. For a DBN, the set of all hidden nodes, $X_t^{(1:D)}$, d-separates the past from the future. The frontier algorithm [Zwe96] is a way of updating the joint distribution on this set of nodes without needing to create, yet alone multiply by, an $O(K^D \times K^D)$ transition matrix. The basic idea is to "sweep" a Markov blanket across the DBN, first forwards and then backwards. We shall call the nodes in the Markov blanket the "frontier set", and denote it by F; the nodes to the left and right of the frontier will be denoted by L and R. At every step of the algorithm, we ensure F d-separates L and R. The forwards pass corresponds to the computation of α, and the backwards pass to the computation of β. We give the details below.[5]

3.3.1 Forwards pass

We use the original notation from [Zwe96], so h_F refers to the hidden nodes in F, e_F refers to the evidence in F, e_L refers to the evidence in L, and e_R refers to the evidence in R. In the forwards pass, $P(F) \stackrel{\text{def}}{=} P(h_F, e_F, e_L)$ (in practice, we normalize to get $P(h_F|e_F, e_L)$). We can compute this recursively as

[4] This notation is based on the convention from Kalman filtering, where $E[X_t|y_{1:\tau}]$ is denoted by $\hat{x}_\tau$.

[5] A special case of the frontier algorithm, applied to factorial HMMs, was published in Appendix B of [GJ97]. Since there are no cross links between the hidden nodes in an FHMM, there are no constraints on the order in which nodes are added to or removed from the frontier, so the resulting algorithm is much simpler than the general case presented here. The frontier algorithm is itself a special case of the junction tree algorithm. We describe it here because it is simple to understand, and because it is the inference algorithm used in GMTK [BZ02].

follows. We can add a node N to the frontier (move it from R to F) when all its parents are already in the frontier:

$$P(e_L, e_F, h_F, N) \quad = \quad P(e_L, e_F, h_F)P(N|e_F, h_F)$$

since $N \perp e_L | e_F, h_F$. $P(e_L, e_F, h_F)$ is available by inductive assumption. In other words, adding a node consists of multiplying its CPD onto the frontier.

We can remove a node N (move it from F to L) when all its children are in the frontier. This can be done as follows. Let $L + N$ represent $L \cup \{N\}$, and $F - N$ represent $F \setminus \{N\}$. Consider first the case that N is hidden, so $e_{L+N} = e_L$ and $e_{F-N} = e_F$.

$$\begin{aligned}
P(e_{L+N}, e_{F-N}, h_{F-N}) &= P(e_L, e_F, h_{F-N}) \\
&= \sum_N P(e_L, e_F, N, h_{F-N}) \\
&= \sum_N P(e_L, e_F, h_F)
\end{aligned}$$

where the last line follows since $h_{F-N} \cup \{N\} = h_F$. In other words, removing a node simply means marginalizing it out. (If we replace sum with max, we get what [Zwe98] calls the "chain decoding" algorithm.) The case where N is observed is similar:

$$\begin{aligned}
P(e_{L+N}, e_{F-N}, h_{F-N}) &= P(e_{L+N}, e_{F-N}, h_F) \\
&= P(e_L, e_N, e_{F-N}, h_F) \\
&= P(e_L, e_F, h_F)
\end{aligned}$$

We can skip the marginalization since N is observed; hence this is essentially a no-op. Note that this procedure is equivalent to the variable elimination algorithm (see Section B.2) with a specific ordering.

3.3.2 Backwards pass

In the backwards pass, $P(F) \stackrel{\text{def}}{=} P(e_R | h_F, e_F)$. We can advance the frontier from slice $t + 1$ to slice t by adding and removing nodes in the opposite order that we used in the forwards pass, e.g., suppose in the forward pass we add X_t^1, remove X_{t-1}^1, add X_t^2, remove X_{t-1}^2; then in the backwards pass we would add X_{t-1}^2, remove X_t^2, add X_{t-1}^1, remove X_t^1. Adding a node means moving it from L to F; removing a node means moving it from F to R; hence R is the set of nodes that have been "processed", the opposite of the forwards pass.

When we add node N to F, we want to compute $P(e_R | e_F, h_F, N)$. Because N was removed at this step in the forwards pass, we know that all N's children are in F, which "shield" N from e_R; hence $P(e_R | e_F, h_F, N) = P(e_R | e_F, h_F)$. So adding N simply means expanding the domain of the frontier to contain it, by duplicating all the existing entries, once for each possible value of N.

When we remove node N from F (and add it to R), we want to compute $P(e_{R+N} | e_{F-N}, h_{F-N})$ from $P(e_R | e_F, h_F)$. Consider first the case that N is a hidden node, so $e_{R+N} = e_R$, and $e_{F-N} = e_F$. Then we have

$$\begin{aligned}
P(e_{R+N} | e_{F-N}, h_{F-N}) &= P(e_R | e_F, h_{F-N}) \\
&= \sum_N P(N, e_R | e_F, h_{F-N}) \\
&= \sum_N P(N | e_F, h_{F-N}) P(e_R | N, e_F, h_{F-N}) \\
&= \sum_N P(N | e_F, h_{F-N}) P(e_R | e_F, h_F)
\end{aligned}$$

since $h_{F-N} \cup \{N\} = h_F$. In other words, to remove node N, we multiply in N's CPD (this is possible since all of N's parents will be in F, since N was added at this step in the forwards pass) and then marginalize out N.

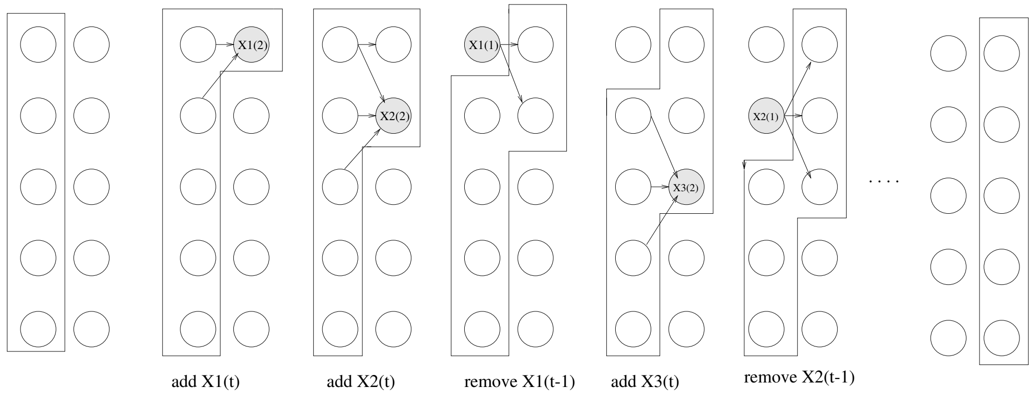


Figure 3.1: The frontier algorithm applied to a coupled HMM with 5 chains (see Figure 2.11); observed leaves (i.e., $y_{1:T}$) are omitted for clarity. Nodes inside the box are in the frontier. The node being operated on is shown shaded; only connections with its parents and children are shown; other arcs are omitted for clarity. See text for details.

If N is observed, the procedure is basically the same, except we don't need to marginalize out N, since it only has one possible value.

$$\begin{aligned}
P(e_{R+N}|e_{F-N}, h_{F-N}) &= P(e_{R+N}|e_{F-N}, h_F) \\
&= P(e_N, e_R|e_{F-N}, h_F) \\
&= P(e_N|e_{F-N}, h_F)P(e_R|e_N, e_{F-N}, h_F)
\end{aligned}$$

3.3.3 Example

We now give an example of the forwards pass applied to a coupled HMM: refer to Figure 3.1. The frontier initially contains all the nodes in slice $t-1$: $F_{t,0} \stackrel{\text{def}}{=} \alpha_{t-1} = P(X_{t-1}^{1:D}|y_{1:t-1})$. We then advance the frontier by moving X_t^1 from R to F. To do this, we multiply in its CPD $P(X_t^1|X_{t-1}^1, X_{t-1}^2)$:

$$F_{t,1} = P(X_t^1, X_{t-1}^{1:D}|y_{1:t-1}) = P(X_t^1|X_{t-1}^1, X_{t-1}^2) \times F_{t,0}$$

Next we add in X_t^2:

$$\begin{aligned}
F_{t,2} &= P(X_t^{1:2}, X_{t-1}^{1:D}|y_{1:t-1}) \\
&= P(X_t^2|X_{t-1}^1, X_{t-1}^2, X_{t-1}^3) \times F_{t,1}
\end{aligned}$$

Now all of the nodes that depend on X_{t-1}^1 are in the frontier, so we can marginalize X_{t-1}^1 out (move it from $\mathcal{F}$ to $\mathcal{L}$):

$$F_{t,3} = P(X_t^{1:2}, X_{t-1}^{2:D}|y_{1:t-1}) = \sum_{X_{t-1}^1} F_{t,2}$$

The process continues in this way until we compute

$$F_{t,D} = P(X_t^{1:D}|y_{1:t-1})$$

Finally, we weight this factor by the likelihood to get

$$F_{t+1,0} = \alpha_t = P(X_t^{1:D}|y_{1:t}) \propto P(y_t|X_t^{1:D}) \times F_{t,D} = \prod_{i=1}^{D} P(y_t^i|X_t^i) \times F_{t,D}$$

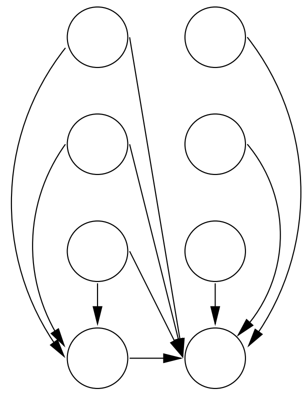


Figure 3.2: A worst case DBN from a complexity point of view, since we must create a clique which contains all nodes in both slices.

3.3.4 Complexity of the frontier algorithm

It is clear that in the above example, exact smoothing takes $O(TDK^{D+2})$ time and space, since the frontier never contains more than $D+2$ nodes, and it takes $O(D)$ steps to sweep the frontier from $t-1$ to t. In general, the running time of the frontier algorithm is exponential in the size of the largest frontier. We would therefore like to keep the frontiers as small as possible. Unfortunately, computing an order in which to add and remove nodes so as to minimize the sum of the frontier sizes is equivalent to finding an optimal elimination ordering, which is known to be NP-hard [Arn85]. Nevertheless, heuristics methods, such as greedy search [Kja90], often perform as well as exhaustive search using branch and bound [Zwe96]. (See Section B.3.5 for more on elimination orderings.)

In the best case, which occurs when all the chains are disconnected, as in a factorial HMM, the frontier algorithm does $O(D)$ multiplications by $O(K \times K)$ matrices to advance the frontier from slice $t-1$ to slice t (and similarly in the backwards direction). Each of these $K \times K$ matrices must be multiplied onto the joint, which has size $O(K^D)$. Hence the total amount of work, in the best case, is $O(DK^{D+1})$ per time step; for a regular HMM ($D = 1$), this becomes $O(K^2)$, as expected. In the worst case, which occurs for a DBN like the one in Figure 3.2, the total amount of work is $O(K^{2D})$, as in the FB algorithm. To see this, note that we must add all the nodes in slice t, while keeping all the nodes in slice $t-1$ in the frontier, before we can remove any nodes; this is because the bottom node depends on all nodes in slice t and all nodes in slice $t-1$, i.e., the corresponding moral graph is a clique containing both slices. Adding the nodes in slice t one by one to the frontier takes $\sum_{i=1}^{D} K^{D+i} = K^D \frac{K^{D+1}-1}{K-1} = O(K^{2D})$ time, since the frontier grows from size K^D to K^{2D}; marginalizing them out also takes $O(K^{2D})$ time. In general, the algorithm will always take at least $O(TK^{D+F_{in}+1})$, where F_{in} is the maximal number of parents of any node within the same slice.

3.4 The interface algorithm

The frontier algorithm uses all the hidden nodes in a slice to d-separate the past from the future. This set is larger than it needs to be, and hence the algorithm is sub-optimal (see Table 3.1). I claim that the set of nodes with outgoing arcs to the next time-slice is sufficient to d-separate the past from the future; following [Dar01], I will call this set the "forward interface". For example, the forward interface for Figure 3.4 is $\{X_t^1, X_t^4\}$. I now state and prove this result formally.

Definition. Let the set of temporal arcs between slices $t-1$ and t be denoted by $E^{\text{tmp}}(t) = \{(u, v) \in E | u \in V_{t-1}, v \in V_t\}$, where V_t are the nodes in slice t. The forward interface is defined as $I_t \stackrel{\text{def}}{=} \{u \in V_t | (u, v) \in E^{\text{tmp}}(t+1), v \in V_{t+1}\}$, i.e., the nodes which have children in the next slice. The set of non-interface nodes is $N_t = V_t \setminus I_t$.

Lemma. $\{V_{1:t-1}, N_t\} \perp V_{t+1:T} | I_t$, i.e., the forward interface d-separates the past from the future, where the past is N_t and earlier nodes, and the future is V_{t+1} and later nodes.

Proof. Let I be a node in the interface, connected to a node P in the past and a node F in the future (which

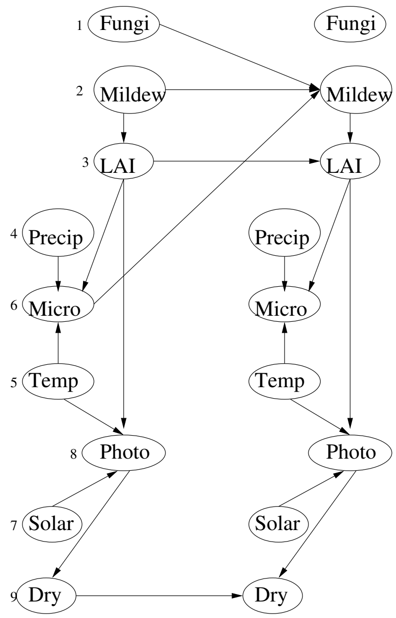


Figure 3.3: The Mildew DBN, designed for foreacasting the gross yield of wheat based on climatic data, observations of leaf area index (LAI) and extension of mildew, and knowledge of amount of fungicides used and time of usage [Kja95]. Nodes are numbered in topological order, as required by BNT.

DBN	Figure	Slice-size	Frontier	Back	Fwd
BAT	4.2	28	18	9	10
Water	4.1	12	8	8	8
Mildew	3.3	9	9	4	5
Cascade	3.5	4	4	4	1
Uffe	3.4	4	4	3	2

Table 3.1: Sizes of various separating sets for different DBNs. The slice-size is the total number of nodes per slice. The frontier is the total number of hidden nodes per slice. Back is the size of the backwards interface, i.e., the number of nodes with incoming temporal arcs, plus parents of such nodes in the same slice (see text for precise definition). Fwd is the size of the forward interface, i.e., the number of nodes with children in the next slice.

must be a child of I, by definition). If P is a parent, the graph looks like this: $P \to I \to F$. If P is a child, the graph looks like this: $P \leftarrow I \to F$. Either way, we have $P \perp F|I$, since I is never at the bottom of a v-structure. Since all paths between any node in the past and any node in the future are blocked by some node in the interface, the result follows. ■

Kjaerulff [Kja95] defines a related quantity, which he called the interface, but which I will call the backward interface, to avoid confusion with the forward interface. He defines the backward interface to be all nodes v s.t. v, or one of its children, has a parent in slice $t-1$, i.e., $\text{int}(t) = \{v \in V_t | (u,v) \in E^{\text{tmp}}(t) \text{ or } \exists w \in \text{ch}(v) : (u,w) \in E^{\text{tmp}}(t), u \in V_{t-1}\}$. The reason for this definition is the following: when we eliminate a node from slice $t-1$, we will create a term which involves all nodes which depend on it, including those in slice t; some of the slice t terms will involve parents in slice t as well. For example,

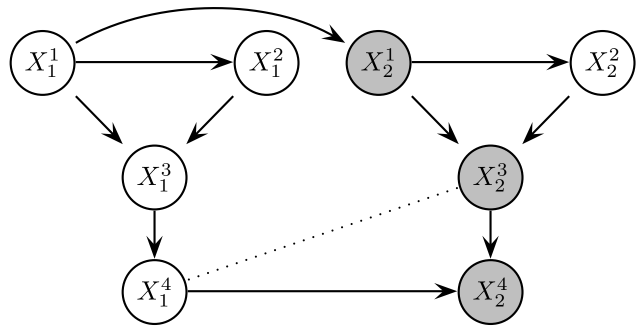


Figure 3.4: The "Uffe" DBN, from Figure 3 of [Kja95]. Nodes are numbered topologically, as required by BNT. The dotted undirected arcs are moralization arcs. The backward interface is shaded.

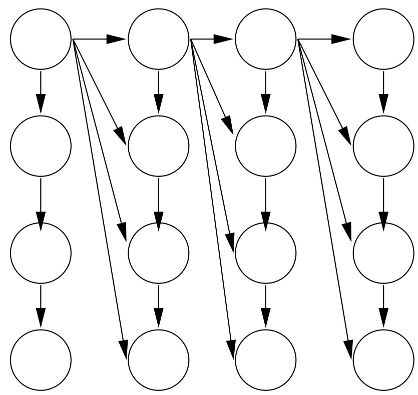


Figure 3.5: The "cascade" DBN, from Figure 2 of [Dar01]. This graph has a treewidth of 2.

consider eliminating the nodes from slice 1 of Figure 3.4. We have

$$\underbrace{\sum_{X_1^3} P(X_1^4|X_1^3) \underbrace{\sum_{X_1^2} \sum_{X_1^1} P(X_1^1)P(X_1^2|X_1^1)P(X_1^3|X_1^1, X_1^2)P(X_2^1|X_1^1)}_{\lambda(X_1^3, X_2^1)}}_{\lambda(X_2^1, X_1^4)}$$

and

$$\underbrace{\sum_{X_1^4} P(X_2^4|X_2^3, X_1^4)\lambda(X_2^1, X_1^4)}_{\lambda(X_2^1, X_2^4, X_2^3)}$$

Clearly we have coupled all the nodes in the backward interface, since the backward interface for Figure 3.4 is $\{X_t^1, X_t^3, X_t^4\}$.

The forward interface can sometimes be dramatically smaller than the backward interface (see Table 3.1). For an extreme example, consider Figure 3.5: The forward interface is $\{1\}$ but the backward interface is all the nodes in the slice. It is easy to see that the size of the forward interface is never larger than the size of the backward interface if all temporal arcs are persistence arcs, i.e., edges of the form X_{t-1}^i to X_t^i.

The other problem with the backward interface is that using it does not lead to a simple online inference algorithm; the one in [Kja95] involves a complicated procedure for dynamically modifying jtrees. Below I present a much simpler algorithm, which always uses the same jtree structure, constructed from a modified two-slice temporal Bayes net (2TBN) using an unconstrained elimination ordering, but with the restriction that the nodes in the forward interface must belong to one clique.

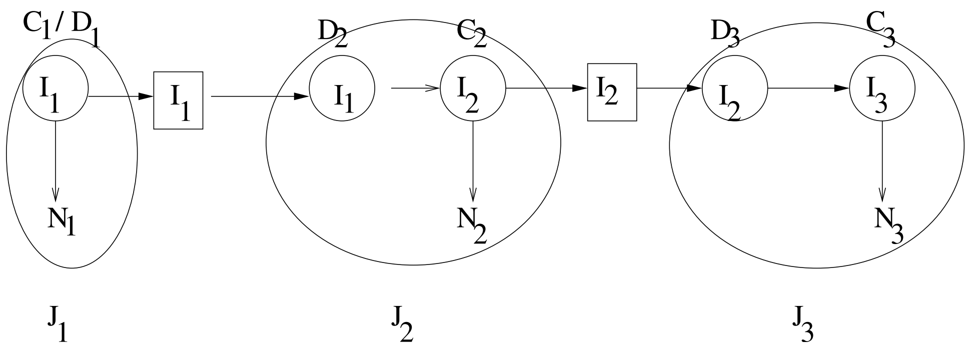


Figure 3.6: A schematic illustration of how to join the junction trees for each $1\frac{1}{2}$-slice DBN. I_t are the interface nodes for slice t, N_t are the non-interface nodes. D_t is the clique in J_t containing I_{t-1}. C_t is the clique in J_t containing I_t. The square box represents a separator, whose domain is I_t.

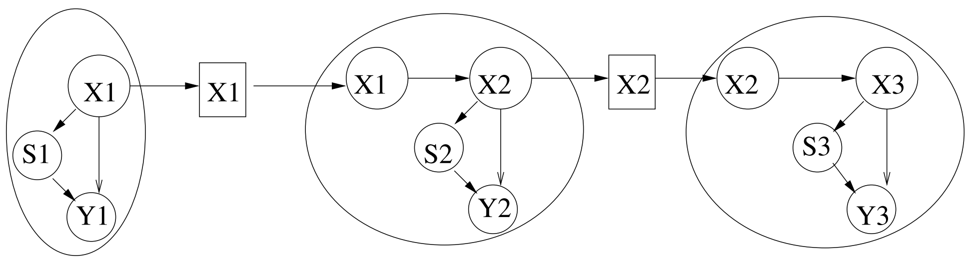


Figure 3.7: The DAGs for an HMM with mixture of Gaussians output, glued together by their interface nodes, X_t. The non-interface is $N_t = (S_t, Y_t)$.

3.4.1 Constructing the junction tree

Let G_t be the DAG created from slices $t-1$ and t from the unrolled DBN. A "$1\frac{1}{2}$-slice DBN" H_t (H for half) is the DAG created by eliminating all non-interface nodes (and their arcs) from the *first* slice of G_t, i.e., $H_t = I_{t-1} \cup V_t$. (For $t = 1$, $H_1 = V_1$.)

We now construct a jtree, J_t, for each H_t. (See Section B.3 for details on how to construct a jtree.) We must enforce the constraint that I_{t-1} and I_t each form a clique, since we need $P(I_{t-1})$ and $P(I_t)$. This can be ensured by simply adding edges to the moral graph between all nodes in I_{t-1}, and similarly for I_t, before construcing J_t.

We can now "glue" all the junction trees together via their interfaces, as shown in Figures 3.6 and 3.7. We can perform inference in each tree separately, and then pass messages between them via the interface nodes, first forwards and then backwards. We give the details below.

3.4.2 Forwards pass

In the forwards pass, we are given a prior belief state $P(I_{t-1}|y_{1:t-1})$, which is passed from C_{t-1} to D_t, where C_{t-1} is the clique in J_{t-1} containing I_{t-1} and D_t is the clique in J_t containing I_{t-1}. We then call collect-evidence on J_t with C_t as the root, which has the effect of doing one step of Bayesian updating. Finally we marginalize down the distribution over C_t onto I_t to compute $P(I_t|y_{1:t})$, and pass this to the next slice. In more detail, the steps are as follows.

1. Construct J_t, where all clique and separator potentials are initialized to the identity element (1's in the case of discrete potentials).

2. From J_{t-1}, extract the potential on C_{t-1}, and marginalize it down to I_{t-1}; this represents the prior, $P(I_{t-1}|y_{1:t-1})$. Multiply the prior onto the potential for D_t.

3. Multiply the CPDs for each node in slice t onto the appropriate potential in J_t, using y_t where necessary.

4. Collect evidence to the root C_t.

5. Return all clique and separator potentials in J_t.

We will denote this operator abstractly as

$$(f_{t|t}, L_t) = \text{Fwd}(f_{t-1|t-1}, y_t)$$

where $f_{t|t}$ contains the clique and separator potentials in J_t. As with HMMs, to prevent underflow, we must normalize all the messages (or else work in the log domain). The normalization constant at the root, C_t, will be $c_t = P(y_t|y_{1:t})$, which can be used to compute the log-likelihood, L_t.

For the first slice, we skip the step involving the prior (step 2). This will be denoted by

$$(f_{1|1}, L_1) = \text{Fwd1}(y_1)$$

After collecting to C_t, not all nodes (cliques) in J_t will have "seen" all the evidence $y_{1:t}$. For example, in Figure 3.7, if we collect to X_2, then the distribution on S_2 will be $P(S_2|y_2)$ rather than the full filtered distribution, $P(S_2|y_{1:2})$, since S_2 will only have received a message from Y_2 below, and not from X_2 above. Hence we must additionally perform a distribute-evidence operation from C_t to compute the distribution over all nodes in J_t; this will be performed in the backwards pass. Hence even when filtering, we must perform a backwards pass, but only within a single slice.

3.4.3 Backwards pass

In the backwards pass, we distribute evidence from C_t, and then pass a message from D_t to C_{t-1}. The details are as follows.

1. The input is $f_{t|t}$, which contains the clique and separator potentials over all nodes in J_t, and $b_{t+1|T}$, which contains the clique and separator potentials over all nodes in J_{t+1}.

2. From $b_{t+1|T}$, extract the potential on D_{t+1}, and marginalize it down to I_t; this represents $P(I_t|y_{1:T})$.

3. Update the potential on C_t in J_t by absorbing from the potential on D_{t+1} in J_{t+1} as follows:

$$\phi_C^* = \phi_C \times \frac{\sum_{D\setminus C} \phi_D}{\sum_{C\setminus D} \phi_C}$$

where $C = C_t$ and $D = D_{t+1}$.[6]

4. Distribute evidence from the root C_t.

5. Return all clique (and optionally separator) potentials.

We will denote this operator by

$$b_{t|T} = \text{Back}(b_{t+1|T}, f_{t|t})$$

To start the process at the final slice, we just distribute evidence from C_T (no need to absorb from J_{T+1}, which does not exist).

$$b_{T|T} = \text{BackT}(f_{T|T})$$

[6] In the forwards pass, we implicitely computed $\phi_D^* = \phi_D \times \frac{\sum_{C\setminus D}\phi_C}{\sum_{D\setminus C}\phi_D}$ where $C = C_{t-1}$ and $D = D_t$. However, since D_t is initially all 1s, we simplified this to $\phi_D = \sum_{C\setminus D}\phi_C = P(I_{t-1}|y_{1:t-1})$. In otherwords, we simply set the potential on D_t to be $P(I_{t-1}|y_{1:t-1})$, with a suitably extended domain.

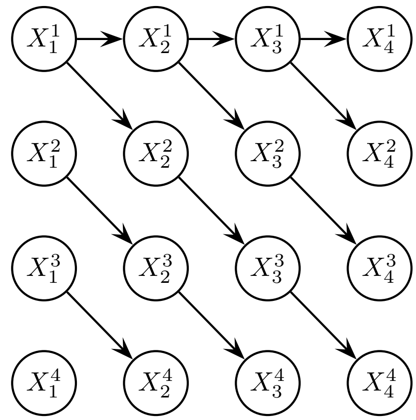


Figure 3.8: A DBN in which all the nodes in the last slice become connected when we eliminate the first 3 or more slices, even though the max clique size is 2.

3.4.4 Complexity of the interface algorithm

The complexity of the interface algorithm can be characterized as follows.

Theorem. The complexity of the interface algorithm is between $\Omega(K^{I+1})$ and $O(K^{I+D})$, where I is the size of the forwards interface, D is the number of hidden nodes per slice, and K is the maximum number of values each discrete hidden node can take.

Proof.[7] When we create the $1\frac{1}{2}$ slice jtree, the nodes are $I_{t-1} \cup V_t$. Let w be the size of the maximum clique created by eliminating this set according to some ordering π. Clearly $I + 1 \leq w$, all the nodes in I_{t-1} have a target in V_t, and $w \leq I + D$, since each node in I_{t-1} can be connected to at most D nodes in V_t. We now show these bounds are tight. The lower bound can be achieved by a Markov chain, where $I = 1$, so $w = 2$ (the cliques correspond to (X_{t-1}, X_t). The upper bound can be achieved by the DBN shown in Figure 3.2. ■

3.5 Computational complexity of exact inference in DBNs

We proved above that the interface algorithm always takes at least $\Omega(K^{I+1})$ time, where I is the size of the forwards interface. But this is a lower bound on the algorithm, not on the problem itself. To get a lower bound on the problem, we need to distinguish offline and online inference.

3.5.1 Offline inference

The simplest way to do exact inference in a DBN is to "unroll" the DBN for T slices and then apply any inference algorithm to the resulting static Bayes net. As discussed in Appendix B, the cost of this is determined by the tree width:

Theorem. Consider a 2TBN G with D nodes per slice, each of which has a maximum of K possible discrete values. Let $G_T = U_T(G)$ be this DBN unrolled for T slices. Then the complexity of any offline inference algorithm is at least $\Omega(K^w)$, where w is the treewidth, i.e., the max clique size of triangulate(moralize(G_T)) using an optimal elimination ordering.

In general, $w \geq I$, the size of the forwards interface, but this is not always the case. For example, in Figure 3.8, $I = 4$ but $w = 2$, since the graph is a tree.

[7] This proof is based on similar ones in [Dar01].

3.5.2 Constrained elimination orderings

When doing inference with sequences that can have variable lengths, it becomes too expensive to repeatedly unroll the DBN and convert it to a jtree. The approach taken in [Zwe98] is to unroll the DBN once, to some maximum length $T_{\max}$, construct a corresponding jtree, and then to "splice out" redundant cliques from the jtree when doing inference on a shorter sequence.

To ensure there is some repetitive structure to the junction tree which can be spliced out, we must use a constrained elimination ordering, in which we eliminate all nodes from slice t before any from slice $t+1$. The temporal constraint ensures that we create "vertical" cliques, which only contain nodes from neighboring time-slices, instead of "horizontal" cliques, which can span many time-slices. The resulting jtree is said to be constrainedly triangulated.

For example, consider the DBN in Figure 3.3. Using the constrained min-fill heuristic, as implemented in BNT, resulted in the following elimination ordering: $7, 1, 2, 4, 5, 6, 7, 3, 8$. The corresponding jtree, for 4 slices, is shown in Figure 3.9. The cliques themselves are shown in Figure 3.11. Notice how the jtree essentially has a head, a repeating body, and then a tail; the head and tail are due to the boundaries on the left and right. This is shown schematically in Figure 3.10. Zweig [Zwe98] shows how to identify the repeating structure (by looking for "isomorphic" cliques); this can then be "spliced out", to create a jtree suitable for offline inference on shorter sequences. By contrast, Figure 3.12 shows the cliques which result from an unconstrained elimination ordering, for which there is no repeating pattern.

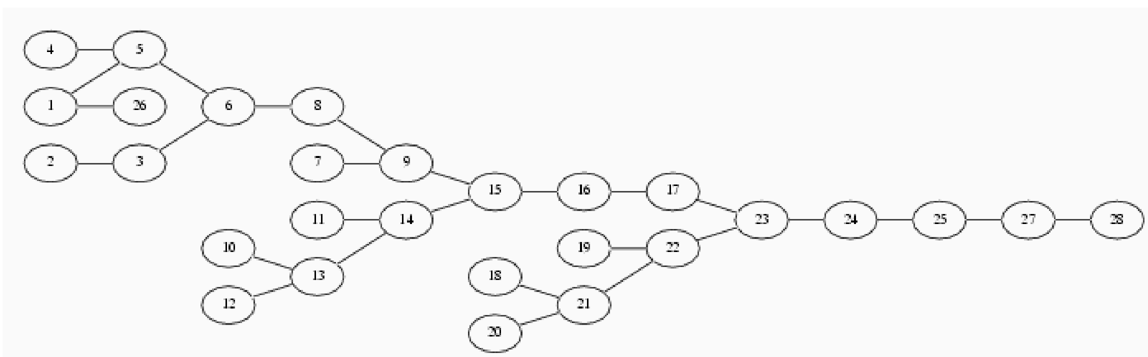


Figure 3.9: The jtree for 4 slices of the Mildew DBN (Figure 3.3) created using a constrained elimination ordering. Notice how the backward interface (cliques 9, 17 and 25) are separators. Node 28 in slice 4 is not connected to the rest of the DBN. The corresponding clique (26) is arbitrarily connected to clique 1 (nodes 3,5,7,8 in slice 1) to make the jtree a tree instead of a forest.

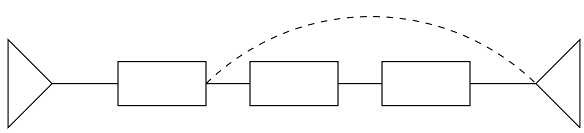


Figure 3.10: A schematic illustration of a generic jtree for a DBN. The diamonds represent the head and tail cliques, the boxes represent the 'body' cliques, that get repeated. The dotted arc illustrates how some slices can be 'spliced out' to create a shorter jtree. Based on Figure 3.10 of [Zwe98].

3.5.3 Consequences of using constrained elimination orderings

Using a constrained elimination ordering for offline inference is suboptimal. For example, Figure 3.8 shows a DBN where the treewidth is 2 (since the graph is essentially a tree), but a constrained elimination ordering creates cliques of size $D = 4$ for slices after $t \geq 4$.[8] This is a consequence of the following theorem.

[8] You can verify this and other facts experimentally using the following BNT script: `BNT/examples/dynamic/jtree-clq-test.m`.

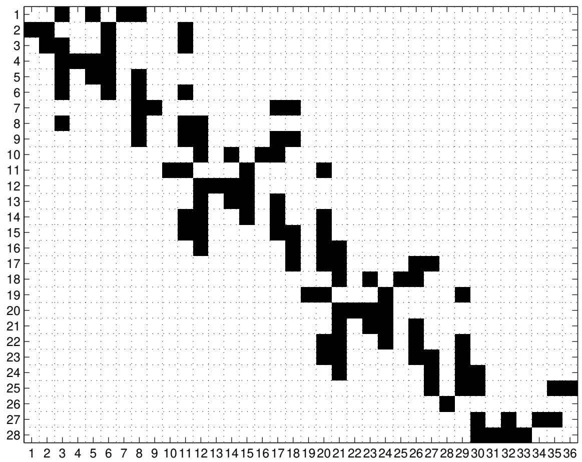


Figure 3.11: A representation of the cliques in the constrained Mildew jtree (Figure 3.9). Each row represents a clique, each column represents a DBN variable. The repetitive structure in the middle of the jtree is evident. For example, rows 17 and 25 are isomorphic, because their "bit pattern" is the same (101100011), and is exactly 9 columns apart. (The bit pattern refers to the 0s and 1s, which is a way of representing which DBN nodes belong to each clique set.)

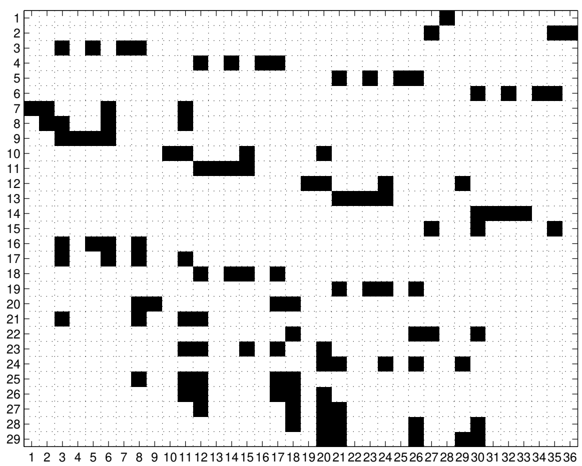


Figure 3.12: This is like Figure 3.11, but using an unconstrained elimination ordering. Notice how some cliques span more than two consecutive time slices, e.g., cliques 22 is $\{18, 26, 27, 30\}$.

Theorem (Constrained elimination ordering) [RTL76]. Let $A_1, \ldots, A_n$ be an elimination sequence triangulating the (moral) graph G, and let A_i, A_j be two non-neighbors in G, $i < j$. Then the elimination sequence introduces the fill-in $A_i - A_j$ iff there is a path $A_i - X_1 - \ldots - A_j$ such that all intermediate nodes X_k are eliminated before A_i.

For example, in Figure 3.8, for any $t \geq 4$, every node in slice t is connected to every other node in slice t via some (undirected) path through the past; hence all the nodes in slice t become connected in one big clique in the triangulated graph.

Another example is the coupled HMM in Figure 3.13. Even though chain 1 is not directly connected to chain 4, they become correlated once we unroll the DBN. Indeed, the unrolled DBN looks rather like a grid-structured Markov Random Field, for which exact inference is known to be intractable. (More precisely, the tree width of an $N = n \times n$ 2D grid with nearest-neighbor (4 or 8) connectivity is $O(n)$ [RS91].)

Although a constrained elimination order cannot yield smaller cliques than an unconstrained one, in

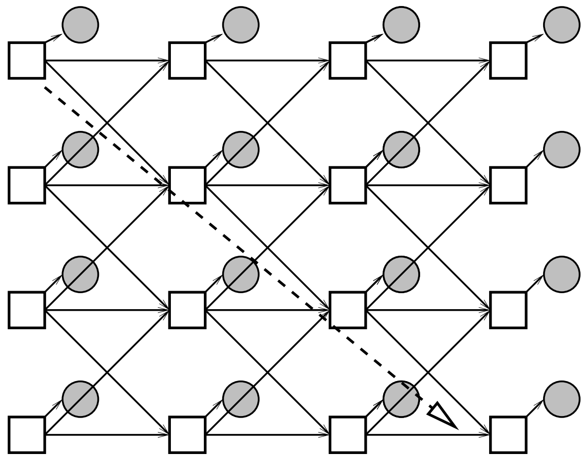


Figure 3.13: A coupled HMM with 4 chains. Even though chain 1 is not directly connected to chain 4, they become correlated once we unroll the DBN, as indicated by the dotted line.

my experience the constraint nearly always helps the min-fill heuristic find much better elimination orders; similar findings are reported in [Dar01]. (Recall that finding the optimal elimination order is NP-hard.) However, Bilmes reports (personal communication) that by using the unconstrained min-fill heuristic with multiple restarts, the best such ordering tends to be one that eliminates nodes which are not temporally far apart, i.e., it is similar to imposing a constraint that we eliminate all nodes within a small temporal window, but without having to specify the width of the window ahead of time (since the optimal window might span several slices). The cost of finding this optimal elimination ordering can be amortized over all inference runs.

3.5.4 Online inference

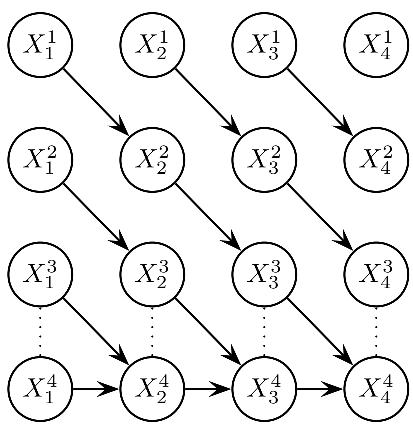
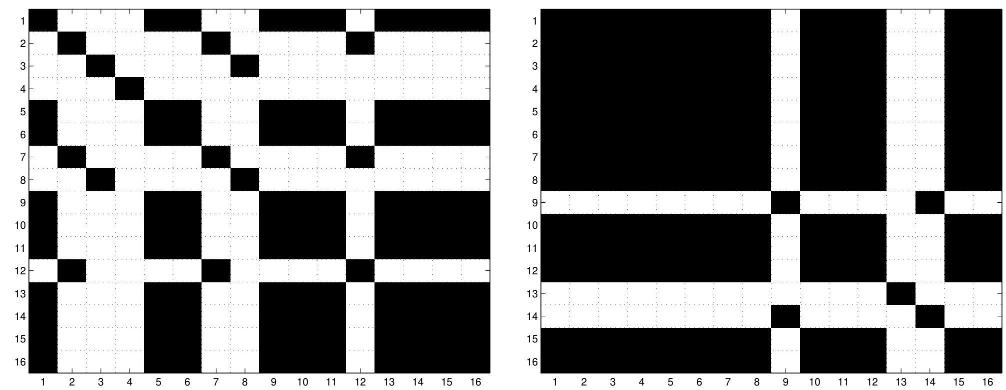


Figure 3.14: A DBN in which the nodes in the last slice do *not* become connected when we eliminate the first 3 or more slices. Dotted lines represent moralization arcs.

For an online algorithm to use constant space and time per iteration, there must be some finite time t at which it eliminates all earlier nodes. Hence online inference must used constrained elimination orderings.

The above suggests that perhaps online inference always takes at least $\Omega(K^{I+1})$ time. However, this is false: Figure 3.14 provides a counter-example. In this case, the largest clique in both the constrained and unconstrained jtree is the triple $\{X_{t-1}^3, X_{t-1}^4, X_t^4\}$, because there is no path through the past X's which connects all the nodes in the last slice. So in this case inference is cheaper than $O(K^I)$.

The interface in Figure 3.14 has size 4, even though it could be represented in factored form without loss of accuracy. We can identify this sort of situation as follows: unroll the DAG for D slices, where D is the



(a) (b)

Figure 3.15: The adjacency matrices for the closure of the moralized unrolled graphs for two DBNs. (a) The DBN in Figure 3.8; the bottom right 4×4 block is all 1s, indicating that the interface is all the nodes. (b) The DBN in Figure 3.14; the bottom right 4×4 block is sparse, indicating that the interface is $\{1\}$, $\{2\}$, and $\{3, 4\}$.

number of nodes per slice; moralize the unrolled graph; find the transitive closure of the moral graph; extract the subgraph corresponding to the last slice; finally, find the strongly connected components of the subgraph, and call them $C_1, \ldots, C_C$ — these are the factors in the interface. See Figure 3.15 for some examples.

We will denote this sequence of operations as

$$(C_1, \ldots, C_C) = cc\,(cl(m(U_D(G))) \cap V_D)$$

where $cc(G)$ means returns the connected components of G, $G \cap V_D$ means the subgraph of G containing nodes in slice D, $cl(G)$ means the transitive closure of G, $m(G)$ means the moral graph of G, and $U_D(G)$ means unroll the 2TBN G for D slices. We summarize our results as follows.

Conjecture. Consider a 2TBN G with D nodes per slice, each of which has a maximum of K possible discrete values. Let $(C_1, \ldots, C_C) = cc\,(cl(m(U_D(G))) \cap V_D)$ be the connected components, as described above. Let $m = \max_{i=1}^{C} C_i$ be the size of the largest interface factor, and let F_{in} be the maximal number of parents of any node within the same slice. Then the complexity of any online inference algorithm is at least $\Omega(K^{m+F_{in}+1})$.

We can achieve this lower bound using what I call the "factored interface" algorithm: this is a simple modification to the interface algorithm, which maintains the distributions over the C_i's in factored form.

3.5.5 Conditionally tractable substructure

The pessimistic result in the previous section is purely graph-theoretic, and hence is a worst-case (distribution-free) result. It sometimes happens that the CPDs encode conditional independencies that are not evident in the graph structure. This can lead to significant speedups. For example, consider Figure 3.16. This is a schematic for two "processes" (here represented by single nodes), B and C, both of which only interact with each other via an "interface" layer, R.

The forward interface is $\{R, B, C\}$; since B and C represent whole subprocesses, this might be quite large. Now suppose we remove the dotted arcs, so R becomes a "root", which influences B and C. Again, the forward interface is $\{R, B, C\}$. Finally, suppose that R is in fact a static node, i.e., $P(R_t|R_{t-1}) = \delta(R_t, R_{t-1})$. (For example, R might be a fixed parameter.) In this case, the model can be simplified as shown in Figure 3.17. This model enjoys the property that, conditioned on R, the forward interface factorizes [TDW02]:

$$P(R, B_t, C_t|y_{1:t}) = P(B_t|R, y_{1:t})P(C_t|R, y_{1:t})P(R|y_{1:t}) = P(B_t|R, y^B_{1:t})P(C_t|R, y^C_{1:t})P(R|y_{1:t})$$

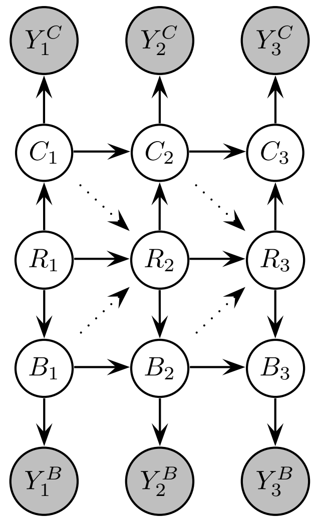


Figure 3.16: Two "processes", here represented by single nodes, B and C, only interact with each other via an "interface" layer, R.

where $y_t = (y_t^B, y_t^C)$. This follows since $B_t \perp y_{1:t}^C | R$ and $C_t \perp y_{1:t}^B | R$, i.e., evidence which is local to a process does not influence other processes: the R node acts like a barrier. Hence we can recursively update each subprocess separately:

$$\begin{aligned} P(B_t | R, y_{1:t}^B) &= P(B_t | R, y_{1:t-1}^B, y_t^B) \\ &\propto P(y_t^B | B_t) \sum_b P(B_t | B_{t-1} = b, R) P(B_{t-1} = b | R, y_{1:t-1}^B) \end{aligned}$$

The distribution over the root variable can be computed at any time using

$$P(R | y_{1:t}) \propto \sum_b P(R) P(B_t = b | R, y_{1:t})$$

The reason we cannot apply the same factoring trick to the model in Figure 3.16 is that $P(B_t | R_t, y_{1:t}) \neq P(B_t | R_t, y_{1:t}^B)$, since there is a path (e.g., via R_{t-1}) connecting Y_t^C to B_t. If we could condition on the whole chain $R_{1:t}$ instead of just on R_t, we could factor the problem. Of course, we cannot condition on all possible values of $R_{1:t}$, since there are K^t of them; however, we can *sample* representative instantiations. Given these samples, we can update the processes exactly. See Section 5.3.3 for an application of this idea to the SLAM problem.

We discuss how to exploit other kinds of "parametric" (non-graphical) conditional independence for exact inference in Section B.6.2. In general, the nodes in the interface will not be conditionally independent, even if we exploit parametric properties in the CPDs. However, they may be only weakly correlated. This is the basis of the approximation algorithms we discuss in the next two chapters.

3.6 Continuous state spaces

3.6.1 Inference in KFMs

The equations for Kalman filtering/ smoothing can be derived in an analogous manner to the equations for HMMs, except the algebra is somewhat heavier. Please see e.g., [BSL93, Min99, Jor02] for derivations. Here we just state the final results without proof.

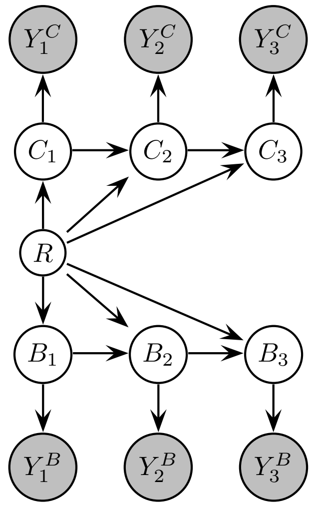


Figure 3.17: Conditioned on the static root, the interface is fully factored.

Forwards pass

Let us denote the mean and covariance of the belief state $P(X_t|y_{1:t})$ by $(x_{t|t}, V_{t|t})$. The forward operator,

$$(x_{t|t}, V_{t|t}, L_t) = \text{Fwd}(x_{t-1|t-1}, V_{t-1|t-1}, y_t; A_t, C_t, Q_t, R_t)$$

is defined as follows.[9] First, we compute the predicted mean and variance.

$$\begin{aligned} x_{t|t-1} &= Ax_{t-1|t-1} \\ V_{t|t-1} &= AV_{t-1|t-1}A' + Q \end{aligned}$$

Then we compute the error in our prediction (the innovation), the variance of the error, the Kalman gain matrix, and the conditional log-likelihood of this observation.

$$\begin{aligned} e_t &= y_t - Cx_{t|t-1} \\ S_t &= CV_{t|t-1}C' + R \\ K_t &= V_{t|t-1}C'S_t^{-1} \\ L_t &= \log \mathcal{N}(e_t; 0, S_t) \end{aligned}$$

Finally, we update our estimates of the mean and variance.

$$\begin{aligned} x_{t|t} &= x_{t|t-1} + K_t e_t \\ V_{t|t} &= (I - K_t C)V_{t|t-1} = V_{t|t-1} - K_t S_t K_t' \end{aligned}$$

These equations are more intuitive than they may seem. For example, our expected belief about x_t is equal to our prediction, $x_{t|t-1}$, plus a weighted error term, $K_t e_t$, where the weight $K_t = V_{t|t-1}C'S_t^{-1}$, depends on the ratio of our prior uncertainty, $V_{t|t-1}$, to the uncertainty in our error measurement, S_t.

Backwards pass

The backwards operator

$$(x_{t|T}, V_{t|T}, V_{t-1,t|T}) = \text{Back}(x_{t+1|T}, V_{t+1|T}, x_{t|t}, V_{t|t}; A_{t+1}, Q_{t+1})$$

[9]Normally random variables are upper case letters, and their values are lower case; however, in this and other sections, we follow the other convention that scalars and vectors are lower case, and matrices are upper case.

is defined as follows (this is the analog of the γ recursion in Section 3.2.3). First we compute the following predicted quantities (or we could pass them in from the filtering stage):

$$\begin{aligned} x_{t+1|t} &= A_{t+1} x_{t|t} \\ V_{t+1|t} &= A_{t+1} V_{t|t} A'_{t+1} + Q_{t+1} \end{aligned}$$

Then we compute the smoother gain matrix.

$$J_t = V_{t|t} A'_{t+1} V_{t+1|t}^{-1}$$

Finally, we compute our estimates of the mean, variance, and cross variance $V_{t,t-1|T} = \text{Cov}[X_{t-1}, X_t | y_{1:T}]$.[10]

$$\begin{aligned} x_{t|T} &= x_{t|t} + J_t \left(x_{t+1|T} - x_{t+1|t} \right) \\ V_{t|T} &= V_{t|t} + J_t \left(V_{t+1|T} - V_{t+1|t} \right) J'_t \\ V_{t-1,t|T} &= J_{t-1} V_{t|T} \end{aligned}$$

These equations are known as the Rauch-Tung-Striebel (RTS) equations. It is also possible to derive a two-filter approach to smoothing, as for HMMs (see [Jor02, ch13] and [FP69]). In this case, the analog of Π_t is Σ_t, which can be computed from the following Lyapunov equation:

$$\Sigma_t = E[X_t X'_t] = E[(AX_{t-1} + W_t)(AX_{t-1} + W_t)'] = AE[X_{t-1} X'_{t-1}]A' + E[W_t W'_t] = A\Sigma_{t-1} A' + Q$$

The limit of this, $\lim_{t\to\infty} \Sigma_t$, is known as the Riccati matrix.

Time and space complexity of Kalman filtering/smoothing

If $X_t \in \mathbb{R}^{N_x}$, and $Y_t \in \mathbb{R}^{N_y}$, then K_t is a $N_x \times N_y$ matrix, S_t is a $N_y \times N_y$ matrix, and A_t is a $N_x \times N_x$ matrix. The cost per update is $O(\min(N_x^2, N_y^3))$; the N_x^2 term arises from the $Ax_{t-1|t-1}$ computation; the N_y^3 term arises from inverting S_t. (For sparse matrices, it is possible to reduce the computational complexity considerably, especially if we use the information filter.) If we want to track N objects, each of size N_x, inference takes $O((N_x N)^2)$; hence KFMs do not suffer from the exponential growth in complexity that HMMs do. However, they are rather restrictive in their assumptions: see Section 1.3.4.

3.6.2 Inference in general linear-Gaussian DBNs

Any DBN in which all CPDs are linear-Gaussian can be converted to a KFM, just as any DBN in which all hidden variables are discrete can be converted to an HMM. If we have D hidden continuous variables, each a vector of size K, the compound state-space will have size KD, and inference (using the Kalman filter/smoother) will take $O((KD)^2)$ time per step. Hence the complexity only grows polynomially in D, unlike the discrete case, in which the complexity grows exponentially in D. Furthermore, the matrices of the compound KFM will be sparse, often block diagonal, reflecting the graph structure. Hence a standard Kalman filter/smoother will usually be as efficient as using the junction tree algorithm.

3.6.3 Switching KFMs

Consider the switching KFM in Figure 2.31. This is a conditionally Gaussian (CG) model (see Section B.5): if we knew $S_{1:T}$, the distribution over $X_{1:T}, Y_{1:T}$ would be jointly Gaussian. [LSK01] prove that inference in CG networks is NP-hard, even if the structure is a (poly)tree, such as a switching KFM where the horizontal arcs between the switch nodes are absent. The proof is by reduction from subset sum, but the intuition is as follows: the prior $P(X_1)$ is a mixture of K Gaussians, depending on S_1; when we marginalize out S_1, we are still left with a mixture of K Gaussians; each one of these gets "passed through" K different matrices, depending on the value of S_2, resulting in a mixture of K^2 Gaussians, etc; in general, the belief state at time t is a mixture of K^t Gaussians. We discuss some approximate inference algorithms for this model in Sections 4.3 and 5.3.1.

[10] The cross variance term is usually computed recursively, but this is unnecessary [Min99], c.f., in an HMM, we can compute $\xi_{t-1,t}$ directly rather than recursively, as shown in Section 3.2.4.

```
function smoother(y_{1:T})
f_{1|1} = Fwd1(y_1)
for t = 2 : T
    f_{t|t} = Fwd(f_{t-1|t-1}, y_t)
b_{T|T} = BackT(f_{T|T})
for t = T - 1 : 1
    b_{t|T} = Back(b_{t+1|T}, f_{t|t})
```

Figure 3.18: Pseudo-code for offline smoothing.

3.6.4 Non-linear/ non-Gaussian models

KFMs support exact inference because of two facts: a Gaussian pushed through a linear transformation, and subjected to additive Gaussian noise, still results in a Gaussian; and updating a Gaussian prior with a Gaussian likelihood, using Bayes' rule, results in a Gaussian posterior. In other words, Gaussians are closed under Bayesian updating in KFMs. A few other distributions enjoy this property (see [WH97]), but in general, exact inference in DBNs with hidden continuous nodes which have CPDs other than linear-Gaussian is not possible.[11] In particular, exact inference in systems which are non-linear and/or non-Gaussian is usually not possible. We must therefore resort to approximations. There are essentially two classes of approximations, deterministic and stochastic: see Chapters 4 and 5 respectively.

3.7 Online and offline inference using forwards-backwards operators

We have now seen several different ways of performing filtering and smoothing in DBNs. To hide these details from higher-level inference algorithms, we defined the abstract forwards and backwards operators. In this section, we use these operators to derive generic efficient filtering and smoothing algorithms.

3.7.1 Space-efficient offline smoothing (the Island algorithm)

In Figure 3.18 we give pseudo-code for computing $(b_{1|T}, \ldots, b_{T|T})$ using a generalized forwards-backwards algorithm. Rather than returning $(b_{1|T}, \ldots, b_{T|T})$, we assume the output is "emitted" to some "consumer" process which e.g., computes the sufficient statistics needed for learning, $\sum_t b_{t|T}$, which can be represented in $O(S)$ space, where S is the size of a forwards or backwards message for a single time-slice.[12]

Although the expected sufficient statistics take constant space, the above algorithm uses $O(ST)$ temporary storage, since it needs to store $f_{t|t}$ for $t = 1, \ldots, T$. To see that this might be a problem, consider the BATnet shown in Figure 3.19. Here, $S \sim 10^3$. To learn the parameters of this model, we need training sequences of length at least $T \sim 10^5$. Hence the storage requirements are $\sim 10^8$ bytes. Longer sequences or more complex models (e.g., for speech recognition or biosequence analysis) will clearly require much more space.

We now present a way to reduce the space requirements from $O(ST)$ to $O(SC \log_C T)$, where C is a tunable parameter of the algorithm to be explained below. The catch is that the running time increases from $O(T)$ to $O(T \log_C T)$. (This algorithm was first presented in [BMR97a]; an equivalent algorithm is presented in [ZP00].)

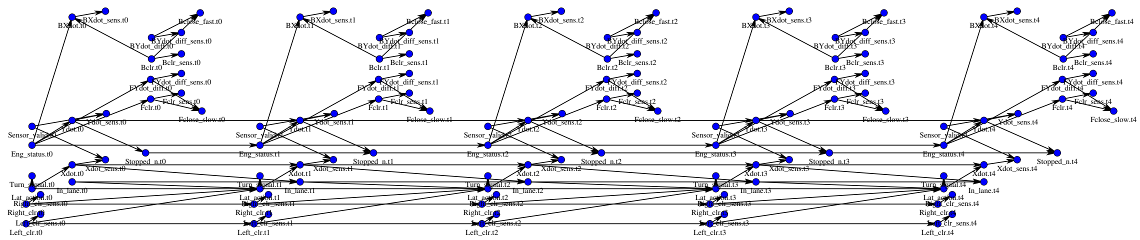


Figure 3.19: The BATnetwork (see Figure 4.2) unrolled for 5 slices.

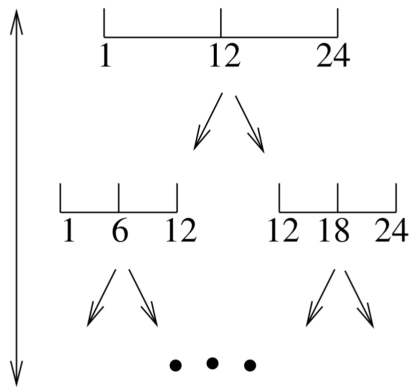


Figure 3.20: The basic idea behind the Island algorithm, with $T = 24, C = 1$. At the top level, we call Island$(f_0, b_{25}, y_{1:24})$, which computes b_1, b_{12} and b_{24}. (f_0 and b_{25} are dummy boundary conditions used at the top level.) Then we call Island$(f_1, b_{12}, y_{2:11})$, followed by Island$(f_{12}, b_{24}, y_{13:23})$.

Basic idea

The key insight is that, given a "boundary condition" on the left, we can do the forwards pass in constant space by recursively computing $f_{t|t} = \text{Fwd}(f_{t-1|t-1}, y_t)$ and throwing away the result at each step; and similarly for the backwards pass, given a boundary condition on the right. In the island algorithm, we run the forwards/backwards algorithm, but instead of throwing away the results, we store the "messages" at C "islands" or "checkpoints" (plus at the two boundaries). This divides the problem into $C + 1$ subproblems, on which we can call the algorithm recursively. See Figure 3.20.[13]

Time and space complexity of the island algorithm

The total running time of the algorithm satisfies the following recurrence:

$$R(T) = C\frac{T}{C} + CR\left(\frac{T}{C}\right)$$

since we are dividing a sequence of length T into C sequences of length T/C, each of which takes $O(T/C)$ time to compute; then we must call the function C times recursively. Solving this recurrences gives $R(T) = O(T \log_C T)$.

[11] Exact inference in DBNs in which all hidden nodes are discrete is always possible, although it may not be efficient. Continuous nodes that are observed do not cause a problem, no matter what their distribution, since they only affect the inference by means of the conditional likelihood, c.f., the $O_t(i, i) = P(y_t|X_t = i)$ term for HMMs.

[12] In the Hugin architecture, we store clique and separator potentials for the $1\frac{1}{2}$ slice DBN; for the Shafer-Shenoy architecture, we just store the clique potentials (see Section B.4.5). When applied to an HMM with K states, Hugin uses $S = K^2$ space, whereas Shafer-Shenoy uses $S = K$ space.

[13] A related approach to space-efficient inference in arbitrary Bayes nets appears in [Dar00]. Standard exact inference takes $O(ne^w)$ time and space, where n is the number of nodes and w is the treewidth. Darwiche's algorithm reduces the space requirement to $O(n)$ but increases the time requirement to $O(ne^{w \log n})$. In the current context, $n = O(T)$, so this is not a big win.

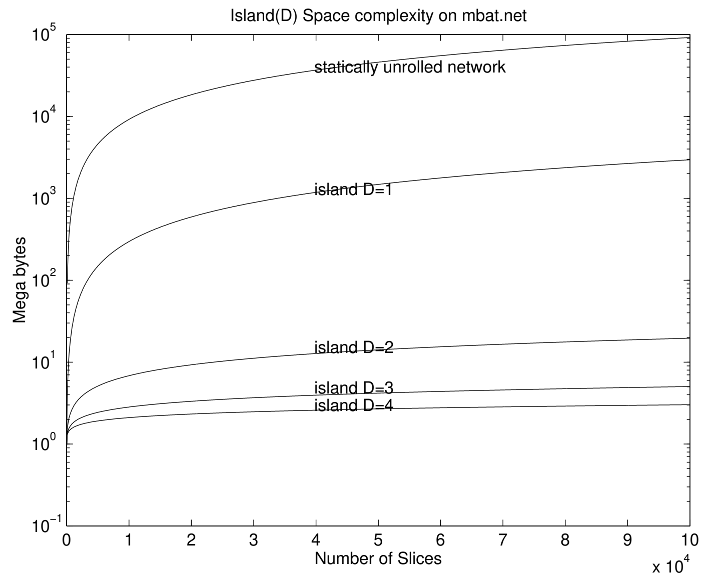


Figure 3.21: Results of the Island algorithm on the BATnet. D represents the maximum depth of recursion allowed before switching over to the linear-space algorithm.

The total amount of space is bounded by the depth of the tree, $D \leq \log_C T$, times the amount of space needed at each recursive invocation, which is $O((C+2)S)$. (This is because we must store checkpoints at C positions plus the two boundaries at every level on the call stack.) So the total space required is $O(SC \log_C T)$.

We can tradeoff the space vs time requirements by varying the number of checkpoints, C. For example, for the BATnet, for $C = 2$, we reduce the space requirements from 10^{11} to 10^7, but the running time increases by a factor of 16. If we use $C = \sqrt{T} = 316$ checkpoints, the space decreases from 10^{11} to 10^8, but the running time only doubles (since $\log_{\sqrt{T}} T = 2$). We can also vary the depth at which we "bottom out", i.e., switch over to the linear space algorithm. Some results on the BATnet are shown in Figure 3.21

Pseudo-code

In Figure 3.22, we give pseudo-code for computing $b_{1|T}, \ldots, b_{T|T}$ in a non-sequential order using space for $C + 2$ messages. When the length of the sequence is less than T_{min}, we switch over to the usual $O(T)$-space smoothing routine. To simplify notation, we assume that Fwd(f, y_t) calls Fwd1(y_t) when f is the (initial) empty message $\emptyset$; similarly, Back(b, f) calls BackT(f) when b is the initial empty message. (This saves us from having to clutter the code with if-then statements that check for boundary conditions.) Also, we use the Matlab notation $a : s : b$ to denote the list $(a, a+s, \ldots, a+cs)$, where $c = \lfloor (b-a)/s \rfloor$. $a : 1 : b$ will be shortened to $a : 1$, and $a : -1 : b$ counts down. We start by calling Island$(\emptyset, \emptyset, y_{1:T}, C, T_{\text{min}})$.

Example

Suppose $C = 1$, $T = 24$ and $T_{\text{min}} = 6$, as in Figure 3.20. We compute $f[1] = f_{1|1}$, $f[2] = f_{12|12}$, $f[3] = f_{24|24}$, $b[1] = b_{1|24}$, $b[2] = b_{12|24}$, and $b[3] = b_{24|24}$; we also store the positions of the checkpoints: $t[1] = 1$, $t[2] = 12$, $t[3] = 24$. After emitting $b[1]$, $b[2]$ and $b[3]$, we call Island$(f[1], b[2], y_{2:11})$ and then Island$(f[2], b[3], y_{13:23})$.

Constant-space smoothing?

Here is a trivial algorithm to do smoothing in $O(1)$ space (i.e., independent of T); unfortunately it takes $O(T^2)$ time, rendering it useless in practice. The algorithm is as follows [Dar01]: for each $1 \leq k \leq T$, do the following: first run a forwards filter from $t = 1$ to $t = k$, to compute α_k in constant space (only keeping

```
function Island(f, b, y, C, T_min)
T = length(y)
if T ≤ T_min
  base-case(f, b, y); return
f[1] = f; t[1] = 1; k = 2
stepsize = ⌊T/(C + 1)⌋
checkpoints = 1 : stepsize : T  ∪  {T}
for t = 1 : T
    f = Fwd(f, y_t)
    if t ∈ checkpoints
      f[k] = f; t[k] = t; k = k + 1
k = C + 2; b[k] = b
for t = T : 1
    b = Back(b, f)
    if t ∈ checkpoints
      b[k] = b; emit b; k = k − 1
for k = 2 : C + 2
    Island(f[k − 1], b[k], y(t[k − 1] + 1 : t[k] − 1))


function base-case(f, b, y)
T = length(y)
f[0] = f
for t = 1 : T
    f[t] = Fwd(f[t − 1], y_t)
for t = T downto 1
    b = Back(b, f[t])
    emit b
```

Figure 3.22: Pseudo-code for the Island algorithm (space-efficient smoothing).

```
f_{1|1} = Fwd1(y_1)
for t = 2 : L
    f_{t|t} = Fwd(f_{t-1|t-1}, y_t)
for t = L + 1 : ∞
    f_{t|t} = Fwd(f_{t-1|t-1}, y_t)
    b_{t|t} = BackT(f_{t|t})
    for τ = t − 1 downto t − L
        b_{τ|t} = Back(b_{τ+1|t}, f_{τ|τ})
    emit b_{t-L|t}
```

Figure 3.23: Pseudo-code for fixed-lag smoothing, first attempt.

α_{t-1} and α_t in memory at each step t), then run a backwards filter from $t = T$ downto $t = k$, to compute β_k in constant space, and then combine the results.

Here is a hypothetical algorithm for smoothing in $O(1)$ space and $O(T)$ time. First run the forwards algorithm, $f_{t|t} = \text{Fwd}(f_{t-1|t-1}, y_t)$, forgetting all the intermediate results (i.e., only keeping $f_{T|T}$), and then run the backwards algorithm, computing $f_{t|t} = \text{Fwd}^{-1}(f_{t+1|t+1}, y_{t+1})$ and $b_{t|T} = \text{Back}(b_{t+1|T}, f_{t|t})$ in parallel. We call this the "country dance" algorithm, because you can think of the forward pass as running to the end to pick up its "partner", $b_{T|T}$, and the two of them moving together back to the start.

Unfortunately, inverting the forward operator is impossible in general: to compute $P(X_t|y_{1:t})$ from $P(X_{t+1}|y_{1:t+1})$ requires figuring out how to "undo" the effect of the transition and the observation y_{t+1}. This is most easily seen from the forwards equation for HMMs:

$$\alpha_{t+1} \propto O_{t+1} A' \alpha_t$$

Inverting this yields

$$\alpha_t \propto (A')^{-1} O_{t+1}^{-1} \alpha_{t+1}$$

But any system in which there is more than one way of reaching a state cannot be inverted. Also, any system with invertible observations is essentially fully observed, so no inference is necessary. (The same argument holds for inverting the backwards operator.) Although this does not constitute a formal proof, I conjecture that smoothing in $O(S)$ space and $O(ST)$ time is impossible (see also Section 3.7.2).

3.7.2 Fixed-lag (online) smoothing

Fixed-lag smoothing estimates $P(X_{t-L}|y_{1:t})$, where $L > 0$ is the lag. If the delay can be tolerated, then this is clearly a more accurate estimate than the filtered quantity $P(X_{t-L}|y_{1:t-L})$, which does not take "future" evidence into account. (The benefit of using a smoother depends on the signal-to-noise ratio.)

One way to implement this is to augment the state-space of the DBN with lagged copies of X_t as in the blind deconvolution model in Section 2.4.3. In the case of KFMs, the the Kalman filter equations can be modified in a straightforward way [Moo73] to implement filtering in this model, which is equivalent to fixed-lag smoothing in the original model.

For the KFM case, the state-space grows from $O(S)$ to $O(LS)$, and the computation grows from $O(S^2)$ to $O(LS^2)$ per time step, where $S = |X|$. However, discrete state-spaces grow from $O(S)$ to $O(S^L)$, and the computation also grows exponentially. Hence we now present an algorithm that uses $O(SL)$ time and space to do fixed-lag smoothing in arbitrary DBNs.

In Figure 3.23, we show a first attempt at implementing a fixed-lag smoother. Suppose $L = 2$; after $t > 2$, this will "emit" $b_{1|3}$, $b_{2|4}$, $b_{3|5}$, etc. to some "consumer" process. Since this is an online algorithm, we cannot store the messages for all t. Hence we must use a wrap-around buffer of length $L+1$, as in Figure 3.24. ($f[i]$ is the i'th entry in this buffer; k is a pointer to the position in the buffer that contains the most recent forward message. We assume one-based indexing, and use the notation $t \oplus 1$ and $t \ominus 1$ to represent addition/ subtraction modulo L.)

$f[1] = \text{Fwd1}(y_1)$
for $t = 2 : L$
    $f[t] = \text{Fwd}(f[t-1], y_t)$
$k = L + 1$
for $t = L + 1 : \infty$
    $(b_{t-L|t}, f[1 : L], k) = FLS(y_t, f[1 : L], k)$

function $(b, f[1 : L], k) = FLS(y_t, f[1 : L], k)$
L = length(f)
$k' = k \ominus 1$
$f[k] = \text{Fwd}(f[k'], y_t)$
$b = \text{BackT}(f[k])$
for $\tau = 1 : L$
    $b = \text{Back}(b, f[k'])$
    $k' = k' \ominus 1$
$k = k \oplus 1$

Figure 3.24: Pseudo-code for fixed-lag smoothing, final version.

$f_{1|1} = \text{Fwd1}(y_1)$
$b_{1|1} = \text{BackT}(f_{1|1})$
for $t = 2 : \infty$
    $f_{t|t} = \text{Fwd}(f_{t-1|t-1}, y_t)$
    $b_{t|t} = \text{BackT}(f_{t|t})$

Figure 3.25: Pseudo-code for online filtering.

Time and space complexity of fixed-lag smoothing

It is clear that the FLS algorithm takes $O(LS)$ time and space. We could implement this in $O(S)$ time and space if we could compute $b_{t-L|t}$ from $b_{t-L-1|t-1}$ (see e.g., [RN02, ch.17]). However, this requires that the model be invertible, as in Section 3.7.1. I conjecture that constant time fixed-lag smoothing (i.e., an algorithm which does an amount of work which is independent of the lag) is impossible in general.

Constant-time FLS would entail changing our beliefs about an event long in the past without having to consider any intermediate events; this amounts to "action at a distance", which seems impossible. However, if we had a hierarchical model, we might be able to approximate this behavior by reasoning (message passing) at a longer length/time scale; see Section 2.3.9 for a discussion of such a hierarchical model.

3.7.3 Online filtering

Filtering is just fixed-lag smoothing where the lag $L = 0$. In this case, the above code simplifies to the code shown in Figure 3.25. Note that, in general, we have to call the backwards operator even though we are doing filtering; however, we only do the backwards pass within one slice. This will be explained in Section 3.4.2.

Since this is an online algorithm, we cannot store the messages for all t. However, we can update $f_{t|t}$ and $b_{t|t}$ in place. Hence filtering takes $O(S)$ space and time per step (where the constant depends on the size of the model, but not on t).[14]

[14]This of course assumes the basic operators can be implemented in closed form. This is not true e.g., for a switching KFM. See Section 3.6.3

Chapter 4

Approximate inference in DBNs: deterministic algorithms

4.1 Introduction

In the case of discrete state-spaces, exact inference is always possible, but may be computationally prohibitive, since it takes $\Omega(K^{\max_i\{C_i\}+F_{in}+1})$ operations per time step, where C_i is the size of the i'th clique in the forward interface, and F_{in} is the maximal numer of parents within the same slice of any node (see Section 3.5.4). Typically $\max\{C_i\} = O(D)$, the number of hidden persistent nodes per slice, making exact inference intractable for large discrete DBNs.

A standard approximation in the discrete case, known as the Boyen-Koller (BK) algorithm [BK98b], is to approximate the joint distribution over the interface as a product of marginals. Unfortunately, sometimes even this approximation can be intractable, since BK does exact inference in a 2-slice DBN. This motivates the factored frontier (FF) algorithm [MW01], which can be seen as a more aggressive form of approximation than BK. I show that BK and FF are both special cases of loopy belief propagation (LBP). In Section 4.2.4, I experimentally compare exact, FF, BK, and LBP on a number of DBNs. The FF algorithm, the connection between FF, BK and LBP, and the experimental evaluation are the main novel contributions of this chapter.

In the case of continuous state-spaces, if everything is linear-Gaussian, we can apply the Kalman filter/smoother. But if we have non-linear and/or non-Gaussian distributions, exact inference is usually not possible. I briefly review some deterministic approximations for this case.

Finally, I consider the problem of mixed discrete-continuous state-spaces. As we saw in Section 3.6.3, exact inference in a switching KFM takes $O(K^t)$ operations at the t'th time step. In Section 4.3, I review the classica moment matching approximate filtering algorithm [BSL93], and its extension to the smoothing case [Mur98]. I also describe how to apply expectation propagation (EP) [Min01] to a switching KFM [HZ02]. It turns out that (in this case) EP is just an iterated version of moment matching, just as LBP is just an iterated version of FF.

Since inference in DBNs uses inference in BNs as a subroutine, you are strongly recommended to read Appendix B before reading this chapter.

Very little is known about the accuracy of the approximation algorithms discussed in this chapter (in contrast to the Monte Carlo approximations in the next chapter). In Section 4.2.1, we state the main result for the BK algorithm which shows the approximation error remains bounded; however, it is unclear how tight the bound is. The theoretical results on loopy belief propagation (LBP) discussed in Section B.7.1 also apply to the DBN case; again, however, it is often intractable to compute the expression which determines the accuracy. At the time of writing, the author is unaware of any theoretical results on expectation propagation (and hence of moment matching, which is just a special case of EP).

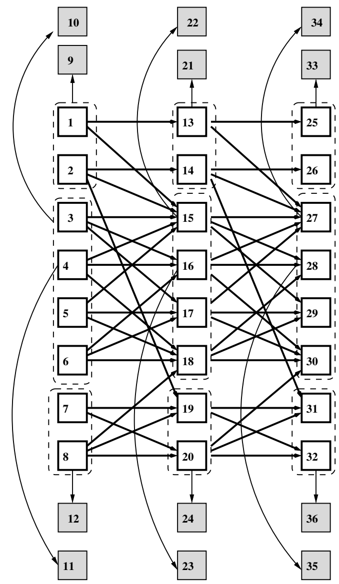


Figure 4.1: The water DBN, designed to monitor a waste water treatement plant. This model is originally from [JKOP89], and was modified by [BK98b] to include (discrete) evidence nodes. The dotted arcs group together nodes that will be used in the BK approximation: see Section 4.2.1. Nodes are numbered in topological order, as required by BNT.

4.2 Discrete-state DBNs

4.2.1 The Boyen-Koller (BK) algorithm

If the interface cliques are too large, one approach is to approximate the joint on the interface as the product of marginals of smaller terms; this is the basic idea behind the BK algorithm [BK98b, BK98a, BK99].

More precisely, BK constructs the junction tree for the $1\frac{1}{2}$ slice DBN H_t, but does not require all the interface nodes to be in the same clique; i.e., $P(I_t|y_{1:t})$ is no longer represented as a single clique potential. Instead, we approximate the belief state by a product of marginals, $P(I_t|y_{1:t}) \approx \prod_{c=1}^{C} P(I_t^c|y_{1:t})$, where $P(I_t^c|y_{1:t})$ is the distribution on nodes in cluster c. The set of clusters partitions the nodes in the interface. Hence rather than connecting together all the nodes in the interface, we just connect together the nodes in each cluster. The assumption is that this will result in smaller cliques in the 2TBN. Figures 4.3 and 4.4 show that the max clique size is indeed reduced in some cases. The basic reason is that we only had to triangulate the two-slice network, not the unrolled network.

The accuracy of the BK algorithm depends on the clusters that we use to approximate the belief state. Exact inference corresponds to using a single cluster, containing all the interface nodes. The most aggressive approximation corresponds to using D clusters, one per variable; we call this the "fully factorized" approximation.

We can implement BK by slightly modifying the interface algorithm (see Section 3.4), as we explain below.

Forwards pass

1. Initialize all clique and separator potentials in J_t to 1s, as before.

2. Include the CPDs and evidence for slice t as before.

3. To incorporate the prior, we proceed as follows. For each cluster c, we find the smallest cliques C_{t-1} and D_t in J_{t-1} and J_t whose domains include c; we then marginalize $\phi_{C_{t-1}}$ onto I_{t-1}^c and multiply this onto ϕ_{D_t}.

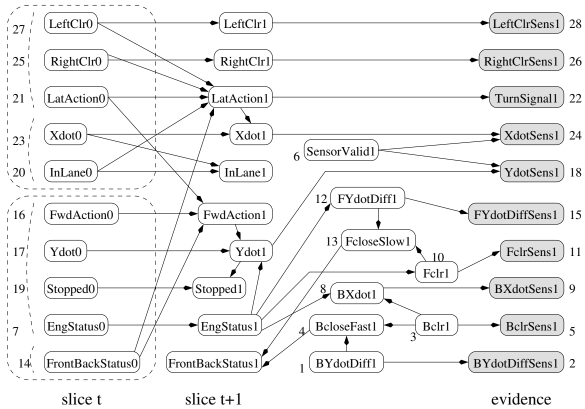
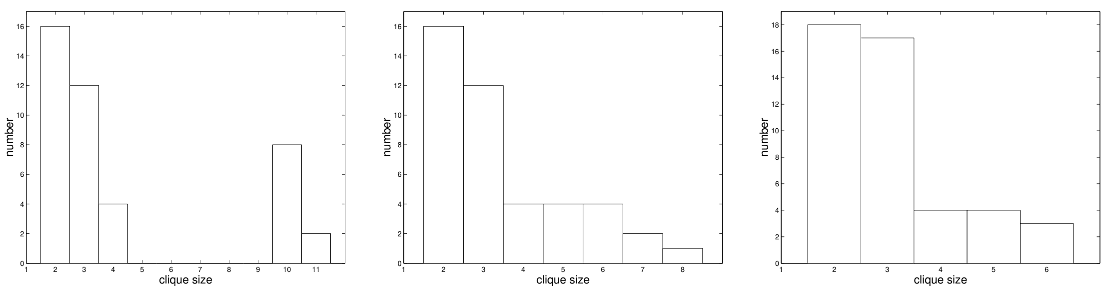
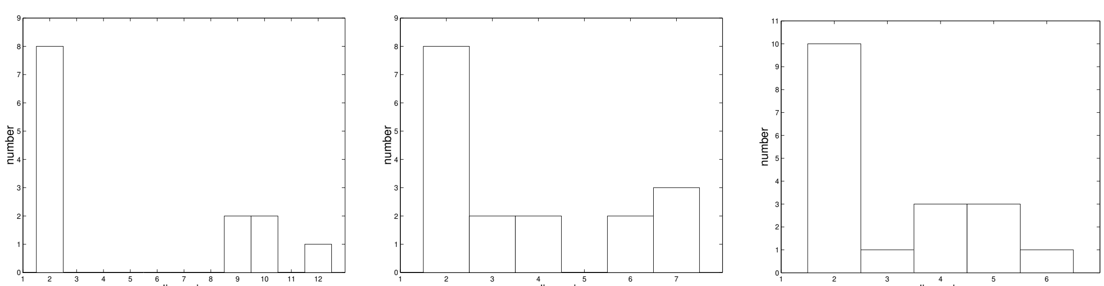


Figure 4.2: The BATnet, designed to monitor the state of an autonomous car (the Bayesian Automated Taxi, or BATmobile) [FHKR95]. The transient nodes are only shown for the second slice, to minimize clutter. The dotted arcs group together nodes that will be used in the BK approximation: see Section 4.2.1. Thanks to Daphne Koller for providing this figure. Nodes are numbered in topological order, as required by BNT.



(a) (b) (c)

Figure 4.3: Clique size distribution for different levels of BK approximation applied to the BATnet (Figure 4.2). The cliques of size 2 are due to the observed leaves. (a) Exact: the interface is $\{7, 14, 16, 17, 19, 20, 21, 23, 25, 27\}$. (b) Partial approximation: the interface sets are $\{7, 14, 16, 17, 19\}$ and $\{20, 21, 23, 25, 27\}$, as in Figure 4.2. (c) Fully factorized approximation: the interface sets are the singletons.



(a) (b) (c)

Figure 4.4: Clique size distribution for different levels of BK approximation applied to the water DBN (Figure 4.1). The cliques of size 2 are due to the observed leaves. (a) Exact: the interface is $\{1, \ldots, 8\}$. (b) Partial approximation: the interface sets are $\{1, 2\}$, $\{3, 4, 5, 6\}$ and $\{7, 8\}$, as in Figure 4.1. (c) Fully factorized approximation: the interface sets are the singletons.

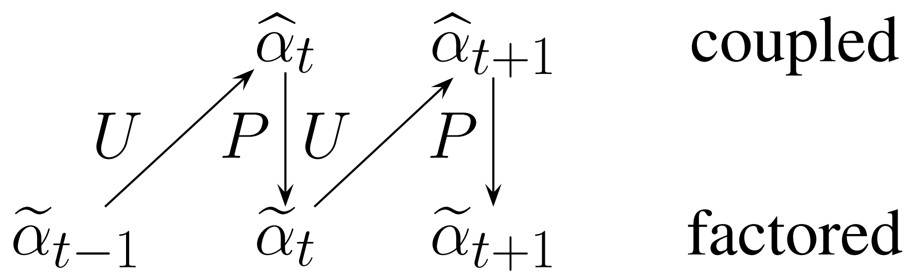


Figure 4.5: The BK algorithm as a series of update (U) and projection (P) steps.

4. We collect and distribute evidence to/from any clique.

The reason we must collect *and distribute* evidence is because the clusters which constitute the interface may be in several cliques, and hence all of them must be updated before being used as the prior for the next step. If there is only one cluster, it suffices to collect to the clique that contains this cluster.

Backwards pass

A backwards (smoothing) pass for BK was described in [BK98a]. ([KLD01] describes a similar algorithm for general BNs called "up and down mini buckets".) Again, it can be implemented by a slight modification to the interface algorithm:

1. Construct J_t from $f_{t|t}$ as before.
2. To incorporate the backwards message, we proceed as follows. For each cluster c, we find the smallest cliques D_{t+1} in J_{t+1} and C_t in J_t whose domains include c; we then marginalize $\phi_{D_{t+1}}$ onto I_t^c and absorb it into ϕ_{C_t}.
3. We collect and distribute evidence to/from any clique.

Analysis of error in BK

The forwards pass may be viewed more abstractly as shown in Figure 4.5. We start out with a factored prior, $\tilde{\alpha}_{t-1}$, do one step of exact Bayesian updating using the junction tree algorithm (which may couple some of the clusters together), and then "project" the coupled posterior $\hat{\alpha}_t$ down to a factored posterior, $\tilde{\alpha}_t$, by computing marginals. This projection step computes $\tilde{\alpha}_t = \arg\min_{q \in \mathcal{S}} D(\hat{\alpha}_t || q)$, where $\mathcal{S}$ is the set of factored distributions, and $D(p||q) \stackrel{\text{def}}{=} \sum_x p(x) \log p(x)/q(x)$ is the Kullback-Liebler divergence [CT91]. This is an example of assumed density filtering (ADF) [Min01].

As an algorithm, BK is quite straightforward. The main contribution of [BK98b] was the proof that the error remains bounded over time, in the following sense:

$$E\ KL(\alpha_t || \tilde{\alpha}_t) \leq \frac{\epsilon_t}{\gamma^*}$$

where the expectation is take over the possible observation sequences, γ^* is the mixing rate of the DBN, and ϵ_t is the additional error incurred by projection, over and above any error inherited from the factored prior:

$$\epsilon_t = KL(\alpha_t || \tilde{\alpha}_t) - KL(\alpha_t || \hat{\alpha}_t).$$

The intuition is that, even though projection introduces an error at every time step, the stochastic nature of the transitions, and the informative nature of the observations, reduces the error sufficiently to stop it building up.

The best case for BK is when the observations are perfect (noiseless), or when the transition matrix is maximally stochastic (i.e., $P(X_t|X_{t-1})$ is independent of X_{t-1}), since in either case, errors in the prior are irrelevant. Conversely, the worst case for BK is when the observations are absent or uninformative, and the transition matrix is deterministic, since in this case, the error grows over time. (This is consistent with the theorem, since deterministic processes can have infinite mixing time.) In Section 5.2, we will see that the best case for BK is the worst case for particle filtering (when we propose from the prior), and vice versa.

4.2.2 The factored frontier (FF) algorithm

Just as BK represents the interface distribution in factored form, so FF represents the frontier distribution in factored form. Of course, we don't need the whole frontier, only the interface, so the real difference is in the way these approximate distributions are updated. BK takes in a factored prior, does one step of exact updating using the jtree algorithm, and then projects the results back down to factored form. Unfortunately, for some models, even one step of exact updating is intractable. For example, consider a model of a video sequence, in which each time slice is an $N = n \times n$ 2D lattice, and in which $X_t(i, j)$ has a connection from $X_{t-1}(i, j)$. in this case, the largest clique in the 2TBN has size at least $n = \sqrt{N}$, since this is the size of the max clique in a 2D grid without any additional incoming temporal arcs. Hence the running time of BK is at least $\Omega(TNK^{\sqrt{N}})$, even in the case of fully factorized BK.

Instead of doing exact updating and then projecting, FF computes the marginals directly. That is, when we add a node to the frontier, we do not multiply its CPD onto the joint frontier, but just compute a "local" joint over the parents, multiply the CPD onto this, and then marginalize out:

$$P(X_t^i|E) = P(X_t^i|\text{Pa}(X_t^i)) \prod_{u \in \text{Pa}(X_t^i)} P(u|E)$$

where the marginal distributions over the parents, $P(u|E)$, are available from the frontier (since we never add a node until all its parents are in the frontier), and E is the evidence in, and to the left of, the frontier. Adding a node to the frontier entails adding its marginal, computed as above, to the list of marginals. Removing a node from the frontier entails removing the corresponding marginal from the list, which of course takes constant time. The backwards pass is analogous. This algorithm clearly takes $O(TDK^{F_{in}+1})$ time, where D is the number of nodes per slice, no matter what the DBN topology.

4.2.3 Loopy belief propagation (LBP)

Loopy belief propagation for general graphical models is explained in Section B.7.1.

FF is a special case of LBP

The key assumption in LBP is that the messages coming into a node are independent (which is true in a tree). But this is exactly the same assumption that we make in the FF algorithm! In fact, it is easy to see that FF is just LBP with a specific forwards-backwards (FB) message passing schedule, instead of the more common parallel protocol (see Section B.4.2). The fixed points of LBP are the same, no matter what protocol is used. However, if there is not a unique fixed point, the algorithms may end up at different answers. They can also have different behavior in the short term. In particular, if the DBN is in fact a tree, then a single FB iteration ($2TD$ message computations) will result in the exact posteriors, whereas it requires T iterations of the decentralized protocol (each iteration computing $2TD$ messages in parallel) to reach the same result; hence the centralized algorithm is more efficient [PS91]. For loopy graphs, it is not clear which protocol is better; it depends on whether local or global information is more important for computing the posteriors. One can imagine performing iterations within a slice and also between slices in an interleaved manner.

BK is a special case of LBP

It turns out that BK is also equivalent to a single forwards-backwards pass of LBP, although on a slightly modified graph (see Figure 4.6). The key insight is that BK also uses a factored prior, but does an exact update step. To simulate the exact update, we can create two "mega nodes" that contain all the (hidden) forward

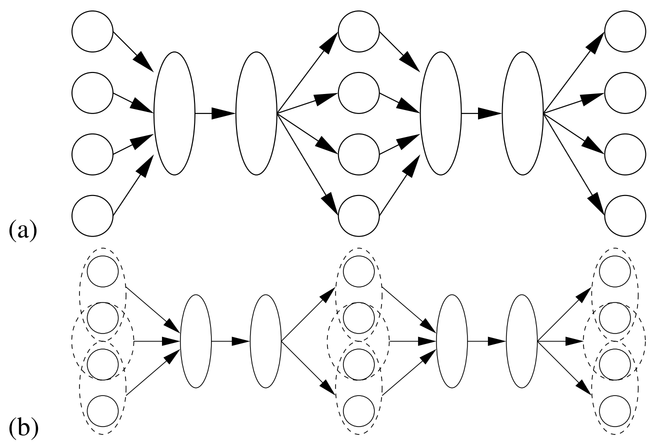


Figure 4.6: Illustration of the clustering process. (a) This is a modified version of a DBN with 4 persistent variables. The big "mega nodes" contain the joint distribution on the whole slice. We have omitted the non-persisitent variables (e.g., observations) for clarity. LBP applied to this graph is equivalent to BK. (b) This is like (a), except we have created overlapping clusters of size 2, for additional accuracy.

interface nodes in that slice. The messages coming into the first mega node are assumed independent; they are then multiplied together to form the (approximate) prior; a single message is then sent to the second mega node, corresponding to the exact update step; finally, the individual marginals are computed by marginalization, and the process is repeated. Of course, BK does not actually construct the mega nodes, and does the exact update using junction tree, but the two algorithms are functionally equivalent.

To simulate BK when the clusters contain more than one node, and possible overlap, we simply create mega-nodes, one per cluster, before applying the above transformation: see Figure 4.6 for an example.

Iteration helps

Since FF and BK are equivalent to one iteration of LBP, on the regular and clustered graphs respectively, we can improve on both of these algorithms by iterating more than once. This gives the algorithm the opportunity to "recover" from its incorrect independence assumptions. (This is also the intuition behind expectation propagation [Min01].) We will see in the Section 4.2.4 that even a small number of iterations can help dramatically.

4.2.4 Experimental comparison of FF, BK and LBP

In this section, we compare the BK algorithm with k iterations of LBP on the original graph, using the FB protocol ($k = 1$ iteration corresponds to FF). We used a coupled HMM model with $N = 10$ chains trained on some real freeway traffic data using exact EM [KM00]. We define the total L_1 error in the marginals as $\Delta_t = \sum_{i=1}^{N} \sum_{s=1}^{K} |P(X_t^i = s|y_{1:T}) - \hat{P}(X_t^i = s|y_{1:T})|$, where $P(\cdot)$ is the exact posterior and $\hat{P}(\cdot)$ is the approximate posterior. (We could have used KL divergence instead.) In Figure 4.7, we plot this against t for 1–4 iterations of LBP. Clearly, the posteriors are oscillating, and this happens on many sequences with this model. We therefore used the damping trick described in Section B.7.1; let m be the damping factor. It is clear from Figure 4.8 that damping helps considerably. The results are summarised in Figure 4.9, where we see that after a small number of iterations, LBP with $m = 0.1$ is doing better than BK. Other sequences give similar behavior.

To check that these results are not specific to this model/ data set, we also compared the algorithms on the water DBN shown in Figure 4.1. We generated observation sequences of length 100 from this model using random parameters and binary nodes, and then compared the marginal posteriors as a function of number of iterations and damping factor. The results for a typical sequence are shown in Figure 4.10. This time we see that there is no oscillation (this is typical for data sampled from a model with random parameters), and that as few as two iterations of LBP can outperform BK.

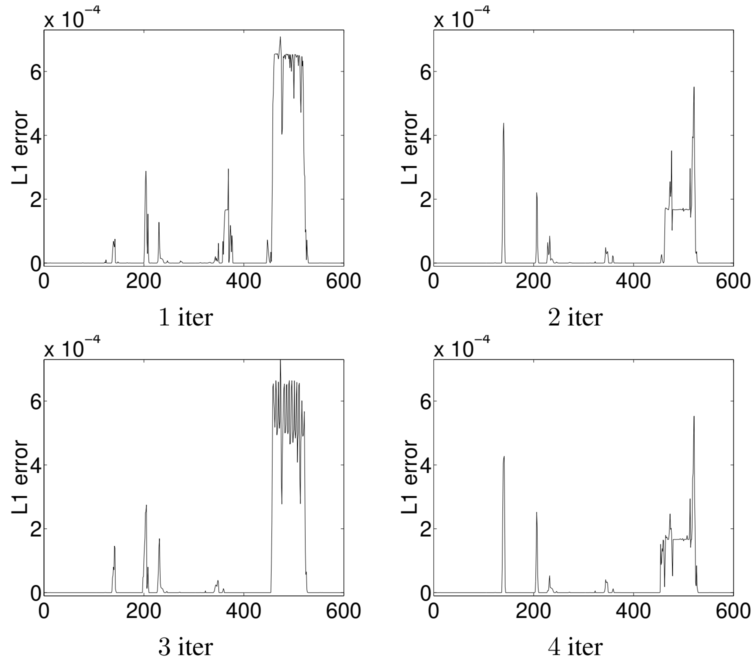


Figure 4.7: L_1 error on marginal posteriors vs. timeslice after iterations 1–4 of undamped LBP applied to the traffic CHMM. The L_1 error oscillates with a period of 2 (as seen by the similarity between the graphs for iterations 1/3 and 2/4); this implies that the underlying marginals are oscillating with the same period.

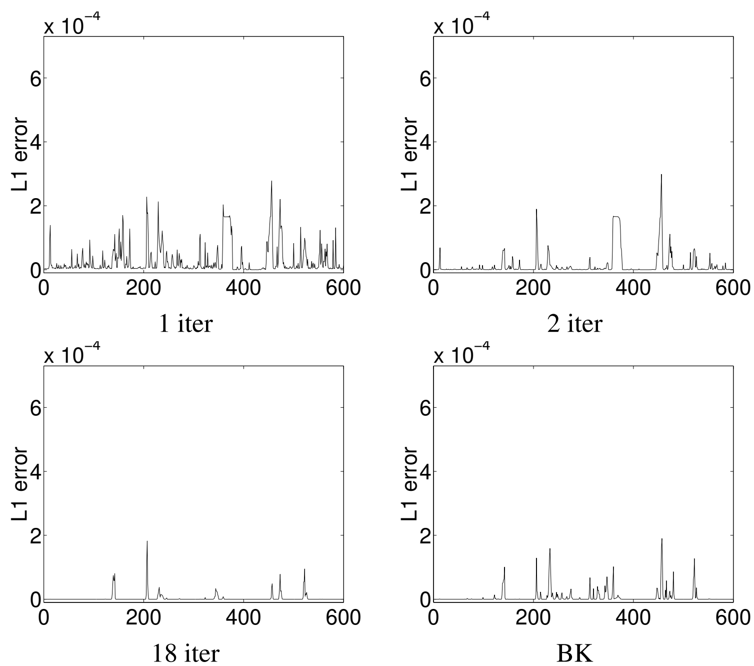


Figure 4.8: Iterations 1, 2, and 18 of LBP with damping factor $m = 0.1$, and after using 1 iteration of BK, on the traffic CHMM.

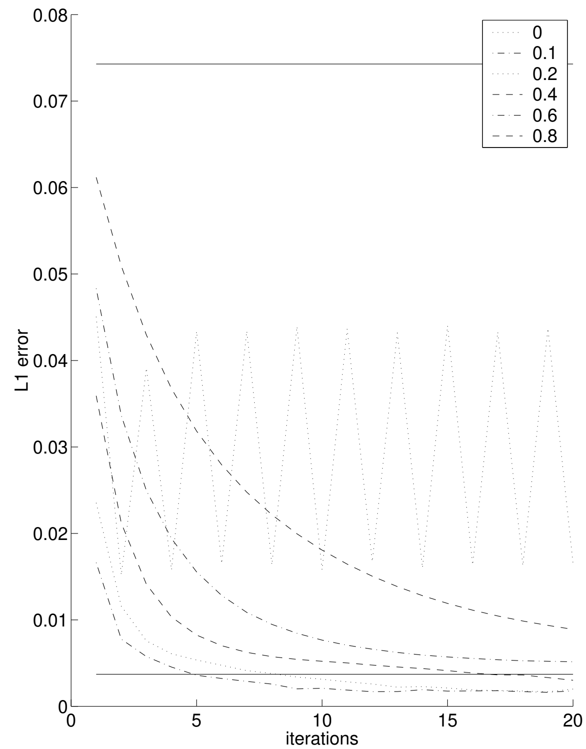


Figure 4.9: Results of applying LBP to the traffic CHMM with 10 chains. The lower solid horizontal line is the error incurred by BK. The oscillating line is the error incurred by LBP using damping factor $m = 0$; the lowest curve corresponds to $m = 0.1$ and the highest to $m = 0.8$. The upper horizontal line corresponds to not updating the messages at all ($m = 1$), and gives an indication of performance based on local evidence alone.

In addition to accuracy, it is also interesting to see how the algorithms compare in terms of speed. We therefore generated random data from CHMMs with $N = 1, 3, \ldots, 11$ chains, and computed the posteriors using the different methods. The running times are shown in Figure 4.11. It is clear that both BK and FF/LBP have a running time which is linear in N (for this CHMM model), but the constant factors of BK are much higher, due to the complexity of the algorithm, and in particular, the need to perform repeated marginalisations. This is also why BK is slower than exact inference for $N < 11$, even though it is asymptotically more efficient.

4.3 Switching KFMs

As we saw in Section 3.6.3, the belief state in a switching KFM at time t has size $O(K^t)$. We now discuss some approximations.

4.3.1 GPB (moment matching) algorithm

The exponential belief state in a switching KFM arises because CG potentials are not closed under marginalization (see Section B.5.2). A common approximation is to always represent the belief state using K^{n-1} Gaussians; this is sometimes called the GPB(n) (Generalized Pseudo Bayesian) approximation [SM80, TSM85, BSL93, Kim94, WH97], and is an instance of assumed density filtering (ADF). For example, in the GBP2 algorithm, the prior has K modes, each gets "passed" through the K different filters, and the resulting mixture of K^2 Gaussians is "collapsed" to the best mixture of K Gaussians, one Gaussian per value of S_t. This is shown schematically in Figure 4.14. A fast approximation to the GPB2 algorithm, which

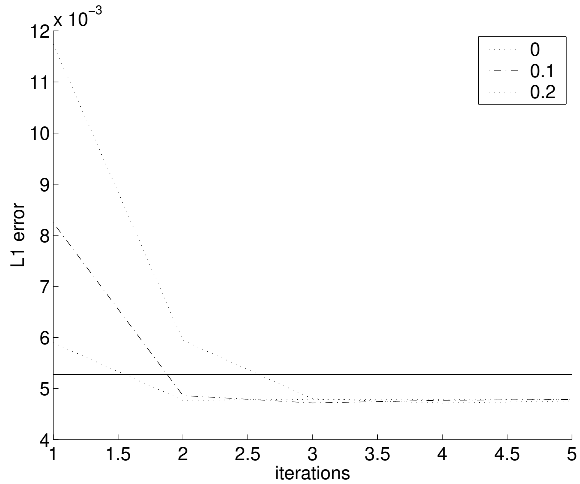


Figure 4.10: Same as Figure 4.9, but for the water network. The dotted lines, from bottom to top, represent $m = 0$, $m = 0.1$ and $m = 0.2$. The solid line represents BK.

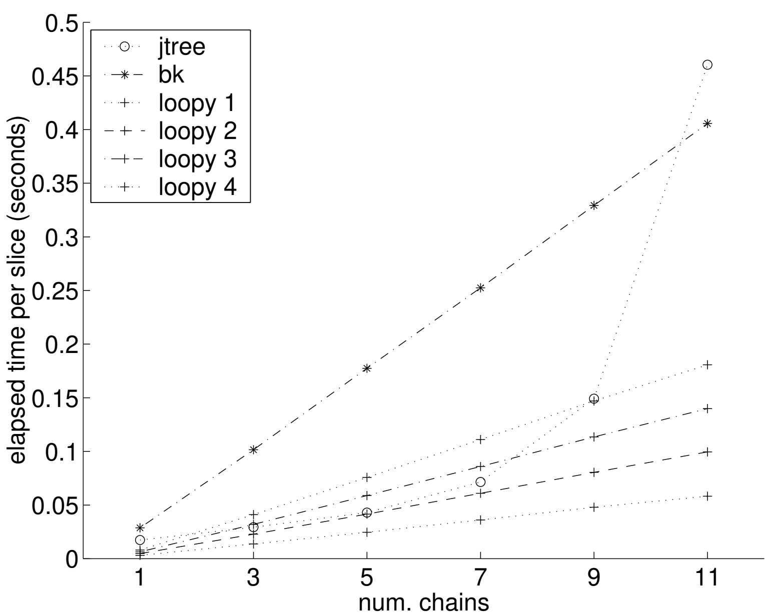


Figure 4.11: Running time on CHMMs as a function of the number of chains. The vertical axis is the total running time divided by the length of the sequence. The horizontal axis is the number of chains. The dotted curve is junction tree, the steep straight line is BK, and the shallow straight lines are LBP; "loopy k" means k iterations of LBP.

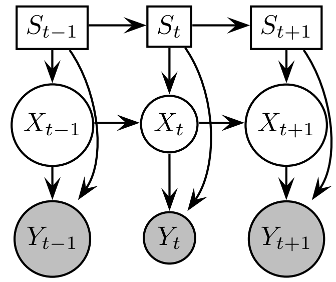


Figure 4.12: A switching Kalman filter model. Square nodes are discrete, round nodes are continuous.

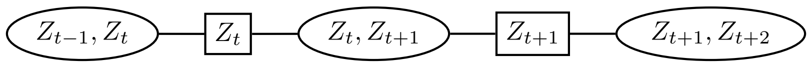


Figure 4.13: A jtree for 4 slices of the switching KFM in Figure 4.12. We use the shorthand $Z_t = (S_t, X_t)$, and have omitted the observation nodes Y_t for simplicity. It is possible to construct a jtree with smaller cliques, namely $(S_{t-1}, X_{t-1}, S_t) - [X_{t-1}, S_t] - (X_{t-1}, X_t, S_t) - [X_t, S_t]$, etc.

requires K Kalman filter updates per step rather than K^2, known as the interacting multiple models (IMM) approximation, is shown schematically in Figure 4.15 [BSL93].

The best Gaussian approximation to a mixture of K Gaussians (in the sense of minimizing KL distance) is obtained by moment matching, also known as weak marginalization (see Section B.5.2). In fact, the GPB2 procedure is identical to the (forwards pass of the) standard jtree algorithm applied to the jtree shown in Figure 4.13. (Note that this is a weak jtree: see Section B.3.6.)

One advantage of the jtree formulation is that it is simple to do backwards smoothing: we just apply the standard message passing scheme to the jtree in the backwards direction. In particular, we do not need to make the additional approximation used in [Kim94, Mur98]. Since the collapsing approximation is only applicable to messages represented in moment form (see Section B.5.1), we use the α/γ formulation instead of the α/β formulation (see Section 3.2), since β is a conditional likelihood that cannot be represented in moment form.

4.3.2 Viterbi approximation

Although we cannot compute $\sum_{S_{t-1}} \int_{X_{t-1}} \phi(S_{t-1}, X_{t-1}, S_t, X_t)$ exactly, and have to use moment matching (see Section B.5.2), we *can* compute $\max_{S_{t-1}} \int_{X_{t-1}} \phi(S_{t-1}, X_{t-1}, S_t, X_t)$: for each S_t, we simply pick the most likely S_{t-1}, and integrate out the corresponding X_{t-1}. In general, one can maintain H hypotheses in the prior, update each of them to give KH posterior modes, and then select the H most likely. This classical idea can be extended to the smoothing context as follows [PRCM99]: compute $P(X_{t-1}, X_t, S_{t-1}, S_t | y_{1:t}, s^*_{1:t-2})$ by doing a forwards Viterbi pass in the jtree, where $s^*_{1:t} = \arg\max_{s_{1:t}} P(s_{1:t}|y_{1:t})$, and then compute $P(X_{t-1}, X_t | s^*_{1:T}, y_{1:T})$ using the RTS smoother (or the jtree algorithm) with parameters specified by $s^*_{1:T}$. (Note we keep $H = K$ hypotheses in the belief state at every step.)

Note that this is not the same as computing $P(X_{t-1}, S_{t-1}, X_t, S_t | y_{1:T})$ which is what we would really like in order to get the exact expected sufficient statistics needed for EM. For instance, the Viterbi approximation says $P(X_t, S_t = i | y_{1:T}) = 0$ for all $i \neq s^*_t$. However, if the "regimes" are easy to segment, the responsibilities $P(S_t|y_{1:T})$ are approximately delta-functions, so the Viterbi approximation is a good one. (This is the basis of Viterbi training, which is widely used in speech recognition.) If the Viterbi approximation does not hold, one can obviously sample $S_{1:T}$ (see Section 5.4.1).

4.3.3 Expectation propagation

Consider the jtree in Figure 4.13. If we could marginalize CG potentials in closed form, a single forwards-backwards pass would suffice to do exact smoothing; furthermore, since this tree is a chain, the α and β

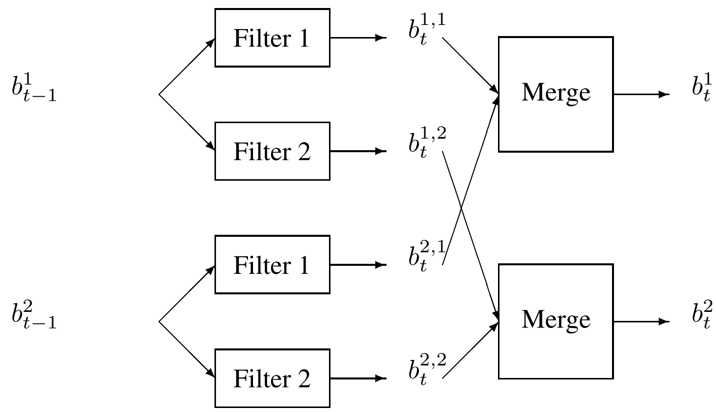


Figure 4.14: The GPB2 algorithm for the case of a binary switch. $b_{t-1}^i = P(X_{t-1}, S_{t-1} = i|y_{1:t-1})$ is the prior, $b_t^{i,j} = P(X_t, S_{t-1} = i, S_t = y|y_{1:t})$, and $b_t^j = P(X_t S_t = y|y_{1:t})$ is the approximate posterior. The filter box implements the Kalman filter equations (see Section 3.6.1). The merge box implements the weak marginalization/ moment matching equations (see Section B.5.2).

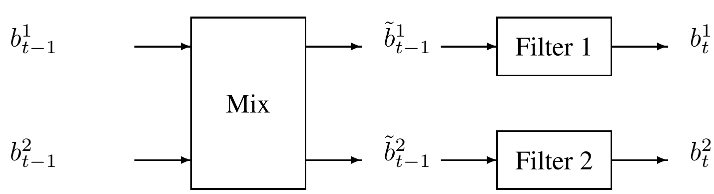


Figure 4.15: The IMM (interacting multiple models) algorithm for the case of a binary switch.

for $t = 1 : T$
$\alpha_t(z_t) = 1$
$\beta_t(z_t) = 1$
if $t == 1$
$\psi_1(z_1) = P(y_1|z_1)P(z_1)$
else
$\psi_t(z_{t-1}, z_t) = P(y_t|z_t)P(z_t|z_{t-1})$
while not converged
for $t = 1 : T$
if $t == 1$
$\hat{p}(z_1) = \psi_1(z_1)\beta_1(z_1)$
else
$\hat{p}(z_{t-1}, z_t) = \alpha_{t-1}(z_{t-1})\psi_t(z_{t-1}, z_t)\beta_t(z_t)$
$q(z_t) = \text{Collapse}_{s_{t-1}} \int_{x_{t-1}} \hat{p}(z_{t-1}, z_t)$
$\alpha_t(z_t) = \frac{q(z_t)}{\beta_t(z_t)}$
for $t = T : 1$
$\hat{p}(z_{t-1}, z_t) = \alpha_{t-1}(z_{t-1})\psi_t(z_{t-1}, z_t)\beta_t(z_t)$
$q(z_{t-1}) = \text{Collapse}_{s_t} \int_{x_t} \hat{p}(z_{t-1}, z_t)$
$\beta_{t-1}(z_{t-1}) = \frac{q(z_{t-1})}{\alpha_{t-1}(z_{t-1})}$

Figure 4.16: Pseudo-code for EP applied to a switching KFM, based on [HZ02]. $Z_t = (X_t, S_t)$. Collapse refers to weak marginalization.

messages would not interact (see Section B.4.6). However, since the messages only contain the first two moments, rather than the full belief state, multiple iterations may help improve accuracy; this is the basic idea behind expectation propagation (EP) [Min01]. The general algorithm is explained in Section B.7.2; a special case of it, applied to the switching KFM, is shown in Figure 4.16 (based on [HZ02]). This could be converted to an online algorithm if we used a fixed lag window.

Note that the BK algorithm is also an instance of EP, where the messages are factored. In this case, Z_t refers to all the nodes in slice t of the DBN, and the computation of $\alpha_{t-1}(z_{t-1})\psi_t(z_{t-1}, z_t)\beta_t(z_t)$ is done using jtree. The collapse procedure simply means computing the desired product of marginals on the nodes in the forward interface.

4.3.4 Variational methods

A structured variational approximation for a special case of a switching KFM, in which the discrete variable selects which hidden variable gets observed (see Section 2.4.6), was proposed in [GH98]. This decoupled the problem into parallel chains, very much like the variational approximation for factorial HMMs [GJ97]. The variational parameter for the discrete chain plays the role of the conditional likelihood term $P(Y_t|S_t = i)$ in a regular HMM; the variational parameters for the continuous chains represent the responsibility of that chain for the evidence: this weights the observations by changing the observation covariance matrix.

A structured variational approximation for the arguably more useful switching KFM in which the discrete variable changes the dynamics (see Section 2.4.3) was sketched in [PRM00]. Again, the proposed approximating structure consists of two chains, one discrete and one continuous. The discrete variational parameters are the conditional likelihoods, as before, but for the continuous chain, it is the system covariance and transition matrices which represent the variational parameters (since they summarize influence from the discrete nodes above).

4.4 Non-linear/ non-Gaussian models

4.4.1 Filtering

The standard way to do filtering in non-linear models is to do a local linearization around $x_{t|t-1}$, and then apply the Kalman filter to the resulting model; this is called the (iterated) Extended Kalman filter (EKF). An alternative to the EKF is the unscented Kalman filter (see [WdM01] for a review), which avoids the need to compute the Jacobian, and generally approximates the covariance more accurately.

For non-Gaussian noise models, other techniques can be used. For example, suppose $P(Y_t|X_t)$ is a generalized linear model [MN83] e.g., Poisson or log-normal. (The overall model is called a dynamic generalized linear model.) One can then use moment matching and approximate the posterior $P(X_t|y_{1:t})$ as Gaussian [WH97]. It is straightforward to generalize this approximation to the DBN case [SSG99].

In all of the above algorithms, the posterior is being approximated by a Gaussian. If the true posterior is highly multimodal, it is probably better to use particle filtering, which we discuss in Chapter 5.

4.4.2 Sequential parameter estimation

Online parameter estimation is equivalent to sequential Bayesian updating (see Section 6.1.1). If the parameters have conjugate priors and the model is fully observed (see Appendix C), this can be done in closed form (see e.g., [CDLS99, sec 9.4] for the case of multinomial CPDs with Dirichlet priors, or [WH97] for various continuous densities).

Difficulties arise if the priors are not conjugate and/or there is partial observability. Various classical techniques for collapsing the mixture of parameters that result from partial observability are discussed in [CDLS99, sec. 9.7]. Alternatively, we can use sequential variational Bayes [HT01]. If the goal is to do simultaneous state and parameter estimation, we can run two EKFs in parallel [WN01]. Stochastic solutions, based on particle filtering, are discussed in Section 5.2. Methods based on online EM are discussed in Section C.4.5.

4.4.3 Smoothing

In an offline setting, one can afford more expensive techniques. For state estimation one can use the extended Kalman smoother [GR98], and for parameter estimation, one can use EM (see Section C.4.2). Stochastic solutions, mostly based on Gibbs sampling, are discussed in Section 5.4. See [DK00] for a recent review of non-linear/ non-Gaussian time-series analysis techniques.

Chapter 5

Approximate inference in DBNs: stochastic algorithms

5.1 Introduction

Stochastic (sampling) algorithms come in two flavors: offline and online. Offline methods are usually based on importance sampling (likelihood weighting) or Monte Carlo Markov Chain (MCMC) [GRS96], which includes Gibbs sampling and simulated annealing as special cases. Online methods usually use particle filtering (PF), also known as sequential importance sampling (SIS), sequential Monte Carlo, the bootstrap filter [Gor93], the condensation algorithm [IB96], survival of the fittest [KKR95], etc.: see [AMGC02, LC98, Dou98] for good tutorials, and [DdFG01] for a collection of recent papers. DBNs specify $P(X_t|X_{t-1})$ in a compact form, and hence can be "plugged in" to any PF algorithm. I discuss this in Section 5.2.

Sampling algorithms have several advantages over deterministic approximation algorithms: they are easy to implement, they work on almost any kind of model (all kinds of CPDs can be mixed together, the state-space can have variable size, the model structure can change), they can convert heuristics into provably correct algorithms by using them as proposal distributions, and, in the limit of an infinite number of samples (in "asymptotia"), they are guaranteed to give the exact answer.

The main disadvantage of sampling algorithms is speed: they are often significantly slower than deterministic methods, often making them unsuitable for large models and/or large data sets. In this chapter, I discuss how to get the best of both worlds by combining exact and stochastic inference. The basic idea is to "integrate out" some of the variables using exact inference, and apply sampling to the remaining ones; this is called Rao-Blackwellisation [CR96]. When combined with particle filtering, the resulting algorithm is called Rao-Blackwellised particle filtering (RBPF). In Section 5.3.1, I explain how RBPF can be applied to a switching KFM [AK77, CL00, DGK99]. The first novel contribution appears in Section 5.3.2, where I apply RBPF to the SLAM (simultaneous localization and mapping) problem. In Section 5.3.3, I discuss how to apply RBPF to arbitrary DBNs. Finally, in Section 5.4.1, I briefly discuss how to combine Rao-Blackwellisation and MCMC.

5.2 Particle filtering

The basic idea behind particle filtering (PF) is to approximate the belief state by a set of weighted particles or samples:

$$P(X_t|y_{1:t}) \approx \sum_{i=1}^{N_s} w_t^i \delta(X_t, X_t^i)$$

(In this chapter, X_t^i means the i'th sample of X_t, and $X_t(i)$ means the i'th component of X_t.) Given a prior of this form, we can compute the posterior using importance sampling. In importance sampling, we assume the target distribution, $\pi(x)$, is hard to sample from; instead, we sample from a proposal or importance distribution $q(x)$, and weight the sample according to $w^i \propto \pi(x)/q(x)$. (After we have finished sampling, we can normalize all the weights so $\sum_i w^i = 1$). We can use this to sample paths with weights

$$w_t^i \propto \frac{P(x_{1:t}^i|y_{1:t})}{q(x_{1:t}^i|y_{1:t})}$$

$P(x_{1:t}|y_{1:t})$ can be computed recursively using Bayes rule. Typically we will want the proposal distribution to be recursive also, i.e., $q(x_{1:t}|y_{1:t}) = q(x_t|x_{1:t-1}, y_{1:t})q(x_{1:t-1}|y_{1:t-1})$. In this case we have

$$\begin{aligned} w_t^i &\propto \frac{P(y_t|x_t^i)P(x_t^i|x_{t-1}^i)P(x_{1:t-1}^i|y_{1:t-1})}{q(x_t^i|x_{1:t-1}^i, y_{1:t})q(x_{1:t-1}^i|y_{1:t-1})} \\ &= \frac{P(y_t|x_t^i)P(x_t^i|x_{t-1}^i))}{q(x_t^i|x_{1:t-1}^i, y_{1:t})} w_{t-1}^i \\ &\stackrel{\text{def}}{=} \hat{w}_t^i \times w_{t-1}^i \end{aligned}$$

where we have defined $\hat{w}_t^i$ to be the incremental weight.

For filtering, we only care about $P(X_t|y_{1:t})$, as opposed to the whole trajectory, so we use the following proposal, $q(x_t|x_{1:t-1}^i, y_{1:t}) = q(x_t|x_{t-1}^i, y_t)$, so we only need to store x_{t-1}^i instead of the whole trajectory. In this case the weights simplify to

$$\hat{w}_t^i = \frac{P(y_t|x_t^i)P(x_t^i|x_{t-1}^i)}{q(x_t^i|x_{t-1}^i, y_t)} \tag{5.1}$$

The most common proposal is to sample from the prior: $q(x_t|x_{t-1}^i, y_t) = P(x_t|x_{t-1}^i)$. In this case, the weights simplify to $\hat{w}_t^i = P(y_t|x_t^i)$. For predicting the future, sampling from the prior is adequate, since there is no evidence. This technique can be used e.g., to stochastically evaluate policies for (PO)MDPs [KMN99].) But for monitoring/ filtering, it is not very efficient, since it amounts to "guess until you hit". For example, if the transitions are highly stochastic, sampling from the prior will result in particles being proposed all over the space; if the observations are highly informative, most of the particles will get "killed off" (i.e., assigned low weight). In such a case, it makes more sense to first look at the evidence, y_t, and then propose:

$$q(x_t|x_{t-1}^i, y_t) = P(x_t|x_{t-1}^i, y_t) \propto P(y_t|x_t)P(x_t|x_{t-1}^i)$$

In fact, one can prove this is the optimal proposal distribution, in the sense of minimizing the variance of the weights (a balanced distribution being more economical, since particles with low weight are "wasted"). Unfortunately, it is often hard to sample from this distribution, and to compute the weights, which are given by the normalizing constant of the optimal proposal:

$$\hat{w}_t^i = P(y_t|x_{t-1}^i) = \int_{x_t} P(y_t|x_t)P(x_t|x_{t-1}^i)$$

In Section 5.2.1, we will discuss when it is tractable to use the optimal proposal distribution.

Applying importance sampling in this way is known as sequential importance sampling (SIS). A well known problem with SIS is that the number of particles with non-zero weight rapidly goes to zero, even if we use the optimal proposal distribution (this is called particle "impoverishment"). An estimate of the "effective" number of samples is given by

$$N_{eff} = \frac{1}{\sum_{i=1}^{N_s}(w_t^i)^2} \tag{5.2}$$

If this drops below some threshold, we can sample with replacement from the current belief state. Essentially this throws out particles with low weight and replicates those with high weight (hence the term "survival of the fittest"). This is called resampling, and can be done in $O(N_s)$ time. After resampling, the weights are reset to the uniform distribution: the past weights are reflected in the frequency with which particles are sampled, and do not need to be kept. Particle filtering is just sequential importance sampling with resampling (SISR). The resampling step was the key innovation in the '90s; SIS itself has been around since at least the '50s. The overall algorithm is sketched in Figure 5.1.

function $[\{x_t^i, w_t^i\}_{i=1}^{N_s}] = \text{PF}(\{x_{t-1}^i, w_{t-1}^i\}_{i=1}^{N_s}, y_t)$
for $i = 1 : N_s$
  Sample $x_t^i \sim q(\cdot|x_{t-1}^i, y_t)$
  Compute $\hat{w}_t^i$ from Equation 5.1
  $w_t^i = \hat{w}_t^i \times w_{t-1}^i$
Compute $w_t = \sum_{i=1}^{N_s} w_t^i$
Normalize $w_t^i := w_t^i / w_t$
Compute N_{eff} from Equation 5.2
if $N_{eff} <$ threshold
  $\pi = \text{resample}(\{w_t^i\}_{i=1}^{N_s})$
  $x_t^{\cdot} = x_t^{\pi}$
  $w_t^i = 1/N_s$

Figure 5.1: Pseudo-code for a generic particle filter. The resample step samples indices with replacement according to their weight; the resulting set of sampled indices is called π. The line $x_t^{\cdot} = x_t^{\pi}$ simply duplicates or removes particles according to the chosen indices.

function $[x_t^i, \hat{w}_t^i] = \text{LW}(x_{t-1}^i, y_t)$
$\hat{w}_t^i = 1$
$x_t^i =$ empty vector of length N
for each node i in topological order
  Let u be the value of $\text{Pa}(X_t^i)$ in (x_{t-1}^i, x_t^i)
  If X_t^i not in y_t
    Sample $x_t^i \sim P(X_t^i|\text{Pa}(X_t^i) = u)$
  else
    $x_t^i =$ the value of X_t^i in y_t
    $\hat{w}_t^i = \hat{w}_t^i \times P(X_t^i = x_t^i|\text{Pa}(X_t^i) = u)$

Figure 5.2: Pseudo-code for likelihood weighting.

Although resampling kills off unlikely particles, it also reduces the diversity of the population (which is why we don't do it at every time step; if we did, then $w_t^i = \hat{w}_t^i$). This a particular severe problem is the system is highly deterministic (e.g., if the state space contains static parameters). A simple solution is to apply a kernel around each particle and then resample from the kernel. An alternative is to use an MCMC smoothing step; a particularly succesful version of this is the resample-move algorithm [BG01].

5.2.1 Particle filtering for DBNs

To apply PF to a DBN, we use the likelihood weighting (LW) routine [FC89, SP89] in Figure 5.2 to sample x_t^i and compute $\hat{w}_t^i$. The proposal distribution that LW corresponds to depends on which nodes of the DBN are observed. In the simplest case of an HMM, where the observation is at a leaf node, LW samples from the prior, $P(X_t^i|x_{t-1}^i)$, and the computes the weight as $w = P(y_t|x_t^i)$. (We discuss how to improve this below. See also [CD00].)

In general, some of the evidence might occur at arbitrary locations within the DBN slice. In this case, the proposal is $q(x_t, y_t) = \prod_j P(x_t(j)|\text{Pa}(X_t(j)))$, and the weight is $w(x_t, y_t) = \prod_j P(y_t(j)|\text{Pa}(Y_t(j)))$, where $x_t(j)$ is the (value of the) j'th hidden node at time t, and $y_t(j)$ is the (value of the) j'th observed node at time t, and the parents of both $X_t(j)$ and $Y_t(j)$ may contain evidence. This is consistent, since (as observed in [RN02])

$$P(x_t, y_t) = \prod_j P(x_t(j)|\text{Pa}(X_t(j))) \times \prod_j P(y_t(j)|\text{Pa}(Y_t(j))) = q(x_t, y_t)w(x_t, y_t)$$

Optimal proposal distribution

Since the evidence usually occurs at the leaves, likelihood weighting effectively samples from the prior, without looking at the evidence. A general way to take the evidence into account while sampling, suggested in [FC89, KKR95], is called "evidence reversal". This means applying the rules of "arc reversal" [Sha86] until all the evidence nodes become parents instead of leaves. To reverse an arc from $X \to Y$, we must add Y's unique parents, Y_p, to X, and add X's unique parents, X_p, to Y (both nodes may also share common parents, C): see Figure 5.3. The CPDs in the new network are given by

$$\begin{aligned} P(Y|X_p, C, Y_p) &= \sum_x P(Y|C, Y_p)P(x|X_p, C) \\ P(X|X_p, C, Y_p) &= \frac{P(Y|X, C, Y_p)P(X|X_p, C)}{P(Y|X_p, C, Y_p)} \end{aligned}$$

Note that X_p, Y_p and C could represent sets of variables. Hence the new CPDs could be much larger than before the arc reversal. (One way to ameliorate this affect, for tree-structured CPDs, is discussed in [CB97].) Of course, if there are multiple evidence nodes, not all of the arcs have to be reversed. In the case of DBNs, the operation is shown in Figure 5.4. The new CPDs are

$$\begin{aligned} P(Y_t|X_{t-1}) &= \sum_{x_t} P(Y_t|x_t)P(x_t|X_{t-1}) \\ P(X_t|X_{t-1}, Y_t) &= \frac{P(Y_t|X_t)P(X_t|X_{t-1})}{P(Y_t|X_{t-1})} \end{aligned}$$

Arc reversal was proposed by [Sha86] as a general means of inference in Bayesian networks. Since then, the junction tree (jtree) algorithm has come to dominate, since it is more efficient. It is possible to efficiently sample from $P(X|E)$ by first building a jtree, collecting evidence to the root, and then, in the distribute phase, drawing a random sample from $P(X_{C_i \setminus S_i}|x_{S_i}, E)$ for each clique C_i, where S_i is the separator nearer to the root [Daw92]. This is the optimal method.

Unfortunately, for continuous valued variables (when PF is most useful), it is not always possible to compute the optimal proposal distribution, because the new CPDs required by arc reversal, or the potentials required by jtree, cannot be computed. An important exception is when the observation model, $P(Y_t|X_t)$, is (conditionally) linear-Gaussian, and the process noise is Gaussian (although the dynamics can be non-linear), i.e.,

$$\begin{aligned} P(X_t|x_{t-1}^i) &= \mathcal{N}(X_t; f_t(x_{t-1}^i), Q_t) \\ P(Y_t|X_t) &= \mathcal{N}(y_t; H_t X_t, R_t) \end{aligned}$$

In this case, one can use the standard Kalman filter rules to show that [AMGC02]

$$\begin{aligned} P(X_t|x_{t-1}^i, y_t) &= \mathcal{N}(X_t; m_t, \Sigma_t) \\ \hat{w}_t^i = P(y_t|x_{t-1}^i) &= \mathcal{N}(y_t; H_t f_t(x_{t-1}), Q_t + H_t R_t H_t') \end{aligned}$$

where

$$\begin{aligned} \Sigma_t^{-1} &= Q_t^{-1} + H_t R_t^{-1} H_t \\ m_t &= \Sigma_t \left(Q_t^{-1} f_t(x_{t-1}^i) + H_t' R_t^{-1} y_t \right) \end{aligned}$$

If the model does not satisfy these requirements, one can still use a Gaussian approximation of the form $q(X_t|x_{t-1}^i, y_t)$ constructed e.g., using the unscented transform [vdMDdFW00]. (This is the sense in which any heuristic can be converted into an optimal algorithm.) If the process noise is non-Gaussian, but the observation model is (conditional) linear-Gaussian, once can propose from the likelihood and weight by the transition prior. (This observation was first made by Nando de Freitas, and has been exploited in [FTBD01].)

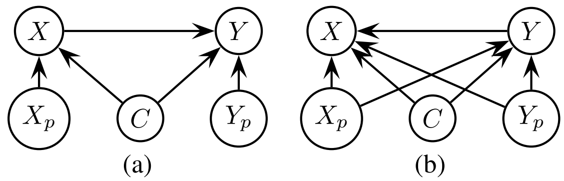


Figure 5.3: Arc reversal. (a) The original network. (b) The network after reversing the $X \to Y$ arc. The modified network encodes the same probability distribution as the original network.

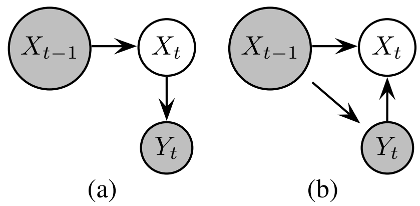


Figure 5.4: A DBN (a) before and (b) after evidence reversal.

Smoothing discrete state-spaces

When applying PF to (low dimensional) continuous state-spaces, it is easy to place a (Gaussian) kernel around each particle before resampling, to prevent "particle collapse". However, most DBNs that have been studied have discrete state-spaces. In this case, we can use the smoothing technique proposed in [KL01]. Let $W_t = \sum_{i=1}^{N_s} w_t^i$. We add a certain fraction, α, of W_t to all entries in the state-space that are consistent with the evidence (i.e., which give it non-zero likelihood), and then renormalize. (This is like using a uniform Dirichlet prior.) That is, the smoothed approximate belief state is

$$\hat{P}(x|y_{1:t}) = \frac{\alpha + \sum_{i:x_t^i = x} w_t^i}{Z}$$

if x is consistent with y_t, and $\hat{P}(x|y_{1:t}) = 0$ otherwise. The sum is over all particles that are equal to x; if there are no such particles, the numerator has value α. The normalizing constant is $Z = W_t + \alpha M$, where M is the total number of states consistent with y_t. (We discuss how to compute M below.) We can sample from this smoothed belief state as follows. With probability W_t/Z, we select a particle as usual (with probability w_t^i), otherwise we select a state which is consistent with the evidence uniformly at random.

Computing M is equivalent to counting the number of satisfying assignments, which in general is #-P hard (worse than NP-hard). If all CPDs are stochastic, we can compute M using techniques similar to Bayes net inference. Unfortunately, the cost of exactly computing M may be as high as doing exact inference. A quick and dirty solution is to add probability mass of α to all states, whether or not they are consistent with the evidence.

Combining PF with BK

[NPP02] suggested combining particle filtering with the Boyen-Koller algorithm (see Section 4.2.1), i.e., approximating the belief state by

$$\hat{P}(X_t|y_{1:t}) \approx \prod_{c=1}^{C} \frac{1}{N_c} \sum_{i=1}^{N_c} \delta(X_{t,c}, x_{t,c}^i)$$

where C is the number of clusters, and N_c is the number of particles in each cluster (assumed uniformly weighted). By applying PF to a smaller state-space (each cluster), they reduce the variance, at a cost of increasing the bias by using a factored representation. They show that this method outperforms standard PF (sampling from the prior) on some small, discrete DBNs. However, they do not compare with BK. Furthermore, it might be difficult to extend the method to work with continuous state-spaces (when PF is most useful), since the two proposed methods of propagating the factored particles — using jtree and using an equi-join operator — only work well with discrete variables.

5.3 Rao-Blackwellised Particle Filtering (RBPF)

The Rao-Blackwell theorem shows how to improve upon any given estimator under every convex loss function. At its core is the following well-known identity:

$$\text{Var}[\tau(X,R)] = \text{Var}[E(\tau(X,R)|R)] + E[\text{Var}(\tau(X,R)|R)]$$

where $\tau(X,R)$ is some estimator of X and R. Hence $\tau'(X,R) = E(\tau(X,R)|R)$ is a lower variance estimator. So if we can sample R and compute this conditional expectation analytically, we will need less samples (for a given accuracy). Of course, less samples does not necessarily mean less time: it depends on whether we can compute the conditional expectation efficiently or not. In the following, we will always assume this is the case.

5.3.1 RBPF for switching KFMs

Consider the switching KFM in Figure 4.12. If we knew $S_{1:t}$, we could compute $P(X_t|y_{1:t}, s_{1:t})$ exactly using a Kalman filter. Hence we can apply particle filtering to S_t, which lives in a small, discrete space, and integrate out X_t, which lives in a (potentially) large, continuous space. This idea was first proposed in [AK77] (without the resampling step) and independently in [CL00, DGK99] using the modern PF method.

Algorithmically, what this means is that each particle i contains a sampled value for S_t, which implicitely represents a whole trajectory, $s^i_{1:t}$, plus the sufficient statistics for the Kalman filter conditioned on this trajectory, $\mu^i_t = E[X_t|y_{1:t}, s^i_{1:t}]$ and $\Sigma^i_t = \text{Cov}[X_t|y_{1:t}, s^i_{1:t}]$. If we propose from the prior, $q(S_t|s^i_{t-1}) = P(S_t|s^i_{t-1})$, we can compute the weight using the one-step-ahead prediction likelihood:

$$\begin{aligned}
\hat{w}^i_t &= \frac{P(y_t|s^i_t)P(s^i_t|s^i_{t-1})}{P(s^i_t|s^i_{t-1})} \\
&= \int_{x_t} P(y_t|x_t, s^i_t)P(x_t|s^i_{1:t}, y_{1:t-1}) \\
&= P(y_t|y_{1:t-1}, s^i_{1:t}) \\
&= \mathcal{N}(y_t; C_s A_s \mu^i_{t-1}, C_s(A_s \Sigma^i_{t-1} A'_s + Q_s)C'_s + R_s)
\end{aligned}$$

where $s = s^i_t$. This term is just a byproduct of applying the Kalman filtering equations to the sufficient statistics $\mu^i_{t-1}, \Sigma^i_{t-1}$ with the parameter set $\theta_s = (A_s, C_s, Q_s, R_s)$. The overall algorithm is shown in Figure 5.5. Note that we can use this algorithm even if S_t is not discrete.

If the number of values of S_t is sufficiently small, and the ratio of transition noise (for S_t) to observation noise sufficiently high, it might be beneficial to use the optimal proposal distribution, which is given by

$$P(S_t = s|s^i_{1:t-1}, y_{1:t}) \propto P(y_t|S_t = s, s^i_{1:t-1}, y_{1:t-1})P(S_t = s|s^i_{t-1})$$

In this case, the incremental weight is just the normalizing constant for the optimal proposal:

$$\hat{w}^i_t = \sum_s P(y_t|S_t = s, s^i_{1:t-1}, y_{1:t-1})P(S_t = s|s^i_{t-1})$$

The modified algorithm is shown in Figure 5.6. This is more expensive than sampling from the prior, since for each particle i, we must loop over all states s. However, this may require fewer particles, making it faster overall.

Fault diagnosis

In principle, it is straightforward to apply RBPF to problems in fault diagnosis, such as the DBN in Figure 2.33. Unfortunately, in that model, there are a large number of discrete variables, so it is too expensive to

```
function [{s_t^i, μ_t^i, Σ_t^i, w_t^i}] = RBPF-SKFM-prior({s_{t-1}^i, μ_{t-1}^i, Σ_{t-1}^i, w_{t-1}^i}, y_t)
for i = 1 : N_s
    Sample s_t^i ~ P(S_t | s_{t-1}^i)
    (μ_t^i, Σ_t^i, ŵ_t^i) = KF(μ_{t-1}^i, Σ_{t-1}^i, y_t, θ_{s_t^i})
    w_t^i = ŵ_t^i × w_{t-1}^i
Compute w_t = Σ_{i=1}^{N_s} w_t^i
Normalize w_t^i := w_t^i / w_t
Compute N_eff from Equation 5.2
if N_eff < threshold
  π = resample({w_t^i}_{i=1}^{N_s})
  s_t^· = s_t^π, μ_t^· = μ_t^π, Σ_t^· = Σ_t^π
  w_t^i = 1/N_s
```

Figure 5.5: Pseudo-code for RBPF applied to a switching KFM, where we sample from the prior.

```
function [{s_t^i, μ_t^i, Σ_t^i, w_t^i}] = RBPF-SKFM-opt({s_{t-1}^i, μ_{t-1}^i, Σ_{t-1}^i, w_{t-1}^i}, y_t)
for i = 1 : N_s
    for each s
        (μ^s, Σ^s, L(s)) = KF(μ_{t-1}^i, Σ_{t-1}^i, y_t, θ_s)
        q(s) = L(s) × P(S_t = s | s_{t-1}^i)
    ŵ^s = Σ_s q(s)
    Normalize q(s) := q(s) / ŵ^s
    Sample s ~ q(·)
    s_t^i = s, μ_t^i = μ^s, Σ_t^i = Σ^s, ŵ_t^i = ŵ^s
    w_t^i = ŵ_t^i × w_{t-1}^i
...
```

Figure 5.6: Pseudo-code for RBPF applied to a switching KFM, where we sample from the optimal proposal. The third return argument from the KF routine is $L(s) = P(y_t|S_t = s, s^i_{1:t-1}, y_{1:t-1})$. $\ldots$ means the code continues as in Figure 5.5.

use the optimal proposal. Obviously one could propose from the prior, but there is an alternative, deterministic heuristic that works well in this case [LP01], namely: enumerate the discretes in order according to their prior probability. This exploits the fact that it is more likely that there is a single fault than that there are two faults, etc. (Since this is a non-linear model, even conditioned on the discretes, one must apply the EKF or UPF, instead of the KF, to each particle.)

Enumerating in a priori order is not applicable if the discrete variables represented cardinal (as opposed to ordinal) quantities, as occurs e.g., in data association problems. Another problem with this scheme is that although faults may have low (prior) probability, they may have a high cost. One should therefore take the loss function into account when proposing samples [TLV01]. An alternative technique in [CT02] tries to identify the most likely assignment to the discretes by allocating a different number of samples to each discrete hypothesis in an adaptive way; this is similar to Hoeffding races [MM93].

Stochastic vs deterministic approximations

RBPF for switching KFMs is very similar to classical multiple hypothesis tracking (MHT). It is not clear which is better. The argument between stochastic vs deterministic approximations often boils down to matters of taste.

5.3.2 RBPF for simultaneous localisation and mapping (SLAM)

When a mobile robot moves through the world, it is important that it knows where it is; this is called the localization problem. It is not always possible to use GPS to solve this problem (e.g., in indoor environments). However, by comparing what it sees with a map of the world, the robot can infer its location (not always uniquely). There are two popular representations for maps: metric (grid-based) and topological. We restrict our attention to the first.

A metric map is usually represented as an occupancy grid [ME85], which is a binary matrix where 1's represent cells which contain obstacles and 0's represent free space. Since we have discretized space, we can think of the map (assumed to be static) as defining the observation (and transition) matrix of an HMM. Hence the map can be learned using EM [TBF98]. (EM provides a solution to the following chicken-and-egg problem: to localize, the robot must know the map, but the learn the map, the robot must know its location.) This learning is usually done offline, after a human has joysticked the robot through the environment.

Localization simply means inferring $P(L_t|y_{1:t}, u_{1:t})$, where L_t is the location of the robot, $y_{1:t}$ are the sensor measurements (typically from a sonar or laser range finder, plus odometry information from the wheels) and $u_{1:t}$ are the control signals sent to the motors. (Henceforth we shall use $z_{1:t} = (y_{1:t}, u_{1:t})$ for brevity.) In principle, this inference problem can be solved exactly by using the forwards algorithm for HMMs. In practice, this is too slow, since the state-space, though discrete, is very large (millions of cells). However, one can apply particle filters in a straightforward way [FTBD01].

Offline map learning with EM is unsatisfactory for many reasons. An online solution, where we regard the map as a random variable and perform joint Bayesian inference to compute $P(L_t, M_t|z_{1:t})$, is much more useful, since it can handle maps that change, and it gives a confidence estimate in the map (which can be used to guide exploration). This joint estimation problem is usually called SLAM (simultaneous localization and mapping).

The standard approach to SLAM (see e.g., [DNCDW01] for a recent account) is to represent the map as a list of locations of landmarks. One then just applies the (extended) Kalman filter. Unfortunately, if there are N_L landmarks, this has complexity $O({N_L}^2)$, since the covariance matrix has size $N_L \times N_L$. In Section 5.3.2, I show how RBPF can reduce this to $O(N_s \times N_L)$, where N_s is the number of particles. By using an appropriate control policy, N_s can be kept constant. Consequently, we can solve the SLAM problem in much larger environments than traditional techniques. By using a tree-based data-structure, [MTKW02] reduced the complexity even further, to $O(N_s \times \log N_L)$, and demonstrated that the algorithm works on a real robot in an environment with 50,000 landmarks.

A disadvantage of representing maps as a list of landmark locations is that this requires identifying landmarks, which requires extracting features (such as corners) from the sensors and then solving the data association problem (see Section 2.4.6). This is often solved by placing beacons with unique identifiers into the environment (e.g., Durrant-Whyte submerges beacons below the sea so he can map the seabed using autonomous submarines.) The advantage of grid-based maps is that they merely require evaluating $P(y_t|L_t, M_t)$, where y_t could be a complex signal (e.g., a dense set of laser range finder distance readings) from an unmodifed environment. ($P(y_t|L_t, M_t)$ is often represented using a neural network [Thr98].) (Of course, the disadvantage of occupancy grids is that they provide a very low-level (hard-to-intrepret) representation.) Below I show how RBPF can also be applied to the SLAM problem using an occupancy grid representation.

The basic idea

Consider a robot that can move on a discrete, two-dimensional grid, and which can observe the "color" of the cells in a 3×3 neighborhood centered on its current location, (L_t^X, L_t^Y). The "color" of a cell could represent whether it is occupied or not. However, for simplicity of exposition, we shall assume that the robot can move anywhere — think of the robot as gliding over a multi-colored checkerboard. (This is a standard assumption.) Hence the map (which stores the color of each cell) affects the observation, Y_t, but not the position: see Figure 5.7.

If we could do inference in this DBN, we would have solved the SLAM problem. Unfortunately, exact inference is intractable. If there are K colors and the grid has size N_L, the belief state, $P(M_t|z_{1:t})$, has size $O(K^{N_L})$, because all the grid cells become correlated due to sharing a common observed child c.f., the explaining away phenomenon in factorial HMMs (Section 2.3.7).

However, note the following crucial insight [Mur00]: if we knew $L_{1:t}$, the grid cells would not become correlated, because we would always know exactly which cells to update for each measurement, c.f., the data association problem (Section 2.4.6). Hence if we sample paths $L_{1:t}$, we can represent the belief state as

$$P(M_t, L_t|z_{1:t}) \approx \sum_{i=1}^{N_s} \prod_j P(M_t(j)|z_{1:t}, L_{1:t}^i) P(L_t^i|z_{1:t})$$

where $M_t(j)$ is the j'th grid cell at time t. (I have assumed a factored prior, $P(M_1) = \prod_j P(M_1(j))$.) This can be implemented similarly to RBPF for switching KFMs: each particle contains a sample of L_t^i and a product of factors, $P(M_t(j)|z_{1:t}, L_{1:t}^i)$, only one of which will get updated at every step (namely the entry $j = L_t^i$).[1]

Results for a 1D grid

To evaluate the effectiveness of this idea, we constructed a simple 1D grid world which was small enough to allow for an exact solution. Specifically, we used the grid in Figure 5.8, with $N_L = 8$ and $K = 2$, so exact inference takes about 50,000 operations per time step (!).

We assume that the robot sees a noisy version of the color of the cell at its current location:

$$P(y_t|m_1, \ldots, m_{N_L}, L_t = l) \begin{cases} 1 - p_o & \text{if } y_t = m_l \\ p_o & \text{if } y_t \neq m_l \end{cases}$$

where p_o is the probability that a color gets misread (observation error). We also assume that the robot can only move left or right, depending on the control input, U_t. The robot moves in the desired direction with probability $1-p_a$, and otherwise stays put. In addition, it cannot move off the edges of the grid. Algebraically, this becomes

$$P(L_t = l'|L_{t-1} = l, U_t = \rightarrow) = \begin{cases} 1 - p_a & \text{if } l' = l + 1 \text{ and } 1 \leq l' < N_L \\ p_a & \text{if } l = l' \text{ and } 1 \leq l' < N_L \\ 1 & \text{if } l = l' = N_L \\ 0 & \text{otherwise} \end{cases}$$

The equation for $P(L_t = l'|L_{t-1} = l, U_t = \leftarrow)$ is analogous. In Section 5.3.2, we will consider the case of a robot moving in two dimensions. In that case, the motion model will be slightly more complex.

[1] [Pfe01] shows that maintaining the marginals of a belief state is sufficient for exact inference (of future marginals) iff the CPDs have the form of a multiplexer. However, he does not consider the case in which the evidence node also has this form, as we assume here.

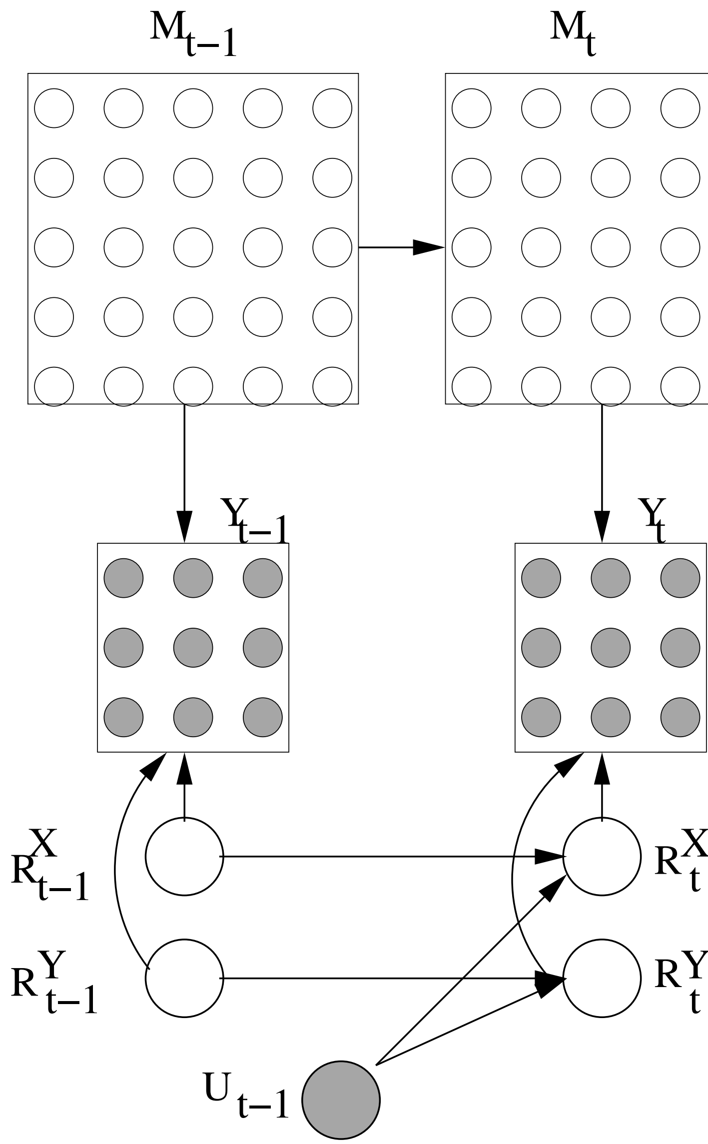
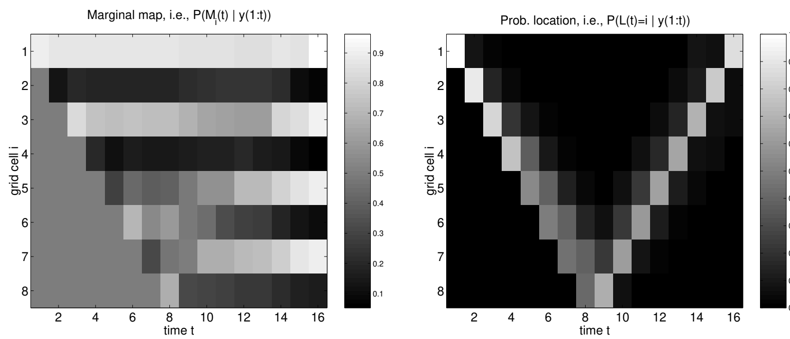


Figure 5.7: The DBN used in the map-learning problem. M represents the map, R the robot's location, Y the observation, and U the input (control). There is one arc from each (i, j) cell of M_t to Y_t. Similarly, each (i, j) cell of M_t connects to the corresponding (i, j) cell in M_{t+1}. $P(Y_t|M_t, L_t)$ is a multiplexer CPD, i.e., $P(Y_t|M_t, L_t = j) = f(Y_t, M_t(j))$.

Figure 5.8: A one-dimensional grid world.



(a) (b)

Figure 5.9: Results of exact inference on the 1D grid world. (a) A plot of $P(\mathbf{M}_t(i) = 1|y_{1:t})$, where i is the vertical axis and t is the horizontal axis; lighter cells are more likely to be color 1 (white). The pattern in the right hand column consists of alternating white and black stripes, which means it has learned the correct map. Notice how the ambiguity about the colors of cells 5–7 at time 9 gets resolved by time 11. (b) A plot of $P(L_t = i|y_{1:t})$, i.e., the estimated location of the robot at each time step.

Finally, we need to specify the prior. The robot is assumed to know its initial location, so the prior $P(L_1)$ is a delta function: $P(L_1 = 1) = 1$. If the robot did not know its initial location, there would be no well-defined origin to the map; in other words, the map is relative to the initial location. Also, the robot has a uniform prior over the colors of the cells. However, it knows the size of the environment, so the state space has fixed size.

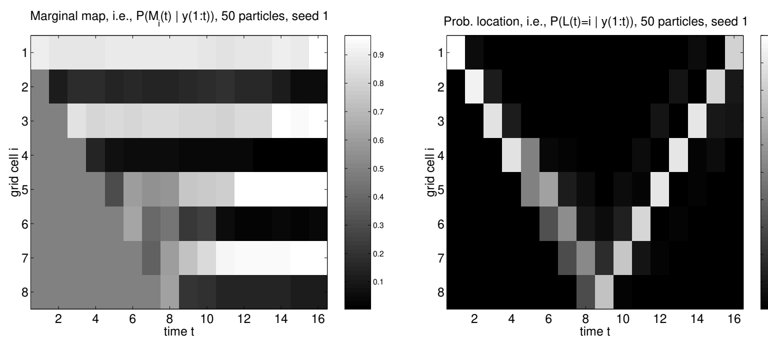
For simplicity, we fixed the control policy as follows: the robot starts at the left, moves to the end, and then returns home. Suppose there are no sensor errors, but there is a single "slippage" error at $t = 4$. What this means is that the robot issued the "go right" command, but the robot actually stayed in the same place. It has no way of knowing this, except by comparing what it sees with what it already knows about the environment, which is very little: by the time the slippage occurs, all the robot knows is the color of the cells it has already seen: cells to the right could have any color, hence if it sees two black cells in a row, it cannot tell if this is how the world really is, or whether it has got stuck, and hence is seeing the same cell twice. Clearly this is an extreme case, but it serves to illustrate the point.

We summarize the evidence sequence below, where U_t represents the input (control action) at time t.

t	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
L_t	1	2	3	**4**	**4**	5	6	7	8	7	6	5	4	3	2	1
Y_t	0	1	0	1	1	0	1	0	1	0	1	0	1	0	1	0
U_t	-	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$

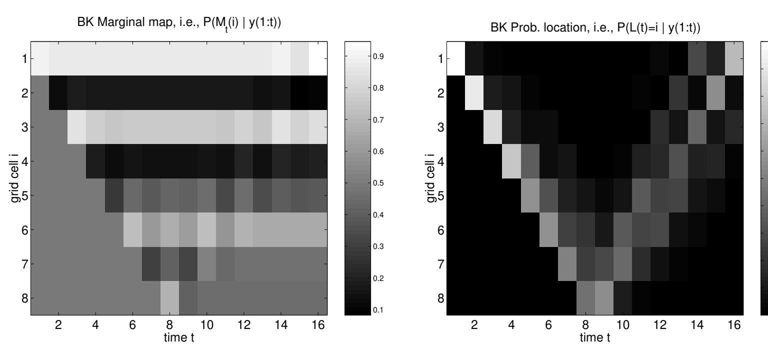
The exact marginal belief states, $P(L_t|z_{1:t})$ and $P(M_t|z_{1:t})$, are shown in Figure 5.9. At each time step, the robot thinks it is moving one step to the right, but the uncertainty gradually increases. However, as the robot returns to "familiar territory", it is able to better localize itself, and hence the map becomes "sharper", even in cells far from its current location. Note that this effect only occurs because we are modelling the correlation between all the cells. (The need for this was first pointed out in [SSC88].)

In Figure 5.10, we show the results obtained using RBPF with 50 particles proposed from the prior. The results shown are for a particular random number seed; other seeds produce qualitatively very similar



(a) (b)

Figure 5.10: Results of the RBPF algorithm on the 1D grid world using 50 particles.



(a) (b)

Figure 5.11: Results of the BK algorithm on the 1D grid world.

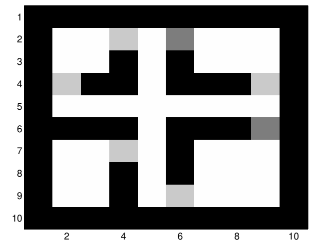


Figure 5.12: A simple 2D grid world. Grey cells denote doors (which are either open or closed), black denotes walls, and white denotes free space.

results, indicating that 50 particles are in fact sufficient in this case. Obviously, as we increase the number of particles, the error and variance decrease, but the running time increases linearly. We discuss the question of how many particles we need in Section 5.3.2.

For comparison purposes, we also tried the fully factorized version of the BK algorithm (see Section 4.2.1). The results of using BK are shown in Figure 5.11. As we can see, it performs very poorly in this case, because it ignores correlation between the cells. Of course, it is possible to use larger clusters, but unless we model the correlation between all of the cells, we will not get good results.

Results for a 2D grid

Now we now consider the 10×10 grid world in Figure 5.12. We use four "colors", which represent closed doors, open doors, walls, and free space. Doors can toggle between open and closed independently with probability $p_c = 0.1$, but the other "colors" remain static; hence the cell transition matrix, $P(M_t(j)|M_{t-1}(j))$, is

$$\begin{pmatrix} 1-p_c & p_c & 0 & 0 \\ p_c & 1-p_c & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The robot observes a 3×3 neighborhood centered on its current location. The probability that each pixel gets misclassified is $p_o = 0.1$. The robot can move north, east, south or west; there is a $p_a = 0.1$ chance it will accidently move in a direction perpendicular to the one specified by U_t.

We use a control policy that alternates between exploring new territory when the robot is confident of its location, and returning to familiar territory to relocalize itself when the entropy of $P(L_t|y_{1:t})$ becomes too high, as in [FBT98].

The results of applying the RBPF algorithm to this problem, using 200 particles, are shown in Figure 5.13. We see that by time 50, it has learnt an almost perfect map, even though the state space has size 4^{100}.

Results for $\mathbb{R}^2$

Recently the above idea has been extended so it works with maps represented in terms of landmark (x, y) locations instead of occupancy grids [MTKW02]. In this case, we sample the sequence of poses (position plus orientation) of the robot, $L_t \in \mathbb{R}^3$; conditioned on a trajectory, the map factorizes as before, $P(M_t|L^i_{1:t}, z_{1:t}) = \prod_j P(M_t(j)|L^i_{1:t}, z_{1:t})$, only now $M_t(j) \in \mathbb{R}^2$, and $P(M_t(j)|L^i_{1:t}, z_{1:t})$ is represented as a Gaussian distribution using a mean vector and a 2×2 covariance matrix. This technique, which takes $O(N_L N_s)$ operations per time step, scales much better than the standard EKF technique, which has complexity $O({N_L}^2)$, assuming the number of particles is reasonable (see Section 5.3.2). Some results are shown in Figure 5.14. In fact, by using a tree-based data structure, we can share submatrices between particles, even if they have different histories; this reduces the update cost of each particle from $O(N_L)$ to $O(\log N_L)$ [MTKW02]. Unfortunately, this scheme is rather complex and has not yet been implemented (Thrun, personal communication).

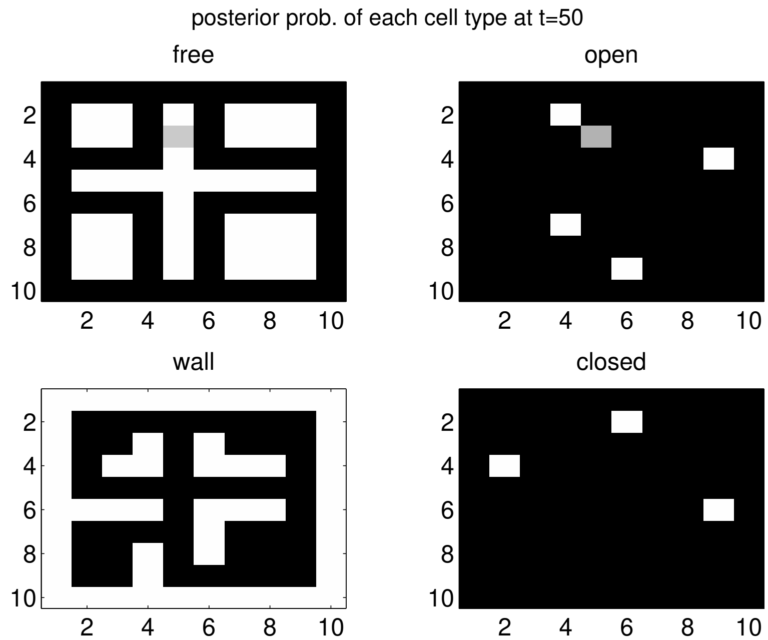


Figure 5.13: Results of RBPF on the 2D grid world using 200 particles. Cell (i, j) in the top left figure represents $P(M_t(i, j) = \text{free}|z_{1:t})$ using a gray scale, and similarly for the other three figures. Hence the open doors correspond to white blobs on the top right figure, etc.

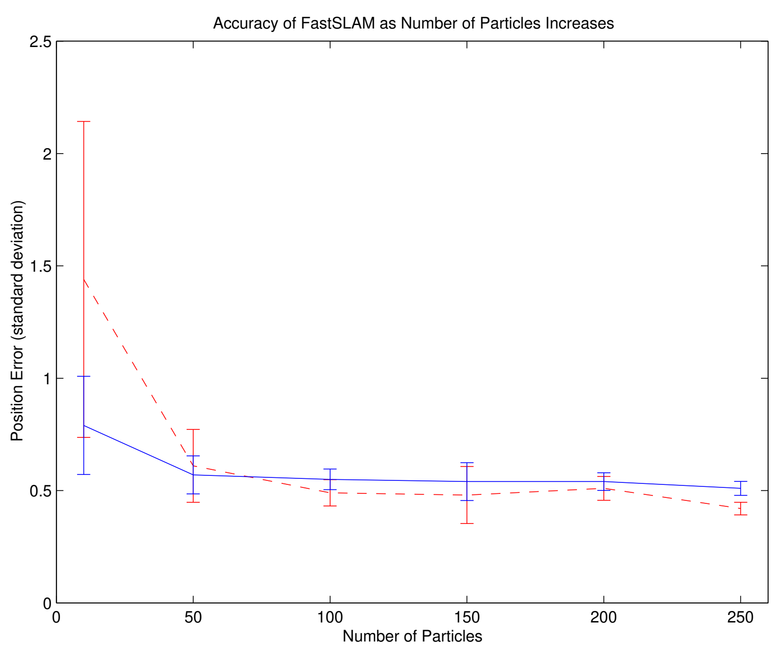


Figure 5.14: Accuracy vs num. particles for an environment with 50,000 landmarks. Solid line: robot location, dotted line: landmark location.

How many particles do we need?

The number of particles required depends both on the noise parameters and the structure of the environment. (If every cell has a unique color, localization, and hence map learning, is easy.) Since we are sampling trajectories, the number of hypotheses grows exponentially with time. In the worst case, the number of particles needed may depend on the length of the longest cycle in the environment, since this determines how long it might take for the robot to return to "familiar territory" and "kill off" some of the hypotheses (since a uniform prior on the map cannot be used to determine L_t when the robot is visiting places for the first time). In the 1D example, the robot was able to localize itself quite accurately when it reached the end of the corridor, since it knew that this corresponded to cell 8. In the 2D example, to keep the number of particles constant, I found it necessary to use a control policy that backtracked whenever the robot got lost. Thrun et al. found that a similar policy worked well in the continuous case.

5.3.3 RBPF for general DBNs: towards a turn-key algorithm

We have now seen two different examples of RBPF, applied to a switching KFM and to the SLAM DBN. In both cases, a human decided which nodes to sample and which to integrate out. However, if we are to build an autonomous life-long learning agent, it must decide by itself how to do inference (especially if it is doing online structure learning). Given an arbitrary DBN, how should it decide which nodes to sample?

The key requirement is that, conditioned on the sampled nodes, call them R_t, we should be able to compute $P(X_t|r^i_{1:t}, y_{1:t})$ more efficiently than by sampling everything. Determining whether this is the case or not depends not only on the graph structure, but also on the parameterization. For example, if the system is linear-Gaussian, conditional on R_t, we know we can perform the computation in closed form using the Kalman filter. To automate this kind of knowledge would require a theorem prover/ symbolic algebra package c.f., the AutoBayes project [FS02]. If the system is not linear-Gaussian, it might be nearly so, conditioned on some other variables, and hence we could apply an EKF or UKF. It will be very hard to automatically detect this kind of situation.

Sometimes the graph structure will be revealing, even without knowing the details of the CPDs. For example, consider sampling R_t in Figure 5.15. Even to sample from the prior, we must compute

$$\begin{aligned}
P(R_t|r^{(i)}_{1:t-1}, y_{1:t-1}) &= \sum_{x_{t-1}} P(R_t|r^{(i)}_{1:t-1}, y_{1:t-1}, x_{t-1}) P(x_{t-1}|r^{(i)}_{1:t-1}, y_{1:t-1}) \\
&= \sum_{x_{t-1}} P(R_t|r^{(i)}_{t-1}, x_{t-1}) P(x_{t-1}|r^{(i)}_{1:t-1}, y_{1:t-1})
\end{aligned}$$

Since this requires summing or integrating over X_{t-1}, this will in general be infeasible. Hence we typically assume that the nodes that are being sampled have no incoming arcs from non-sampled nodes; in this case, we require there to be no arc from X_{t-1} to R_t. Hence the sampled nodes will generally be roots of the graph (R for root), or children of sampled roots. Note that parameters are usually root nodes; once the parameters have been sampled, the remaining variables can often be handled exactly.

In the best case, the problem decomposes into two or more completely separate subproblems, conditioned on the root(s). An example was shown in Figure 3.16, where it is crucial that the dotted arcs coming into the root are absent. If the dotted arcs were present, R_t would act like a common observed child, correlating the subprocesses instead of separating them.

5.4 Smoothing

So far, we have only considered filtering problems. Often we want to compute $P(X_t|y_{1:T})$, or the fixed lag version, $P(X_{t-L}|y_{1:t})$. This is especially useful for learning (see Chapter 6). This can be done by sampling trajectories using particle filtering, and then recalculating the particles' weights via a backwards recursion [GDW00]. However, we would really like future evidence to affect the positions of the particles as well as their weights. It is possible to run a PF in both the forwards and backwards directions [IB98], but this has complexity $O({N_s}^2)$, so is not popular.

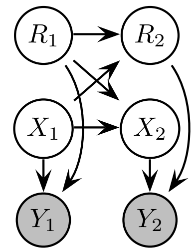


Figure 5.15: Example of an DBN for which RBPF is unsuitable.

The most widely used technique is based on MCMC, in particular, Gibbs sampling (see e.g., [DK02] for a very recent approach). Unfortunately, Gibbs sampling mixes very slowly for time series, because the data are correlated. It is therefore crucial to use Rao-Blackwellisation. The most important case where this is possible is in conditionally linear-Gaussian models, which we discuss below.

5.4.1 Rao-Blackwellised Gibbs sampling for switching KFMs

Carter and Kohn [CK96] proposed the following Rao-Blackwellised MCMC algorithm for inference in switching KFMs. ([KN98] is a whole book devoted to MCMC for switching KFMs.)

1. Randomly initialise $s^0_{1:T}$.
2. For $i = 1, 2, \ldots$ until convergence
   (a) For $t = 1, \ldots, T$, sample $s^i_t \sim P(S_t|y_{1:T}, s^i_{-t})$
   (b) Compute $x^i_{0:T} = \mathrm{E}\left[x_{0:T}|y^i_{1:t}, s^i_{1:T}\right]$

where $s^i_{-t} \stackrel{\text{def}}{=} \left(s^i_1, \ldots, s^i_{t-1}, s^{i-1}_{t+1}, \ldots, s^{i-1}_T\right)$ contains new values of S_t to the left of t, and old values to the right. The final estimate is then obtained by averaging the samples $s^i_{1:T}$ and $x^i_{1:T}$, possibly discarding an initial segment corresponding to the burn-in period of the Markov chain.

The main issue is how to efficiently compute the sampling distribution $P(S_t|y_{1:T}, s^i_{-t})$. [CK96] propose an $O(T)$ algorithm which involves first running the backwards Kalman filter[2], and then working forwards, sampling S_t and doing Bayesian updating conditional on the sampled value.

The paper is hard to read because of the heavy notation involved in explicitely manipulating Gaussians. Here, we rederive the algorithm for the simpler case in which X is a discrete random variable. (Y can have an arbitrary distribution, since it is observed.) The original algorithm can then be recovered by replacing the discrete multiplication and addition operators on the X_t nodes with their Gaussian equivalents, as discussed in Section B.5. We then show how to apply the same idea to arbitrary DBNs using the jtree algorithm.

Computing the sampling distribution

The sampling distribution is

$$P(s_t|y_{1:T}, s_{-t}) \propto P(y_{1:T}|s_{1:T})P(s_t|s_{-t})$$

[2] [CK96] assume that the transition matrices are all invertible, so they can compute the backwards filter in moment form (c.f., Section 3.2.5). [DA99] use the information form of the backwards filter, which does not require invertible dynamics.

The first term is given by

$$
\begin{aligned}
P(y_{1:T}|s_{1:T}) &= \sum_i P(y_{t+1:T}|y_{1:t}, s_{1:T}, X_t = i) P(y_{1:t}, X_t = i|s_{1:T}) \\
&= \sum_i P(y_{t+1:T}|s_{t+1:T}, X_t = i) P(y_{1:t}, X_t = i|s_{1:t}) \\
&= P(y_{1:t}|s_{1:t}) \sum_i P(y_{t+1:T}|s_{t+1:T}, X_t = i) P(X_t = i|y_{1:t}, s_{1:t}) \\
&= P(y_{1:t-1}|s_{1:t-1}) P(y_t|y_{1:t-1}, s_{1:t}) \sum_i P(y_{t+1:T}|s_{t+1:T}, X_t = i) P(X_t = i|y_{1:t}, s_{1:t})
\end{aligned}
$$

Hence, dropping terms that are independent of s_t,

$$
P(s_t|y_{1:T}, s_{-t}) \propto P(s_t|s_{t-1}) P(s_{t+1}|s_t) P(y_t|y_{1:t-1}, s_{1:t}) \sum_i P(y_{t+1:T}|s_{t+1:T}, X_t = i) P(X_t = i|y_{1:t}, s_{1:t})
$$

The $P(X_t = i|y_{1:t}, s_{1:t})$ and $P(y_t|y_{1:t-1}, s_{1:t})$ terms can be computed by running the forwards algorithm on the X/Y HMM, sampling S_t as we go. The $P(y_{t+1:T}|s_{t+1:T}, X_t = i)$ term can be computed by using the backwards algorithm on the X/Y HMM. The key observation is that this term is independent of $s_{1:t}^{(k)}$, and hence can be computed ahead of time, before sampling $s_{1:t}$ on the forwards pass.

Using the jtree algorithm

We now show how to implement the above scheme using the jtree algorithm. The jtree is shown in Figure 4.13. Let us make the first clique (on the left) the root. Initially the clique potentials will contain the CPDs: $\phi(X_{t-1}, S_{t-1}, X_t, S_t) = P(y_t|X_t, S_t) P(S_t|S_{t-1}) P(X_t|X_{t-1}, S_t)$. After instantiating $S_{1:T}$, and doing a backwards pass (collect to root), the separator potentials contain terms of the form $\phi(X_t, S_t) \propto P(y_{t+1:T}|s_{t+1:T}, X_t, s_t)$, c.f., β_t in an HMM. Now, in the forwards pass (distribute from root), we sample all the S variables in a clique using Dawid's algorithm [Daw92] (see Section 5.2.1). Hence the updated separator potentials have the form $\phi^*(X_{t-1}, S_{t-1}) \propto P(X_{t-1}|y_{1:T}, s_{1:t-1}^*, s_{t:T})$, c.f. γ_t in an HMM. The message entering a clique is now

$$
\frac{\phi^*(X_{t-1}, S_{t-1})}{\phi(X_{t-1}, S_{t-1})} \propto P(X_{t-1}, y_{1:t-1}|s_{1:t-1}^*)
$$

since $\alpha_t \propto \gamma_t/\beta_t$. Combining, we have

$$
\alpha_t \phi(X_{t-1}, S_{t-1}, X_t, S_t) \beta_t \propto P(X_{t-1}, S_{t-1}, X_t, S_t|y_{1:T}, s_{1:t-1}^*, s_{1:t})
$$

Hence we can sample a new S_t and continue. This is equivalent to the algorithm above.

Stochastic vs deterministic approximations

An obvious question is: how does the above Rao-Blackwellised Gibbs sampler compare to offline deterministic approximations such as variational methods and expectation propagation, in terms of accuracy vs computation time. As far as I know, this have never been studied.

Chapter 6

DBNs: learning

The techniques for learning DBNs are mostly straightforward extensions of the techniques for learning BNs (see Appendix C); we discuss the differences in Section 6.1. The novel contributions of this chapter are the application of these algorithms. In Section 6.2.1, we discuss how to use the structural EM algorithm to learn "genetic network" topologies from synthetic data [FMR98]. In Section 6.2.2, we discuss how to use hierarchical HMMs to discover "motifs" buried in synthetic data. In Section 6.2.3, we discuss how to use abstract HMMs to predict a person's goal as they move through a building, based on real tracking data. In Section 6.2.4, we discuss how to model some real traffic flow data using coupled HMMs [KM00]. Note that most of these "applications" are more proof of concept rather than serious attempts to solve the respective problems. However, we believe they illustrate the applicability of DBNs.

6.1 Differences between learning static and dynamic networks

6.1.1 Parameter learning

Offline learning

To do offline parameter estimation, we can use the techniques discussed in Appendix C, such as EM. Here we just comment on some points that are applicable specifically to dynamical systems.

- Parameters must be tied across time-slices, so we can model sequences of unbounded length. This is straightforward to handle: we simply pool the expected sufficient statistics for all nodes that share the same parameters.
- The parameters, π, for $P(X_1)$ are usually taken to represent the initial state of the dynamical system; in this case, π can be estimated independently of the transition matrix. However, if π represents the stationary distribution of the system, the parameters of π and the transition model become coupled. A way to compute maximum likelihood estimates (MLEs) in this case is discussed in [Nik98].
- Linear-Gaussian DBNs (i.e., KFMs) have been studied very extensively. In some cases, closed-form solutions to the MLEs can be found. In particular, for a KFM with no output noise, one can use subspace methods [OM96].[1] This is analogous to the fact that PCA can be used instead of EM for noise-free factor analysis [RG99].

Online learning

To do online parameter estimation, we simple add the parameters to the state space and then do online inference (i.e., filtering). We give some examples in Section 6.2.5. For (non-Bayesian) techniques for recursive (online) parameter estimation in partially observed linear dynamical systems, see [LS83, Lju87].

[1] These are implemented in the n4sys function in the Matlab system identification toolbox. For an impressive application of this technique to synthesizing dynamic visual textures, see [SDW01].

6.1.2 Structure learning

When learning the structure of a DBN, we can learn the intra-slice connectivity, which must be a DAG, and the inter-slice connectivity, which is equivalent to the variable selection problem, since for each node in slice t, we must choose its parents from slice $t-1$. If we assume the intra-slice connections are fixed, this means structure learning for DBNs reduces to feature selection.

If the system is fully observed, we can apply standard feature selection algorithms, such as forward or backwards stepwise selection, or the leaps and bounds algorithm [HTF01, p55]. Hierarchical priors, which encode the fact that we only expect a subset of the inputs to be relevant to each output, are discussed in [NJ01]. These can be used inside a greedy search or an MCMC algorithm.

When the system is partially observed, structure learning becomes computationally intensive. One practical approach is the structural EM (SEM) algorithm (see Section C.8.2). It is straightforward to extend this to DBNs [FMR98]. We give an example in Section 6.2.1. An extension to the structural EM algorithm for DBNs is presented in [BFK99]. These authors add extra hidden variables when violations of the first-order Markov condition are detected. (A very similar idea was proposed in [McC95] for learning POMDP state-spaces). See also [ELFK00] for methods to discover hidden variables in static Bayes nets.

Learning uses inference as a subroutine, so learning can be slow. We can obviously use standard approximate DBN inference algorithms to speed things up, but there is an additional, natural approximation which arises in the context of the SEM algorithm: Since SEM needs to compute the joint distribution over sets of nodes which may not all belong to the same clique, one may approximate the joint as a product of marginals [BFK99].

6.2 Applications

6.2.1 Learning genetic network topology using structural EM

Here we describe some initial experiments using DBNs to learn small artificial examples typical of the causal processes involved in genetic regulation. We generate data from models of known structure, learn DBN models from the data in a variety of settings, and compare these with the original models. The main purpose of these experiments is to understand how well DBNs can represent such processes, how many observations are required, and what sorts of observations are most useful. We refrain from describing any *particular* biological process, since we do not yet have sufficient real data on the processes we are studying to learn a scientifically useful model.

Simple genetic systems are commonly described by a *pathway model*—a graph in which vertices represent genes (or larger chromosomal regions) and arcs represent causal pathways (Figure 6.1(a)). A vertex can either be "off/normal" (state 0) or "on/abnormal" (state 1). The system starts in a state which is all 0s, and vertices can "spontaneously" turn on (due to unmodelled external causes) with some probability per unit time. Once a vertex is turned on, it stays on, but may trigger other neighboring vertices to turn on as well—again, with a certain probability per unit time. The arcs on the graph are usually annotated with the "half-life" parameter of the triggering process. Note that pathway models, unlike BNs, can contain directed cycles. For many important biological processes, the structure and parameters of this graph are completely unknown; their discovery would constitute a major advance in scientific understanding.

Pathway models have a very natural representation as DBNs: each vertex becomes a state variable, and the triggering arcs are represented as links in the transition network of the DBN. The tendency of a vertex to stay "on" once triggered is represented by persistence links in the DBN. Figure 6.1(b) shows a DBN representation of the five-vertex pathway model in Figure 6.1(a). The nature of the problem suggests that noisy-ORs (or noisy-ANDs) should provide a parsimonious representation of the CPD at each node (see Section A.3.2). To specify a noisy-OR for a node with k parents, we use parameters $q_1, \ldots, q_k$, where q_i is the probability the child node will be in state 0 if the ith parent is in state 1. In the five-vertex DBN model that we used in the experiments reported below, all the q parameters (except for the persistence arcs) have value 0.2. For a strict persistence model (vertices stay on once triggered), q parameters for persistence arcs are fixed at 0. To learn such noisy-OR distributions, we used the EM techniques discussed in Section C.2.3. We also tried using gradient descent, following [BKRK97], but encountered difficulties with convergence in cases where the optimal parameter values were close to the boundaries (0 or 1). To prevent structural

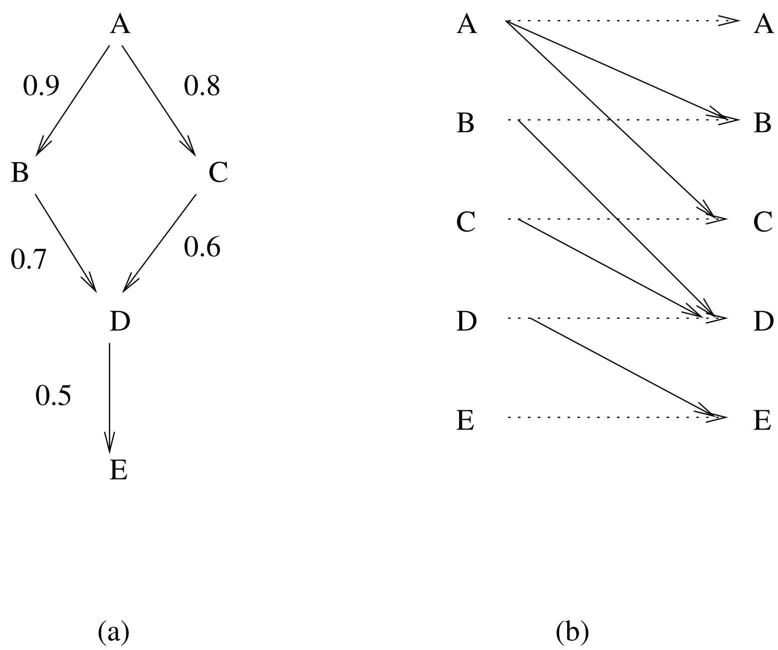


Figure 6.1: (a) A simple pathway model with five vertices. Each vertex represents a site in the genome, and each arc is a possible triggering pathway. (b) A DBN model that is equivalent to the pathway model shown in (a). (c) The DBN model extended by adding a switch node S and an observation node O, whose value either indicates that this slice is hidden, or it encodes the state of all of the nodes in the slice.

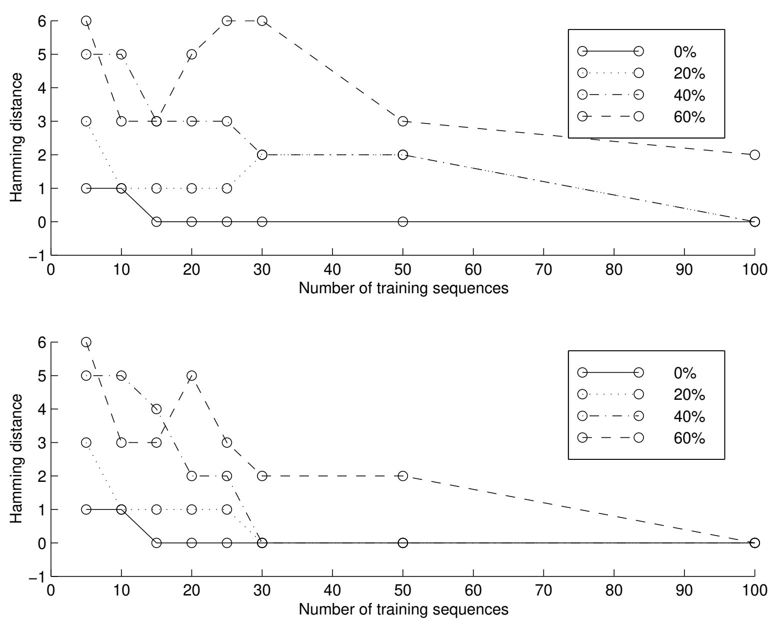


Figure 6.2: Results of structural EM for the pathway model in Figure 6.1. We plot the number of incorrect edges against number of training slices, for different levels of partial observability. 20% means that 20% of the slices, chosen at random, were fully hidden. Top: tabular CPDs; bottom: noisy-OR CPDs.

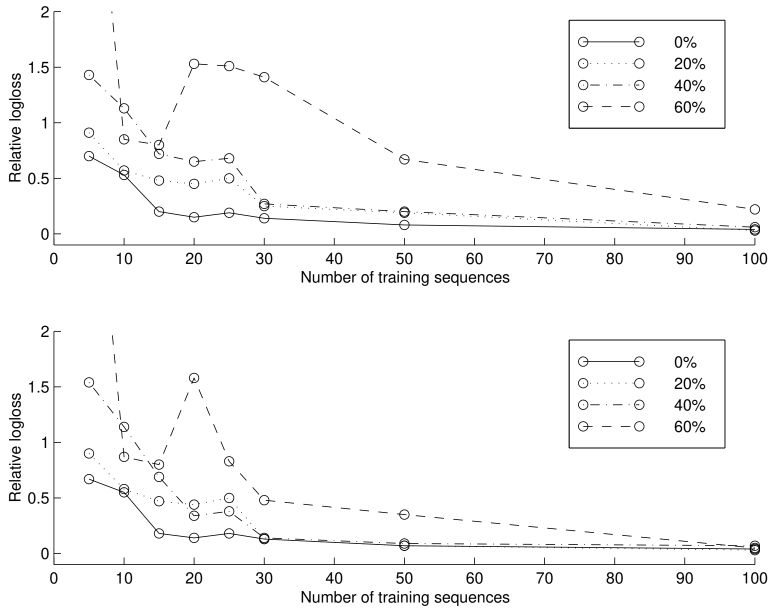


Figure 6.3: As in Figure 6.2, except we plot relative log-loss compared with the generating model on an independent sample of 100 sequences. Top: tabular CPDs; bottom: noisy-OR CPDs.

overfitting, we used a BIC penalty, where the number of parameters per node was set equal to the number of parents.

In all our experiments, we enforced the presence of the persistence arcs in the network structure. We used two alternative initial topologies: one that has only persistence arcs (so the system must learn to add arcs) and one that is fully interconnected (so the system must learn to delete arcs). Performance in the two cases was very similar. We assumed there were no arcs within a slice.

We experimented with three observation regimes that correspond to realistic settings:

- The complete state of the system is observed at every time step.
- Entire time slices are hidden uniformly at random with probability h, corresponding to intermittent observation.
- Only two observations are made, one before the process begins and another at some unknown time $t_{\rm obs}$ after the process is initiated by some external or spontaneous event. This might be the case with some disease processes, where the DNA of a diseased cell can be observed but the elapsed time since the disease process began is not known. (The "initial" observation is of the DNA of some other, healthy cell from the same individual.)

This last case, which obtains in many realistic situations, raises a new challenge for machine learning. We resolve it as follows: we supply the network with the "diseased" observation at time slice T, where T is with high probability larger than the actual elapsed time $t_{\rm obs}$ since the process began.[2] We also augment the DBN model with a hidden "switch" variable S that is initially off, but can come on spontaneously. When the switch is off, the system evolves according to its normal transition model $P(X_t|X_{t-1}, S = 0)$, which is to be determined from the data. Once the switch turns on, however, the state of the system is frozen—that is, the conditional probability distribution $P(X_t|X_{t-1}, S)$ is fixed so that $X_t = X_{t-1}$ with probability 1. The persistence parameter for S determines a probability distribution over $t_{\rm obs}$; by fixing this parameter such that (a priori) $t_{\rm obs} < T$ with high probability, we effectively fix a scale for time, which would otherwise be

[2] With the q parameters set to 0.2 in the true network, the actual system state is all-1s with high probability after about $T = 20$, so this is the length used in training and testing.

arbitrary. The learned network will, nonetheless, imply a more constrained distribution for t_{obs} given a pair of observations.

We consider two measures of performance. One is the number of different edges in the learned network compared to the generating network, i.e., the Hamming distance between their adjacency matrices. The second is the difference in the logloss of the learned model compared to the generating model, measured on an independent test set. Our results for the five-vertex model of Figure 6.1(a) are shown in Figures 6.2 and 6.3. We see that noisy-ORs perform much better than tabular CPTs when the amount of missing data is high. Even with 40% missing slices, the exact structure is learned from only 30 examples by the noisy-OR network.[3] However, when all-but-two slices are hidden, the system needs 10 times as much data to learn. The case in which we do not even know the time at which the second observation is made (which we modeled with the switching variable) is even harder to learn (results not shown).

Of course, it will not be possible to learn real genetic networks using techniques as simple as this. For one thing, the data sets (e.g., micro-arrays) are noisy, sparse and sampled at irregular intervals.[4] Second, the space of possible models is so huge that it will be necessary to use strong prior domain knowledge to make the task tractable. Succesful techniques will probably be more similar to "computer assisted pathway refinement" [ITR$^+$01] than to "de novo" learning.

6.2.2 Inferring motifs using HHMMs

Motifs are short patterns which occur in DNA and have certain biological significance (see e.g., [XJKR02] and references therein). In the simplest case, a motif can be represented as a string over the alphabet $\Sigma = \{A, C, G, T\}$. More realistically, a motif can be represented as a probability distribution over strings of a fixed length; if each position (column) is assumed to be independent, this is often called a profile.

Given a set of sequences, the problem is discover all the motifs in the set. It is common to "wish away" the model selection problem by assuming the number of motif types, and their lengths, are known already. It is also common to assume motifs do not overlap, and to model the "background" by a single multinomial, which contains the overall probabilities of each letter in the alphabet. With these assumptions, we can model the problem by using a 2-level HHMM, as shown in Figure 6.4.

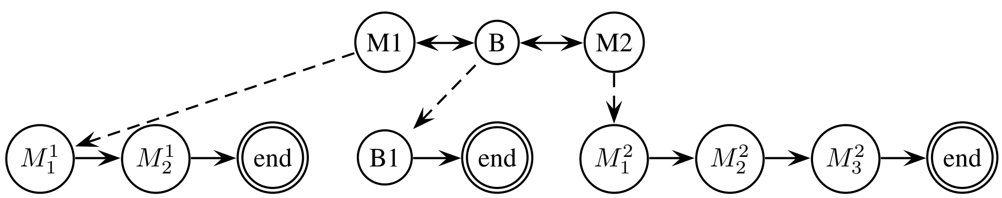


Figure 6.4: State transition diagram of an HHMM for modelling two types of motifs of lengths 2 and 3, and a single-state background model.

The transition probabilities for the top level of this model encode the probability of transition from the background to a motif of type 1 or 2. (We assume we cannot go directly from one motif to another.) The self-loop probability for state $B1$ encodes the expected length of the inter-motif background segments by using a geometric distribution. The output distribution of state M_i^j represents the profile column for position i in motif type j.

I conducted the following very simple experiment, to see how feasible it is to learn motifs in an unsupervised fashion. I created a set of $N_{\mathrm{train}} = 100$ strings, each of length $T = 20$, with each position drawn uniformly at random from Σ. I then "embedded" a single occurrence of a known motif, say "accca", at a random position (away from the boundaries) in each string. Here is a sample of the training data.

```
gcggccgtccacccacagtt
cagcagtgttaacccaggtt
cgtatacccagctagctcga
```

[3] What is not clear from the graphs, however, is that noisy-ORs require many more EM iterations to converge, and each iteration is much slower; this is because of the extra hidden nodes we introduce to learn the noisy-OR CPDs.

[4] See [BJGGJ02] for a spline-based interpolation approach which could be useful as a preprocessing step for learning model structure.

I created an HHHM like the one in Figure 6.4, but with a single motif type of length 4. I initialized parameters randomly, but subject to the known state transition topology, and ran EM until convergence. I then computed the MAP motif by looking at the most likely output symbol for each M_i^j. Even using multiple restarts, I found that it was almost impossible to recover the "true" motif.

To make the problem easier, I added the prior knowledge that I expected the output distributions of states in the motif model (but not the background model) to be (nearly) deterministic, i.e., each column in the motif should only emit a single symbol, although I did not specify which one. This can be modelled by using a minimum entropy prior [Bra99a]:

$$P(\theta) \propto e^{-H(\theta)}$$

where θ are the multinomial parameters for a given column, and $H(\theta) = -\sum_i \theta_i \log \theta_i$ is the entropy.[5] With the minent prior, EM reliably recovered the "true" motif.[6] For example, here is the learned profile for each state, where the rows represent A, C, G, T in that order, and columns represent the 5 states of the motif sub-HMM.

0.95977	0.0182	0.0024127	3.9148e-05	0.96126
0.0097752	0.96917	0.97285	0.9954	0.02166
0.00060423	0.012363	0.023105	0.0003953	0.0053652
0.029847	0.00026544	0.0016343	0.0041678	0.011715

Taking the most probable entry per column indicates that the system has learned the "accca" pattern.

In the future I plan to work with Eric Xing on extending this model so that it can be applied to real motif data.

6.2.3 Inferring people's goals using abstract HMMs

We now show how a DBN can be used to infer people's intentional states by observing their behavior c.f., Section 2.3.15. The raw data consists of the estimated (x, y) position of a person as they move through the ground floor of a house; this is inferred from a series of stationary laser range finders, as shown in Figure 6.5. The aim is to predict which landmark they will go to next. The landmarks are manually chosen locations within the building, and correspond to top level goal states. In addition to the person's goal, G_t, the actual position of the person, S_t, is represented using another discrete state variable; this is essentially a vector quantization of their actual, observed (but noisy) (x, y) position, Y_t.[7] The overall DBN is shown in Figure 2.29.

We trained this model using EM from a single training sequence of length $T \sim 2000$. We automatically segmented the sequence into pieces by detecting periods of no motion (which simulated the person spending time at the goal landmark), and then manually labelled each segment with the its goal end-point: see Figure 6.6. After training, the observed nodes essentially tiled the space where the person walked, as shown in Figure 6.7. The transition matrix for the state nodes, $P(S_t|S_{t-1}, G_t)$, is like a conditional plan, in that it specifies what state the person goes to next, given the state they were in and conditioned on their goal. However, it is unlike a plan in that it does not (explicitly) optimize any cost function. An example is shown in Figure 6.8. The transition matrix for the goal nodes is shown in Figure 6.9. This can be estimated by simple counting, since we assume the goals are always observed in the training data.

To demonstrate that the abstract HMM was better than the HMM at predicting, we compute $P(G_t|y_{1:t})$ using two models, one with the learned $P(G_t|G_{t-1})$ and one with a uniform $P(G_t|G_{t-1})$. We compared this with the ground truth, which is available from the hand-labelled data. The results are shown in Figure 6.10. The bimodality of the posterior during time 57–85 using the uniform (bottom) model indicates an ambiguity between the two goals states 5 (TV) and 7 (study). By using longer range information about what order the person moves from goal to goal, we can eliminate the ambiguity (middle graph in figure), and correctly infer that the person's goal is state 3 (armchair).

[5] An alternative prior would be to use a mixture of Dirichlets [BHK+93], with one mixture component for each base, plus a fifth component representing a uniform Dirichlet prior. A more biologically sophisticated prior, which models characteristic "shapes" in the motif such as the "U" shape (which represented conserved (low entropy) ends with variable (high entropy) middles), is discussed in [XJKR02].

[6] This example is included in the BNT distribution in BNT/examples/dynamic/HHMM/Motif/learn-motif-hhmm.m.

[7] By modelling $P(Y_t|S_t)$ as a mixture of Gaussians, we can get some robustness to outliers.

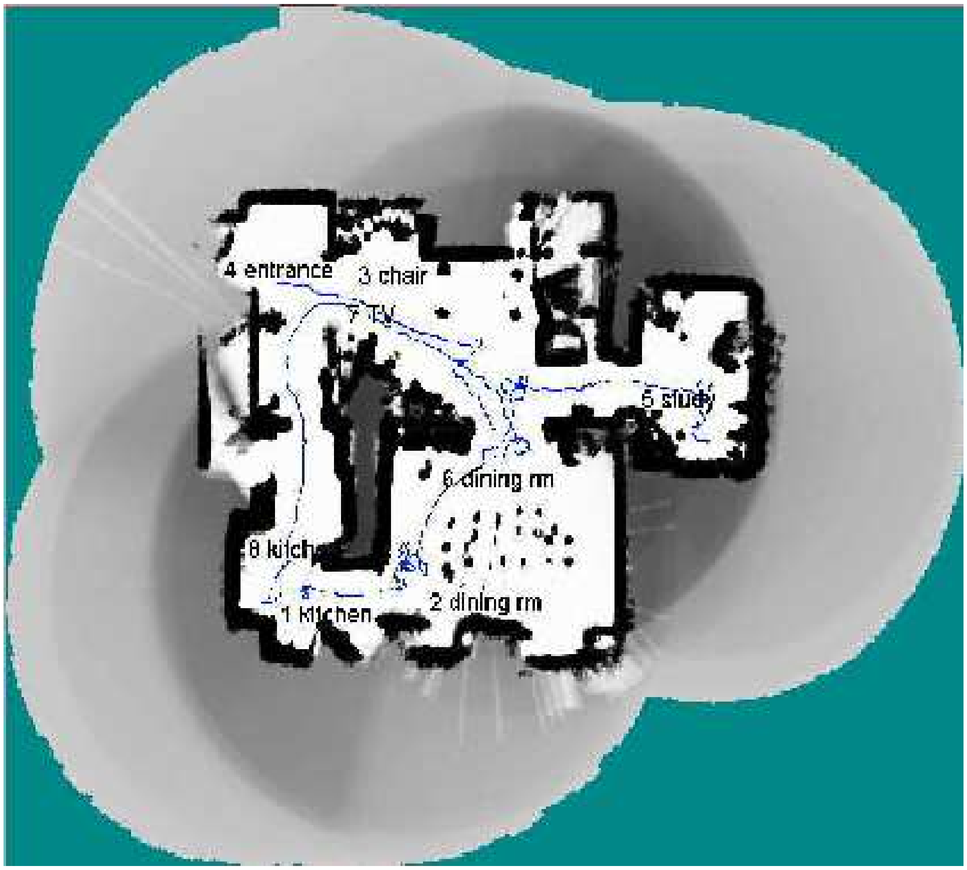


Figure 6.5: Laser range finding data from the ground floor of a house: Dark regions correspond to obstacles like walls and furniture, white regions correspond to free space. Once a map has been learned, any new obstacles which are not in the map are assumed to be people. In this way, the trajectory of a moving person (shown as a wiggly blue line) can be inferred. Various places where the person likes to spend a lot of time are labelled, and are considered landmarks or goals. This data is from [BBT02].

6.2.4 Modelling freeway traffic using coupled HMMs

Many urban freeways are equipped with induction loop detectors. These detectors can report three kinds of information in real-time [Hal96]: the number of vehicles passing the location during a given time interval (flow rate), the fraction of time that vehicles are over the detector (occupancy), and the vehicle velocities averaged over the time interval. Here, we will restrict ourselves to aggregated velocity data. See Figures 6.11 and 6.12 for a sample of the data we are using, which was collected from I-880 in Oakland, California. For details about this dataset, see [KCB00].

In Figure 6.11, we can clearly see onset, propagation and dissipation of traffic congestion; this appears as the well-known inverted triangular shape [May90]. Essentially, traffic gets slower downstream, this effect propagates upstream, only to eventually disappear. There are various theories which try to explain this kind of behavior, based on models of fluid flow, cellular automata, and microscopic computer simulations. But all of these approaches have limitations, particularly the need to tune many parameters specific to each individual freeway.

In this section, we try to learn a model of traffic velocities from data. We assume that there is a hidden variable at each of the L detector locations, which takes on K possible values, typically two (free flow and congestion), and that the observed velocity is a noisy representation of the current state. (We use a discrete hidden variable, since the time series of the average velocity vector is highly nonlinear.)

The most naive model is to assume each hidden state variable evolves independently, resulting in L independent HMMs. However, such a model cannot capture spatial correlation, and hence is incapable of

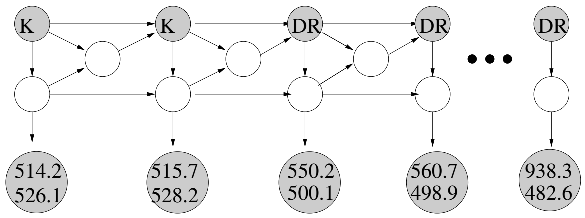


Figure 6.6: Semi-supervised training of a 2-level HHMM. The person's goal and (x, y) positions are observed, but the state, S_t, and the finished status, F_t, are hidden. K = kitchen, DR = dining room, etc.

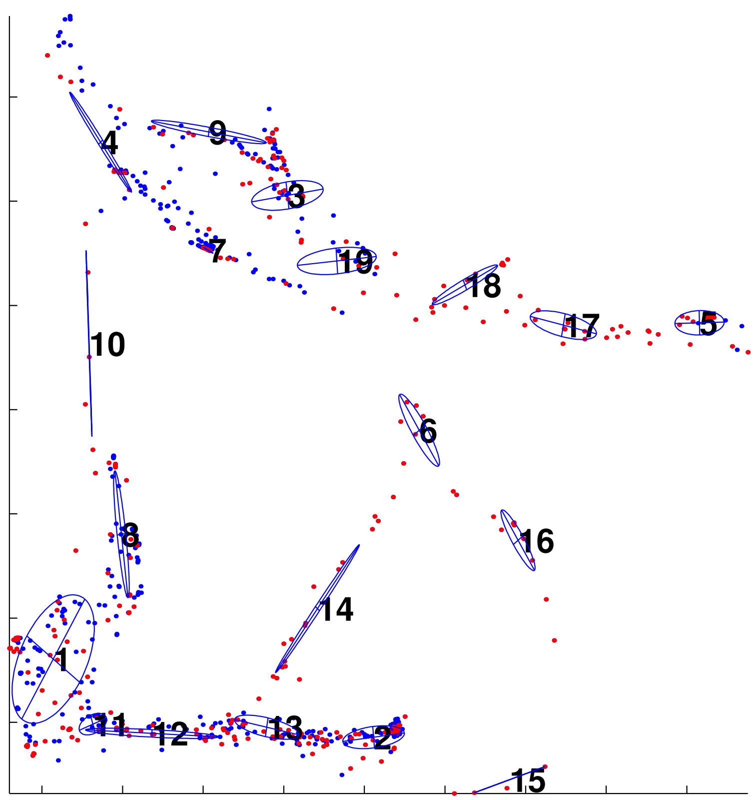


Figure 6.7: Learned observation distributions $P(Y_t|S_t)$. Each ellipse represents the coavriance matrix of the Gaussian for a given state S_t.

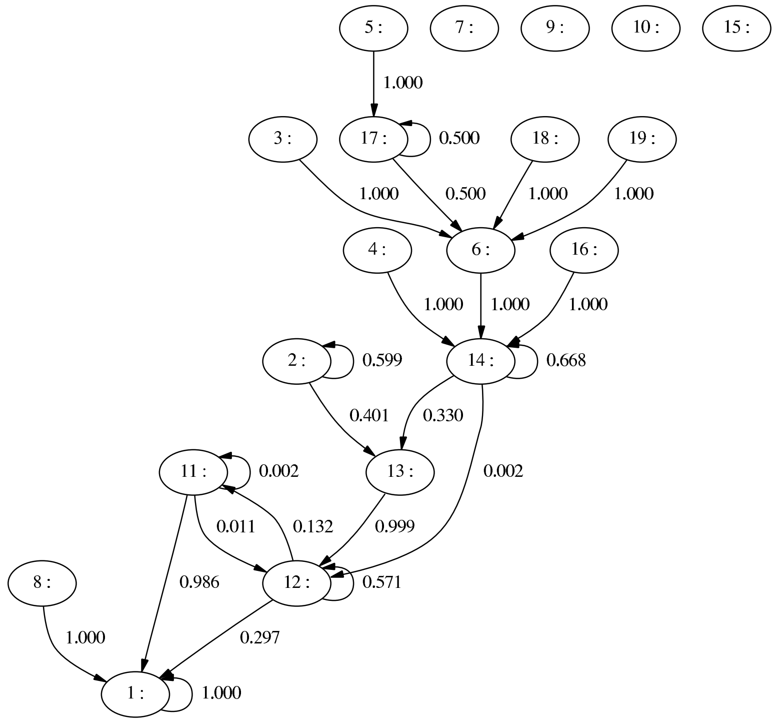


Figure 6.8: Learned state transition matrix $P(S_{t+1}|S_t, G_{t+1} = 1)$.

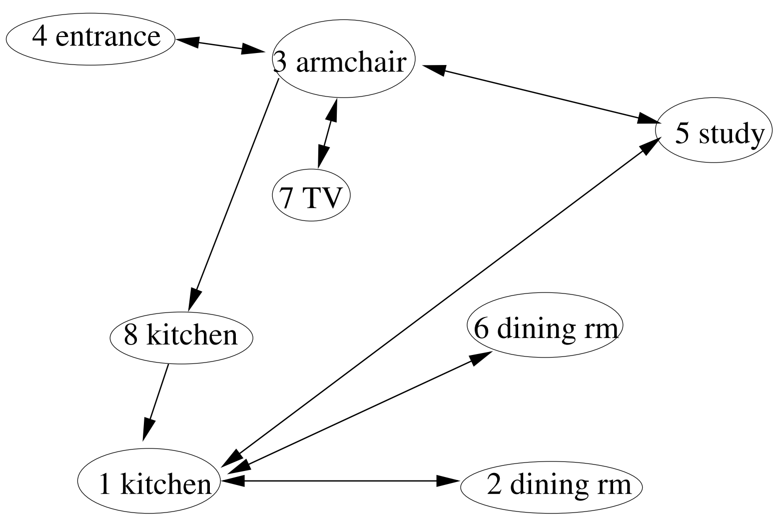


Figure 6.9: Learned goal transition matrix $P(G_t|G_{t-1})$. Only the non-zero arcs are shown.

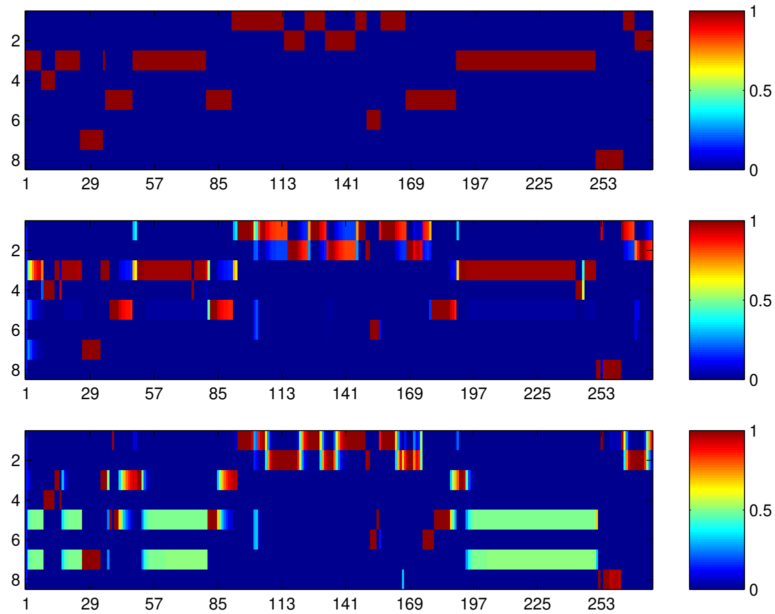


Figure 6.10: Inferring goals using three different methods. Rows represent states (goal locations), columns represent time. Shading represents $P(G_t|y_{1:t})$. First line: ground truth. Second line: using the learned goal transition matrix. Third line: using a uniform goal transition matrix.

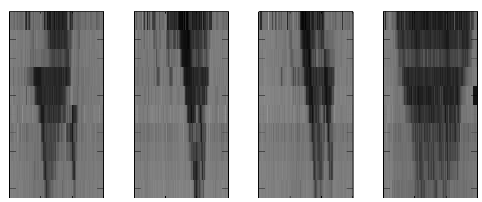


Figure 6.11: The full data set contains the velocity field for 20 weekdays, 2-7pm, between February 22 and March 19, 1993. The measurements come from 10 loop detectors (0.6 miles apart) in the middle lane of I-880, with 1 measurement per 30 seconds. Here we plot the first four days, temporally subsampled by 2. The x-axis corresponds to time, and the y-axis to space. Vehicles travel upward in this diagram. The darkest gray-scale corresponds to the average velocity of 20 miles per hour (mph) and the lightest to 70 mph. The isolated dark blob on the right edge of the fourth day is probably due to a sensor failure.

capturing the global dynamics, such as the inverted triangle shape. The other extreme is to assume each variable at time t depends on all the other (hidden) variables at time $t-1$; this results in a single HMM with a state space of size K^L, which requires $K^L(K^L-1)$ parameters just to specify its transition matrix. We therefore adopt a middle ground, and assume each variable only depends on its local neighbors in space and time; such a model has been called a coupled HMM (see Section 2.3.8). We describe the model in more detail below.

We will estimate the parameters using EM. The M step is straightforward, but the E step is in general computationally intractable. We therefore need to use approximate inference. Below we compare particle filtering (Section 5.2) and the Boyen-Koller (BK) algorithm (Section 4.2.1). We then give the results of learning using exact and approximate inference in the E step. Finally, we discuss the adequacy of the model in light of our results.

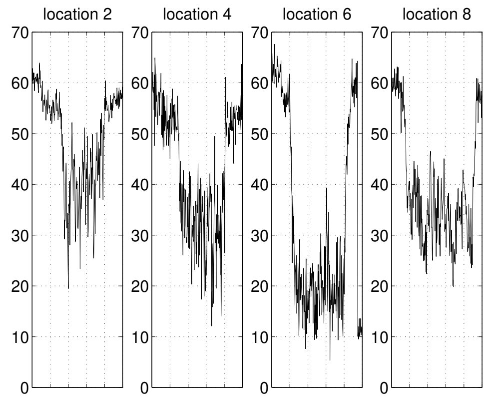


Figure 6.12: The velocity measurements for day 4 at locations 2, 4, 6 and 8. The x-axis is minutes (0 to 300), the y-axis is mph. The sudden drop from 64 to 10 mph at location 6 (which corresponds to the dark blob in Figure 6.11) is probably due to a sensor failure.

The model

Assume there are L loop detector stations indexed by $l = 1, ..., L$, from upstream to downstream. The observed (aggregated) velocity $y_{l,t}$ (mph) at location l and time t has a distribution that depends only on the underlying state variable $x_{l,t} \in \mathcal{S} = \{s_1, \cdots, s_K\}$. The simplest model is a binary chain with $K = 2$, where the two states s_C and s_F correspond to 'congestion' and 'free flow'. We initialise the mean for the congested state to be less than the mean for the free-flow state, although we do not enforce this constraint during learning.

We assume that the hidden process of $x_t = (x_{1,t}, \cdots, x_{L,t}) \in \mathcal{S}^{\otimes L}$ is Markovian and its transition probability can be decomposed as $P(x_{t+1}|x_t) = \prod_{l=1}^{L} P(x_{l,t+1}|x_t) = \prod_{l=1}^{L} P(x_{l,t+1}|x_{l-1,t}, x_{l,t}, x_{l+1,t})$, i.e., the traffic state at a location is affected only by the previous state of the neighboring locations. The initial distribution on the state is assumed to decompose as $P(X_1) = \prod_{l=1}^{L} P(X_{l,1})$. Finally, the observed velocity is a Gaussian whose mean and variance depends on the underlying state at the location: $P(y_{l,t}|x_{l,t} = s_k) \sim N(\mu_{l,k}, \sigma^2_{l,k})$. We need $LK^3(K-1) + 2K^2(K-1)$ parameters to specify the transition model, $2LK$ to specify the observation model. and LK to specify the initial distributions. If $L = 10$ and $K = 2$, this is a total of $88 + 40 + 20 = 148$ parameters.

Because of the non-stationary nature of the data, we split the 5 hour period of each day into five 1-hour blocks (60 time slices). Then we fit a binary state ($K = 2$) CHMM for each block separately, considering 20 days as independent replicates of the process. We use days 1,3,...,19 as the training set and 2,4,...,20 as the test set. Hence we have 10×60 observations to fit 148 parameters per model.

Inference

Exact inference in this model takes $O(T(L-1)K^{L+2})$ time and space. For $L = 10$, $K = 2$ and $T = 60$, this is about 2 million operations. Although this is tractable, in the future we hope to scale up to modeling complex freeway networks with $L \sim 100$ detectors and $K \sim 5$ states, so we wanted to examine the performance of approximate inference. We tried both particle particle filtering (Section 5.2) and the BK algorithm (Section 4.2.1).

For particle filtering, we used the optimal proposal distribution, and between $N_s = 100$ and $N_s = 1000$ particles. Note that our samples are complete paths from $t = 1, \ldots, T$, so we can easily estimate the smoothed joint posterior $P(X_{\pi,t-1}, X_{l,t}|y_{1:T})$, where $\pi = (l-1, l, l+1)$ are the parents of l; we need this quantity to compute the expected sufficient statistics for EM. For BK, we used the fully factorized approximation, with one variable per cluster.

To compare the accuracy of PF and BK relative to exact inference, we computed the smoothed posterior marginal estimates $P(X_{l,t}|y_{1:T})$ using each of the methods on each of the test sequences, and using the estimated parameters. The results for test sequence 2 using the 4-5pm model are shown in Figure 6.13, using parameters estimated using EM/PF. BK yields posterior estimates that are indistinguishable from exact inference to at least three decimal places. PF yields a noisier estimate, but it is still very accurate: define $\Delta_{l,t}$ to be the L_1 difference of the estimates computed using exact and PF; then the empirical mean of this quantity is 0.0139 ± 0.0093 for 100 particles, and 0.0135 ± 0.0028 for 1000 particles. We see that using more particles slightly increases the accuracy and reduces the variance, but it seems that 100 particles is sufficient. The reason for this is the near-deterministic nature of the transitions (see below) and the informativeness of the observations.

Since the inference algorithms perform similarly, we expect the estimated parameters to be similar, too. This is indeed the case for the μ and σ parameters of the observation model, where the differences are not statistically significant (even using only 100 particles). PF does a poorer job at estimating the transition parameters, however, due to the fact that it only counts 100 sample paths per sequence. The total normalized L1 error is 4.9 for BK and 8.0 for PF. Using more particles would obviously help.

In addition to accuracy, speed is a major concern. A single E step takes about 1 second/slice using exact inference, about 1.3 s/slice using BK, and about 0.1 s/slice using PF with 100 particles.[8] The reason that BK is slower than exact (in this case) is because of the high constant factors, due to the complexity of the algorithm, and especially the need to perform the projection (marginalisation) step. Of course, the asymptotic complexity of BK is linear in L, while exact inference is exponential in L, so it is clear that for larger models, BK will rapidly become faster than exact.

The learned model

To aid interpretability of the parameters, we initialised the means for state 0 (congestion) to be 40 mph, and for state 1 (free flow) to be 60 mph. All other parameters were initialised randomly. We ran EM until the change in log-likelihood was less than 10^{-4}, which usually took 10–20 iterations. Some of the learned μ and σ values (using exact inference) are shown in Figure 6.14. The parameters for the models for 3-4pm, 4-5pm and 5-6pm are all very similar; we call this the "rush-hour" model. The parameters for the models for 2-3pm and 6-7pm are also very similar; we call this the "offpeak" model.

It is clear that when the traffic is slow, the variance is high, but when the traffic is fast, the variance is low. It is also clear that congestion gets worse as one approaches location 10, which corresponds to the part of I-880 that is near the San Mateo Bridge, a notorious bottleneck. Thus the learned μ and σ values seem sensible. Unfortunately, we were not able to intrepret the transition parameters in any meaningful way. The estimated parameter values are fairly insensitive to the initialisation and the inference algorithm used in the E step.

One of the advantages of a generative model is that we can simulate future traffic patterns. A typical sample drawn from this model is shown in Figure 6.15. We see that this resembles the training data; in particular, the model is capable of generating the triangular shape.

Using the model for prediction

Once trained, we can use the model to do online prediction. For exact inference, we first compute $\alpha_{t+\Delta|t} = P(X_{t+\Delta}|y_{1:t}) = M^{\Delta}\alpha_{t|t}$, where M is the transition matrix of the equivalent HMM; we then compute $P(Y_{t+\Delta}|y_{1:t})$, which is a mixture of Gaussians with $\alpha_{t+\Delta|t}$ as the mixing weights. See Figure 6.16 for an example. We compared these predictions to the naive approach of predicting that the future is the same as the present, $\hat{y}_{t+\Delta} = y_t$, for leads up to 20 minutes ahead. For the sequence in Figure 6.16, the rms error is 10.83 for the naive method and 9.76 for the model-based method. (We are ignoring the predicted σ, i.e., the confidence in the prediction, which is only available for the model-based approach.) Other sequences give similar results. It is clear from these numbers, and from the figure, that our predictions are not very accurate. We discuss ways to improve the model in the next section.

Discussion

Perhaps the most serious problem with our approach is that we have learned a separate model for each 1 hour period between 2-7pm, making it tricky to predict across boundaries. One approach would be to use

[8] Jaimyoung Kwon implemented PF in S-Plus on a 400 MHz PC. Kevin Murphy implemented exact inference and BK in Matlab on a Sun Ultra. The latter code is part of BNT. These times were measured using sequences of length $T = 60$.

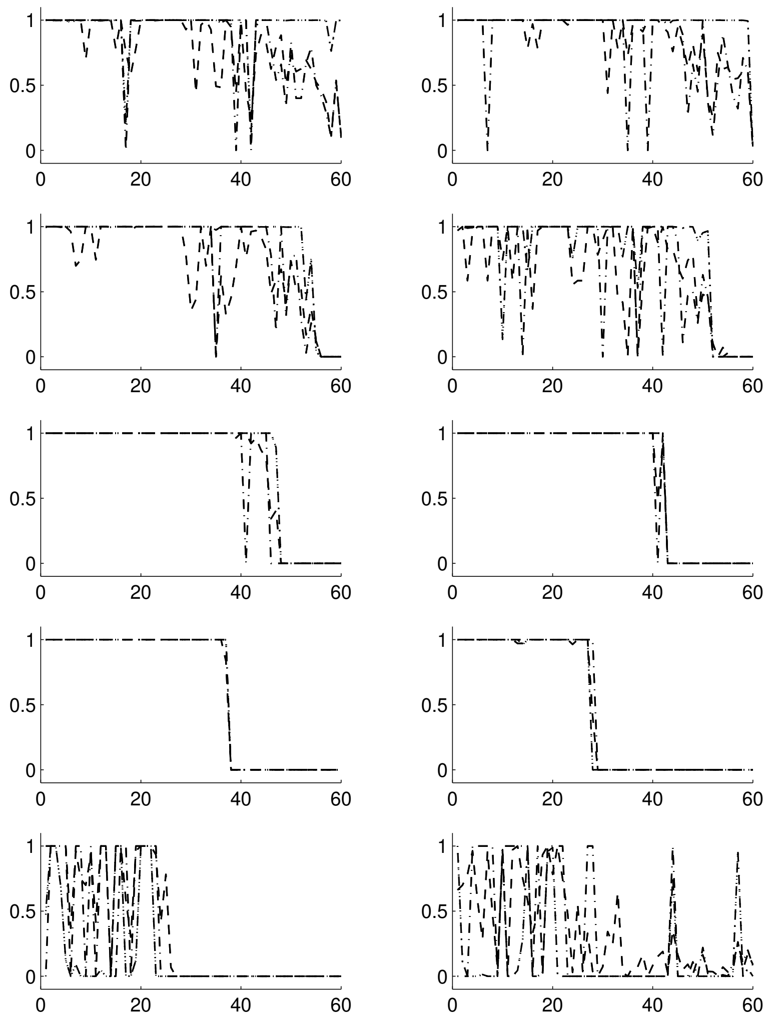


Figure 6.13: Posterior estimates of the probability of being in the free-flow state for the 4-5pm model on test sequence 2. Dotted lines represent exact/BK (which are indistinguishable at this resolution), dot-dash is PF with 100 particles, and dash-dash is PF with 1000 particles. Plots denotes locations 1 to 10 in left-right, top-bottom order. Notice how the change from free-flow to congested takes place earlier in the downstream locations.

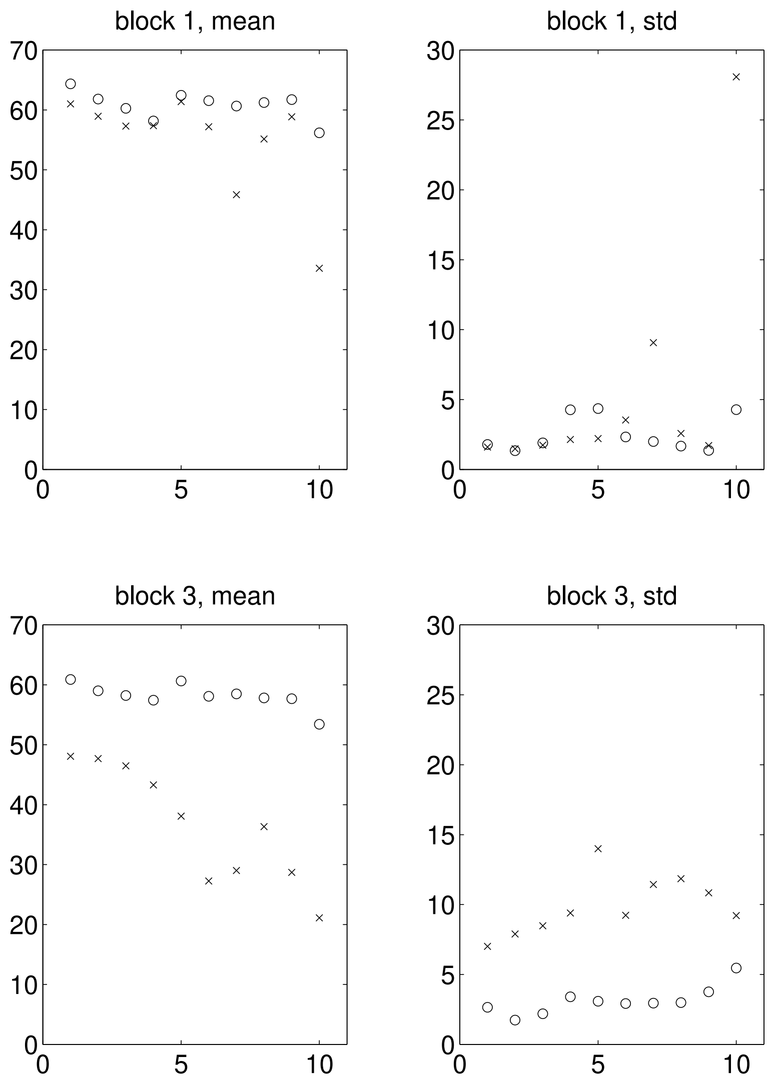


Figure 6.14: Maximum likelihood estimates of the mean and standard deviations of the models for block 1 (2-3 pm) and block 3 (4-5 pm). Crosses refer to the congested state, circles to free-flow.

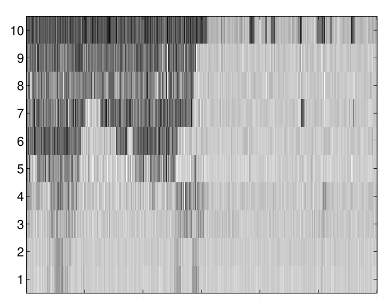


Figure 6.15: Data sampled from the learned model for 4-5pm.

the posterior from the previous model as the prior for the next. Alternatively, we could fit a single (mixture) model, by adding an extra hidden variable to each time slice to represent the current regime; all the other variables could then be conditioned on this. (In this case, the fully factorized version of BK takes $O(TLK^5)$,

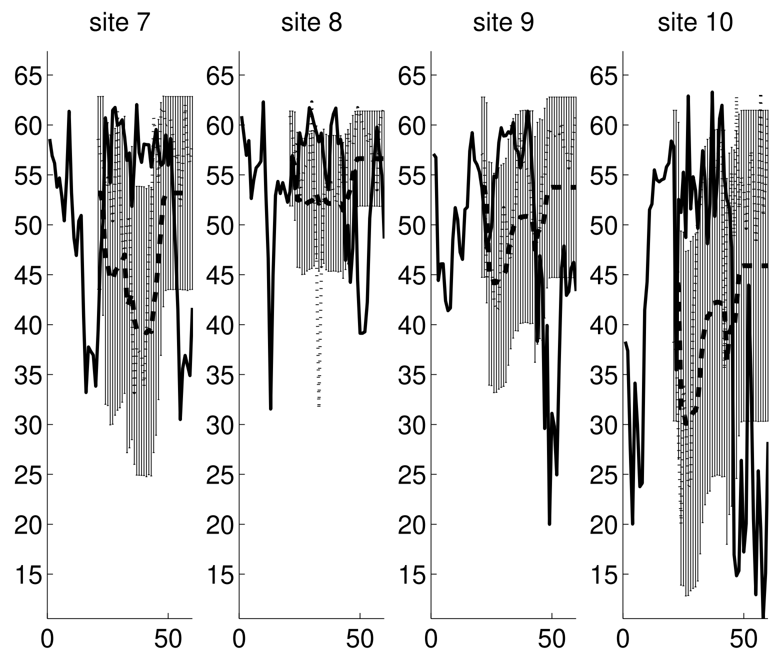


Figure 6.16: 20 minute-ahead predictions. Solid is the truth, dotted is the naive prediction, dashed (in the middle of the error bars) uses the model.

since the maximum clique size is 5.) Such a switching model could capture periodic non-stationarities. The number of regimes could be chosen using cross validation, although our results suggest that two might be sufficient, corresponding to rush-hour and off-peak.

We could also choose the number of hidden states for each location based on cross-validation. However, it is clear that $K = 2$ is inadequate, since it is incapable of distinguishing whether congestion is increasing or decreasing. It would be straightforward to use $K > 2$, or to make the model second-order Markov by adding longer distance dependencies, or to add extra variables to represent the (sign of the) derivative of the speed.

A third weakness is our assumption of Gaussian noise on the observations. Sensor failures, such as those shown in Figure 6.11, clearly invalidate this assumption. We can use a mixture of Gaussians as the noise model to handle this.

In the future, we plan to try using $K > 2$ with the switching model. We will also include a deterministic node that encodes the current time; thus predictions will be based both on historical patterns (using the time node) and the current state of the system (using the other hidden nodes). Ultimately, our goal is to predict travel time, rather than just velocity. We plan to build a model that can take as input the belief state about the current conditions, and combine it with historical (supervised) data, to provide a real-time travel forecast engine.

6.2.5 Online parameter estimation and model selection for regression

Here we discuss how to do online learning in the context of linear and non-linear regression. This section is based on [Jor02] and [AdFD00].

Linear regression

Suppose we want to recursively (i.e., sequentially, or online) estimate the coefficients, α, in a linear regression model, where we assume the noise level, R, is known. This can be modelled as shown in Figure 6.17. The

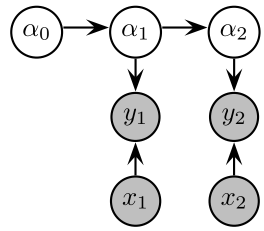


Figure 6.17: A DBN for the recursive least squares problem.

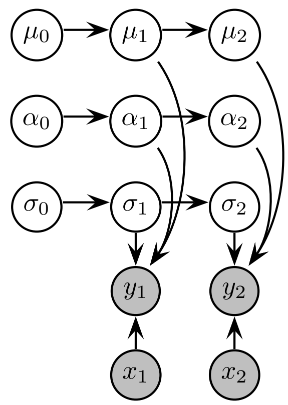


Figure 6.18: A DBN for sequential Bayesian non-linear regression.

CPDs are as follows:

$$
\begin{aligned}
P(\alpha_0) &= \mathcal{N}(\alpha_0; 0, \infty I) \\
P(y_t|x_t, \alpha_t) &= \mathcal{N}(y_t; x_t'\alpha_t, R) \\
P(\alpha_t|\alpha_{t-1}) &= \mathcal{N}(\alpha_t; \alpha_{t-1}, 0)
\end{aligned}
$$

We do not need to specify the CPD for x_t, since in linear regression, we assume the input is always known, so it suffices to use a conditional likelihood model. The infinite variance for α_0 is an uninformative prior. The zero variance for α_t reflects the fact that the parameter is constant.

We can perform exact inference in this model using the Kalman filter, where α_t is the hidden state, and we use a time-varying output matrix $C_t = x_t'$; we set the transition matrix to $A = I$, the transition noise to $K = 0$, and the observation noise to R. Now recall the Kalman filter equations from Section 3.6.1:

$$
\begin{aligned}
x_{t|t} &= Ax_{t|t-1} + K_t(y_t - CAx_{t-1|t-1}) \\
K_t &= V_{t|t}C'R^{-1}
\end{aligned}
$$

(This equation for K_t can be derived using the matrix inversion lemma.) Applied to this particular problem, we have

$$
\hat{\alpha}_{t|t} = \alpha_{t-1|t-1} + V_{t|t}x_tR^{-1}(y_t - x_t'\alpha_{t-1|t-1}) = \alpha_{t-1|t-1} + V_{t|t}R^{-1}(y_t - x_t'\alpha_{t-1|t-1})x_t
$$

This equation, plus the update for $V_{t|t}$, is known as the recursive least squares (RLS) algorithm; if we approximate $V_{t|t}R^{-1}$ by a constant, this reduces to the least mean squares (LMS) algorithm [Jor02].

Non-linear regression

Now consider a non-linear regression model [AdFD00]:

$$
y_t = \sum_{j=1}^{k} a_{j,t}\phi(||x_t - \mu_{j,t}||) + b_t + \beta_t'x_t + n_t
$$

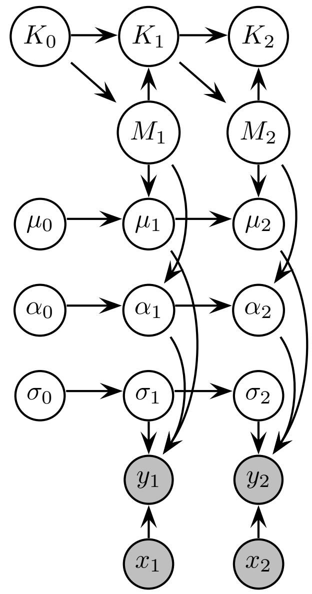


Figure 6.19: A DBN for sequential Bayesian model selection.

where $n_t \sim \mathcal{N}(0, \sigma_t)$ is a noise term and $\phi(||x_t - \mu_{j,t}||)$ is e.g., a radial basis function (RBF); if $k = 0$, this reduces to a linear regression model. We can write this in vector-matrix form as follows:

$$y_t = D(\mu_t, x_t)\alpha_t + n_t$$

where $\alpha_t = (a_t, b_t, \beta_t)$ are all the weights. The resulting DBN is shown in Figure 6.18. If we allow the parameters to change over time (modelled by a random walk), the CPDs are

$$\begin{aligned}
P(\mu_t|\mu_{t-1}) &= \mathcal{N}(\mu_t; \mu_{t-1}, \delta_\mu I) \\
P(\alpha_t|\alpha_{t-1}) &= \mathcal{N}(\alpha_t; \alpha_{t-1}, \delta_\alpha I) \\
P(\log \sigma_t| \log \sigma_{t-1}) &= \mathcal{N}(\log \sigma_t; \log \sigma_{t-1}, \delta_\sigma I) \\
P(y_t|x_t, \alpha_t, \mu_t) &= \mathcal{N}(y_t; D(\mu_t, x_t)\alpha_t, \sigma_t)
\end{aligned}$$

(I have omitted the priors at time 1 for simplicity.)

We can do inference in this model using particle filtering.[9] In fact, since this model is linear in α_t, we only need to sample μ_t and σ_t, and can integrate out α_t using a Kalman filter: see Section 5.3. Alternative, fully deterministic approximations are discussed in Section 4.4.2.

Non-linear regression with online model selection

Now we will let the number of basis functions (and hence the size of the state space) vary over time [AdFD00]. Let K_t be the number of bases at time t, and let M_t be a random variable with five possible values: B=birth, D=death, S=split, M=merge and N=no-change. These specify the ways in which K_t can increase/decrease by 1 at each step. Birth means we create a new basis function; its position can depend on the previous basis function centers, μ_{t-1}, as well as the current data point, x_t; death means we remove a basis function at random, split means we split one center into two, and merge means we combine two centers into one. We

[9] If we do not add noise to the parameters (i.e., if $\delta = 0$), the particle filter will not work; see [LW01] for a possible fix.

enforce that $0 \leq K_t \leq K_{max}$ at all times. The DBN is shown in Figure 6.19, and the CPDs are as follows.

$$
\begin{aligned}
P(M_t = (B, D, S, M, N)|K_{t-1} = k) &= \begin{cases} (\frac{1}{2}, 0, 0, 0, \frac{1}{2}) & \text{if } k = 0 \\ (\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, 0, \frac{1}{4}) & \text{if } k = 1 \\ (0, \frac{1}{3}, 0, \frac{1}{3}, \frac{1}{3}) & \text{if } k = K_{max} \\ (\frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}) & \text{otherwise} \end{cases} \\
P(K_t = k'|K_{t-1} = k, M_t = m) &= \begin{cases} \delta(k', k+1) & \text{if } m = B \text{ or } m = S \\ \delta(k', k-1) & \text{if } m = D \text{ or } m = M \\ \delta(k', k) & \text{if } m = N \end{cases} \\
P(\mu_t|\mu_{t-1}, M_t = m) &= \begin{cases} \mathcal{N}(\mu_t; \mu_{t-1}, \delta_\mu I) & \text{if } m = N \\ \mathcal{N}(\mu_t; \text{birth}(\mu_{t-1}), \cdot) & \text{if } m = B \\ \mathcal{N}(\mu_t; \text{death}(\mu_{t-1}), \cdot) & \text{if } m = D \\ \mathcal{N}(\mu_t; \text{split}(\mu_{t-1}), \cdot) & \text{if } m = S \\ \mathcal{N}(\mu_t; \text{merge}(\mu_{t-1}), \cdot) & \text{if } m = M \end{cases} \\
P(\alpha_t|\alpha_{t-1}, M_t = m) &= \begin{cases} \mathcal{N}(\alpha_t; \alpha_{t-1}, \delta_\alpha I) & \text{if } m = N \\ \mathcal{N}(\alpha_t; \text{grow}(\alpha_{t-1}), \cdot) & \text{if } m = B \text{ or } m = S \\ \mathcal{N}(\alpha_t; \text{shrink}(\alpha_{t-1}), \cdot) & \text{if } m = D \text{ or } m = M \end{cases} \\
P(\log \sigma_t| \log \sigma_{t-1}) &= \mathcal{N}(\log \sigma_t; \log \sigma_{t-1}, \delta_\sigma I) \\
P(y_t|x_t, \alpha_t, \mu_t) &= \mathcal{N}(y_t; D(\mu_t, x_t)\alpha_t, \sigma_t)
\end{aligned}
$$

The function birth(μ_{t-1}) takes in the vector of RBF centers and adds a new one to the end, according to some heuristic. In principle, this heuristic could depend on x_t as well (not shown). Call the new center μ_{birth}. The confidence associated with μ_{birth} is yet another free parameter; suppose it is Q_{birth}; then the full CPD would be

$$P(\mu_t|\mu_{t-1}, M_t = B) = \mathcal{N}\left(\mu_t; \begin{pmatrix}\mu'_{t-1} & \mu'_{\text{birth}}\end{pmatrix}', \begin{pmatrix} 0 & 0 \\ 0 & Q_{\text{birth}} \end{pmatrix}\right)$$

The 0 terms in the covariance matrix do not mean we are certain about the locations of the other centers, only that we have not introduced any extra noise, i.e.,

$$\text{Cov}(\mu_t|y_{1:t}, M_t = B) = \begin{pmatrix} \text{Cov}(\mu_{t-1}|y_{1:t-1}) & 0 \\ 0 & Q_{\text{birth}} \end{pmatrix}$$

Of course, we could relax this assumption. If we were full-blooded Bayesians, we would now add priors to all of our parameters (such as Q_{birth}, δ_α, etc.); these priors would in turn have their own (hyper-)parameters; we would continue in this way until the resulting model is insenstive to the parameters we choose. This is called hierarchical Bayesian modelling.

Despite the apparent complexity, it is actually quite easy to apply particle filtering to this model. [AdFD00] suggest a more complex scheme, which combines particle filtering with reversible jump MCMC [Gre98]. (Regular MCMC can only be applied if the state-space has a fixed size.) We can use the simpler method of particle filtering because we included M_t in the state-space.

Appendix A

Graphical models: representation

A.1 Introduction

Probabilistic graphical models are graphs in which nodes represent random variables, and the (lack of) arcs represent conditional independence assumptions. Hence they provide a compact representation of joint probability distributions. For example, if we have N binary random variables, an "atomic" representation of the joint, $P(X_1, \ldots, X_N)$, needs $O(2^N)$ parameters, whereas a graphical model may need exponentially fewer, depending on which conditional assumptions we make. This can help both inference and learning, as we explain below.

There are two main kinds of graphical models: undirected and directed. Undirected graphical models, also known as Markov networks or Markov random fields (MRFs), are more popular with the physics and vision communities. (Log-linear models are a special case of undirected graphical models, and are popular in statistics.) Directed graphical models, also known as Bayesian networks (BNs)[1], belief networks, generative models, causal models, etc. are more popular with the AI and machine learning communities. It is also possible to have a model with both directed and undirected arcs, which is called a chain graph.

Figures A.1 and A.2 give a summary of the relationship between various popular graphical models. Please see [CDLS99, Jor99, Jen01, Jor02] for more information,

A.2 Undirected graphical models

The conditional independence statements encoded by an MRF are easy to state: $X_A \perp X_B | X_C$ iff all paths between all nodes in A and all nodes in B are blocked by some node in C, i.e., there is some intervening $c \in C$ on every path between every $a \in A$ to every $b \in B$. This is called the global Markov property. This implies that a single node X_i is independent of all the other nodes in the graph given its neighbors (which are called X_i's Markov blanket); this is known as the local Markov property.

The joint distribution of an MRF is defined by

$$P(x) = \frac{1}{Z} \prod_{C \in \mathcal{C}} \psi_C(x_C)$$

where $\mathcal{C}$ is the set of maximal cliques[2], $\psi_C(x_C)$ is a potential function (a positive, but otherwise arbitrary, real-valued function) on the clique x_C, and Z is the normalization factor

$$Z = \sum_x \prod_{C \in \mathcal{C}} \psi_C(x_C).$$

[1] Note that, despite the name, Bayesian networks do not necessarily imply a commitment to Bayesian methods; rather, they are so called because they use Bayes' rule for inference: see Appendix B.

[2] A clique in a graph is a set of nodes all of which are interconnected. A maximal clique is one which is not a proper subset of any other clique in the graph. Sometimes the term clique'is used to mean maximal clique , but we will find it useful to distinguish maximal from non-maximal.

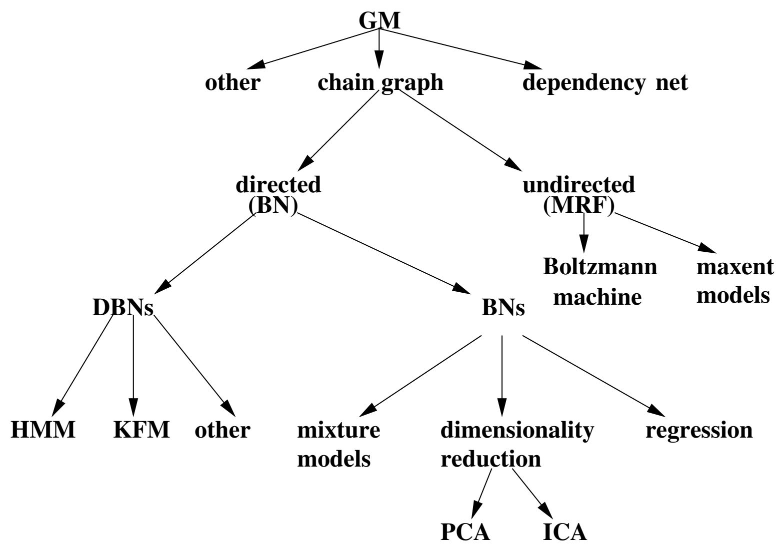


Figure A.1: A partial hierarchy relating different kinds of graphical models. Abbreviations used: GM = graphical model, BN = Bayes net, MRF = Markov Random field, DBN = dynamic Bayes net, HMM = hidden Markov model, KFM = Kalman filter model, PCA = principal components analysis, ICA = independent components analysis, maxent = maximum entropy (see Section A.2.2). Dependency nets are defined in [HCM$^+$00].

For example, the model in Figure A.3 represents

$$P(X_{1:5}) = \frac{1}{Z}\psi(X_1, X_2, X_3)\psi(X_3, X_4)\psi(X_4, X_5)$$

The potential functions may be defined in terms of subsets of their arguments, e.g.,

$$\psi(X_1, X_2, X_3) = \psi(X_1, X_2)\psi(X_2, X_3)\psi(X_1, X_3)$$

If all potential functions are defined on pairs of nodes (i.e., edges), the model is called a pairwise MRF, or MRF2. Another example is shown in Figure A.4. In this case, the joint distribution is

$$P(x, y) \propto \Psi(x_1, x_2)\Psi(x_1, x_3)\Psi(x_2, x_4)\Psi(x_3, x_4)\prod_{i=1}^{4}\Psi(x_i, y_i)$$

In low-level vision problems (e.g., [GG84, FPC00]), the X_i's are usually hidden, and each X_i node has its own "private" observation node Y_i, as in Figure A.4. The potential $\Psi(x_i, y_i) = P(y_i|x_i)$ encodes the local likelihood; this is often a conditional Gaussian, where Y_i is the image intensity of pixel i, and X_i is the underlying (discrete) scene "label".

Lattice MRFs are identical to what physicists call "Potts models". They define the probability distribution over nodes using the Boltzman distribution:

$$P(x_{1:N}) = \frac{1}{Z}e^{-E(x_{1:N})/T}$$

where T is the temperature and the energy is defined by

$$E(x_{1:N}) = -\sum_{(ij)} J_{ij}(x_i, x_j) - \sum_i h_i(x_i)$$

The notation $\sum_{(ij)}$ means summing over all pairs i, j s.t., $i \neq j$. If we set $J_{ij}(x_i, x_j) = \ln \Psi_{i,j}(x_i, x_j)$ and $h_i(x_i) = \ln \Psi(x_i, y_i)$, this becomes identical to a pairwise MRF. The Ising model is a special case of the Potts model with binary random variables. The Boltzmann machine is also a binary pairwise MRF, although the structure need not be a 2D lattice.

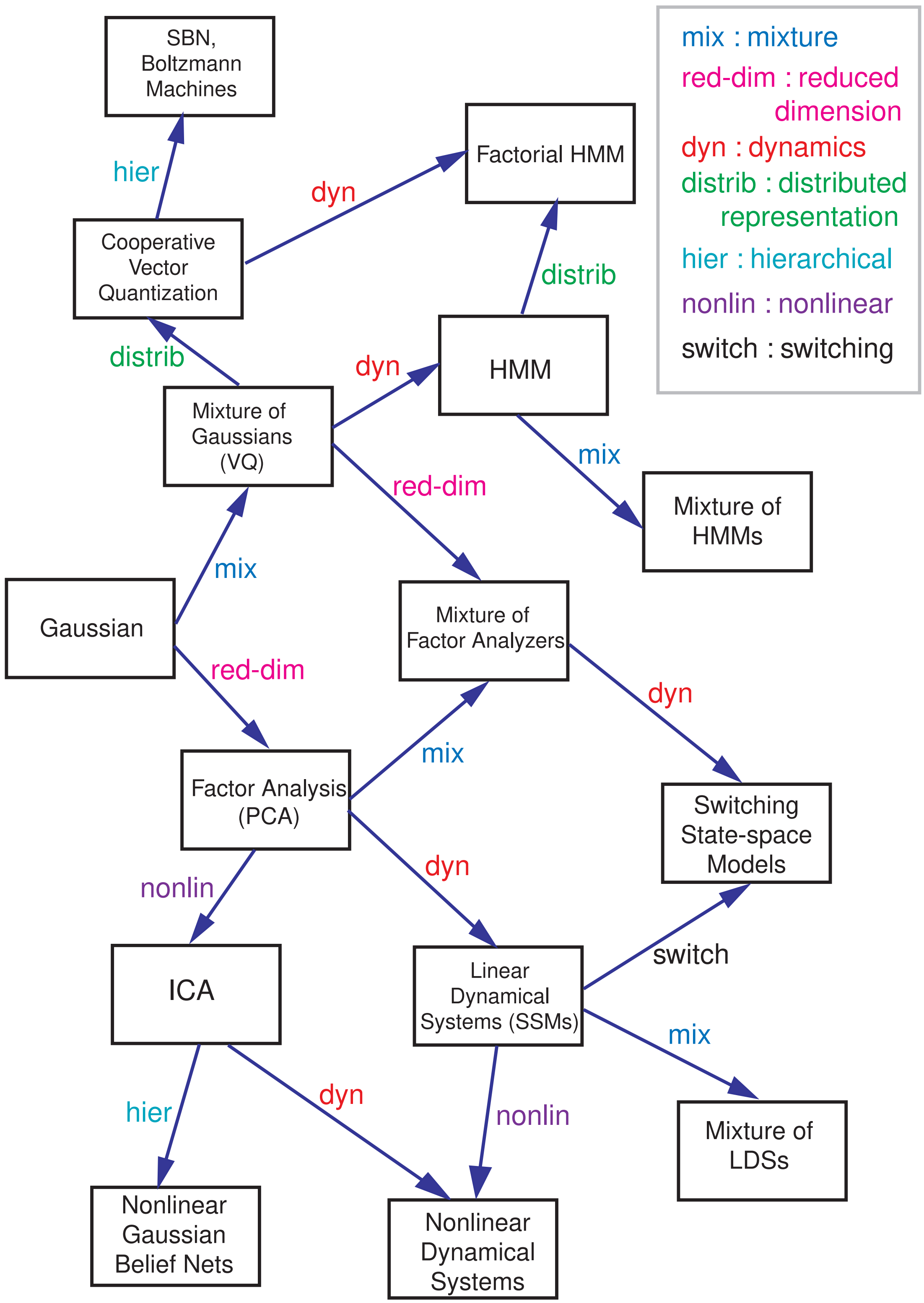


Figure A.2: A generative model for generative models. Thanks to Sam Roweis and Zoubin Ghahramani for providing this figure.

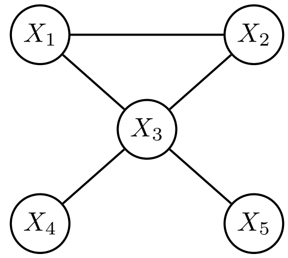


Figure A.3: A simple Markov random field.

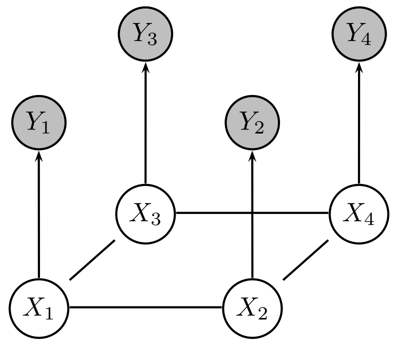


Figure A.4: A pairwise MRF defined on a lattice. Each hidden variable X_i has its own "private" observed child Y_i. Technically, this is a chain graph since the arcs from X_i to Y_i are directed.

A.2.1 Representing potential functions

For discrete random variables, we can represent the potential functions as tables. To keep the number of parameters tractable, it is common to make factorization assumptions, e.g.,

$$\psi(X_1, X_2, X_3) = \psi(X_1, X_2)\psi(X_2, X_3)\psi(X_1, X_3).$$

For continuous random variables, we can model the potential function as a Gaussian or mixture of Gaussians. For mixed discrete-continuous, we can again use a mixture of Gaussians.

A.2.2 Maximum entropy models

When the discrete random variables can have a large number of values (e.g., if they represent words), it can be useful to represent joint probability distributions in terms of features, f_i:

$$P(X_1, \ldots, X_N) = \frac{1}{Z(\lambda)} \exp\left(\sum_{i=1}^{F} \lambda_i f_i(X_{1:N})\right)$$

where $Z(\lambda) = \sum_{x_{1:N}} \exp(\sum_{i=1}^{F} \lambda_i f_i(x_{1:N}))$. This is called a Gibbs distribution, or a log-linear model. We can represent this as an MRF where we add connections between nodes which co-occur in the domain of the same feature. However, the graph will not reflect the fact that the potentials may have a restricted form. For example, consider the following log-linear model:

$$\log P(X_1, X_2, X_3) = \lambda_1 f_1(X_1, X_2) + \lambda_2 f_2(X_1, X_3) + \lambda_3 f_1(X_1, X_3) - \log Z(\lambda)$$

The corresponding graph is a triangle (clique) on X_1, X_2, X_3, which does not reflect the fact the potential does not have any three-way interaction terms. The factor graph representation discussed in Section A.4 makes such restrictions graphically explicit.

It turns out that maximum likelihood estimation of the above density is equivalent to maximum entropy estimation of an arbitrary distribution p subject to the expectation constraints $\sum_x p(x) f_i(x) = \alpha_i$, where

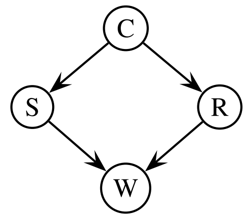


Figure A.5: A simple Bayes net. C=cloudy, S=Sprinkler, R=rain, W=wet grass. The wet grass can either be caused by rain or by a water sprinkler. Clouds make it less likely the sprinkler will turn on, but more likely it will rain.

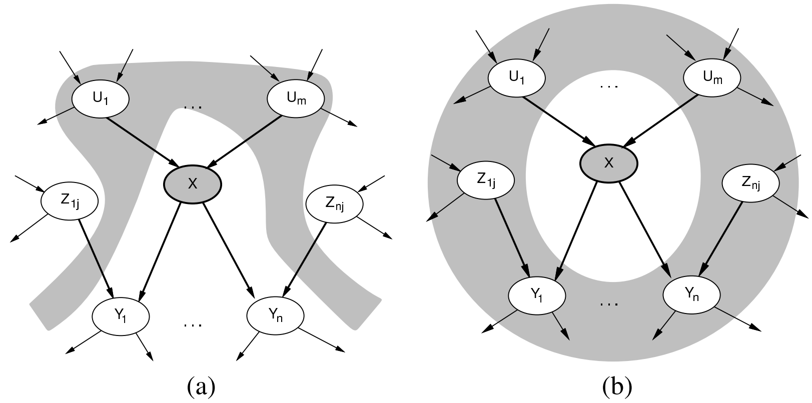


Figure A.6: Local Markov properties for Bayes nets. (a) A node X is conditionally independent of its non-descendants (e.g., $Z_1j, \ldots, Z_nj$) given its parents $U_1, \ldots, U_m$. (b) A node X is conditionally independent of all other nodes in the network given its Markov blanket (shaded). From [RN02].

the α_i are known constants (e.g., empirical counts). See [Ber] and [Jor02, ch.19] for details. Such models are popular in the text processing community (see e.g., [PPL97, LMP01]). See also [Goo01] for a way to fit maxent models by using EM training of HMMs.[3]

A.3 Directed graphical models

In a directed graphical model (i.e., a Bayesian network), an arc from A to B can be informally interpreted as indicating that A "causes" B. Hence directed cycles are disallowed, i.e., the graph is a directed acyclic graph (DAG). See Figure A.5 for an example.

We now define the semantics more formally. As in an MRF, a node in a BN is independent of all the other nodes in the graph given its Markov blanket. However, in the case of a BN, the Markov blanket of a node is the node's parents, children and children's parents (see Figure A.6).

The reason why we include the children's parents can be explained by considering the example in Figure A.5 but without the C node, so what is left is just the "v-structure" $S \to W \leftarrow R$. Let us suppose all nodes are binary, i.e., have values in $\{0, 1\}$. Consider the Rain node, and suppose it is true ($R = 1$); if we observe that the grass is wet ($W = 1$), then it is now less likely that the sprinkler is on than if we did not know it was raining (i.e., $P(S = 1|W = 1, R = 1) < P(S = 1|W = 1)$), since the rain has "explained away" the fact that the grass is wet [Pea88]. Hence $R \not\perp S|W$, i.e., R is correlated with its children's parents given its children. In an MRF, we would have $R \perp S|W$, since W separates S and R; however, in a BN, we must take into account the directionality of the arcs; this gives rise to the notion of d-separation (d for directed): see Section A.3.1. Hence when converting a BN to an MRF, we must connect together "unmarried" parents (such as S and R) who share a common child; this is called "moralization".

[3] Unfortunately, these HMMs contain silent loops, making it hard to represent them as DBNs.

Dpa	Cpa	Name	CPD
-	-	Multinomial	$P(Y=j)=\pi(j)$
i	-	Cond. multinomial	$P(Y=j\|X=i)=A(i,j)$
-	u	Softmax	$P(Y=j\|U=u)=\sigma(u,W,j)$
i	u	Cond. softmax	$P(Y=j\|U=u,X=i)=\sigma(u,W_i,j)$

Table A.1: Some common CPDs for when we have 0 or 1 discrete parents (Dpa), 0 or 1 continuous parents (Cpa), and the child is discrete. X represents the discrete parent, U represents the continuous parent, and Y represents the discrete child. "cond" stands for conditional. $\sigma(u, W, j)$ is is the softmax function: see Section A.3.2. If we have more than one discrete parent, they can all be combined to make one discrete "mega" parent with exponentially many values; more parsimonious representations will be discussed in Section A.3.2.

Dpa	Cpa	Name	CPD
-	-	Gaussian	$P(Y=y)=\mathcal{N}(y;\mu,\Sigma)$
i	-	Cond. Gaussian	$P(Y=y\|X=i)=\mathcal{N}(y;\mu_i,\Sigma_i)$
-	u	Linear Gaussian	$P(Y=y\|U=u)=\mathcal{N}(y;Wu+\mu,\Sigma)$
i	u	Cond. linear Gaussian	$P(Y=y\|U=u,X=i)=\mathcal{N}(y;W_iu+\mu_i,\Sigma_i)$

Table A.2: Some common CPDs for when we have 0 or 1 discrete parents (Dpa), 0 or 1 continuous parents (Cpa), and the child is continuous. X represents the discrete parent, U represents the continuous parent, and Y represents the continuous child. "cond" stands for conditional. If we have more than one continuous parent, they can all be combined into one large vector-valued parent.

An alternative definition of independence for BNs, known as the directed local Markov property, is the following: a node is conditionally independent of its non-descendants given its parents. If we topologically order the nodes (parents before children) as $1, \ldots, N$, this means we can write the joint distribution as follows:

$$\begin{aligned}
P(X_1, \ldots, X_N) &= P(X_1)P(X_2|X_1)P(X_3|X_1,X_2)\cdots P(X_N|X_1,\ldots,X_{N-1}) \\
&= \prod_{i=1}^{N} P(X_i|X_{1:i-1}) \\
&= \prod_{i=1}^{N} P(X_i|\mathrm{Pa}(X_i))
\end{aligned}$$

where $X_{1:i-1} = (X_1, \ldots, X_{i-1})$ and $\mathrm{Pa}(X_i)$ are the parents of node X_i. (This is known as the directed factorization property.) The first line follows from the chain rule of probability, the second line is the same as the first, and the third line follows because node X_i is independent of all its ancestors, $X_{1:i-1}$, given its parents. For example, the joint implied by Figure A.5 is

$$P(C,S,R,W) = P(C)P(S|C)P(R|C)P(W|S,R)$$

The function $P(X_i|\mathrm{Pa}(X_i))$ is called node i's conditional probability distribution (CPD). This can be an arbitrary distribution (see Tables A.1 and A.2 for some possibilities). In Section A.3.2, we discuss representations of CPDs which require fewer parameters; we will use these extensively in Chapter 2.

A.3.1 Bayes ball

We now discuss how to infer global indepenence relationships from a BN using the "Bayes ball" algorithm [Sha98], which is equivalent to d-separation [Pea88]. $X_A \perp X_B | X_C$ iff an imaginary ball, starting at any node in A, can not reach any node in B, by following the rules in Figure A.7, where shaded nodes correspond to nodes in C. Curved dotted arrows mean that the ball is turned back (blocked) by a node; straight arrows means the ball can pass through. For example, in Figure A.5, we have $S \perp R | C$, since the path via C is blocked (since C is shaded) and the path via W is blocked (since W is not shaded). However, $S \not\perp R | C, W$, and $S \not\perp R | W$, since W will now be shaded (this is the explaining away phenomenon).

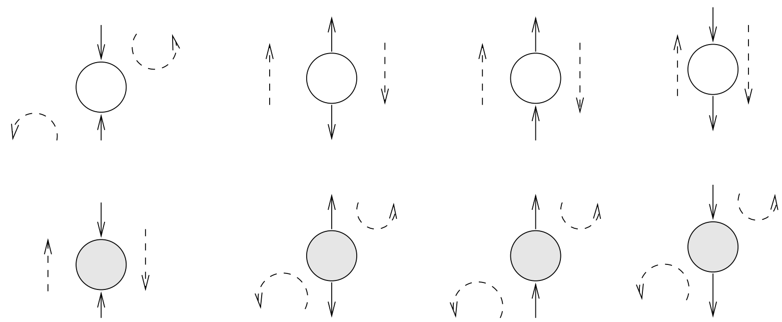


Figure A.7: The rules of Bayes ball. Shaded nodes are conditioned on. See text for details.

A.3.2 Parsimonious representations of CPDs

If a node Y has many discrete parents, $X_1, \ldots, X_N$, the number of parameters needed to specify the CPD is $O(K^N)$, where we assume each X_i can have K values. In this section we consider more parsimonious representations, i.e., ones which require fewer parameters. Such representations are always easier to learn[4], are often easier to intepret, and sometimes lead to faster inference (see Section B.6).

Mixtures of small multinomials

If Y is discrete, we can represent $P(Y|X_{1:N})$ as a mixture of smaller multinomials, as in a mixed-memory Markov model (Section 2.3.5). In particular, we can define a (discrete) multiplexer CPD as follows, where S is a switching parent:

$$P(Y = y|X_{1:N} = x_{1:N}, S = i) \stackrel{\text{def}}{=} P(Y = y|X_i = x_i) \stackrel{\text{def}}{=} A_i(y, x_i)$$

Since S is hidden, we can marginalize it out to get the final CPD

$$P(y|x_{1:N}) = \sum_i P(S = i) A_i(y, x_i)$$

If we assume X_i and Y can have K values, this CPD requires NK^2 parameters to represent, whereas a full multinomial would require K^{N+1}.

Context-specific independence

If Y is discrete, we can define $P(Y|X_{1:N})$ using a classification tree, by storing a histogram over Y's values in each leaf. If Y is continuous, we can define $P(Y|X_{1:N})$ using a regression tree, by storing the conditional mean and variance in each leaf. In both cases, if the tree is not full, we will need less than $O(K^N)$ parameters.

Any tree can be "flattened" into a table. Parameter learning in this flattened table is equivalent to parameter learning in the tree, if we exploit the fact that some of the entries should be tied. If the tree structure is unknown, it can be learned using standard decision tree methods. Tree-structured CPDs can also help structure learning [FG96].

In a non-full tree, not all parents are always relevant in determining the output (child) distribution; this property is called context-specific independence (CSI) [BFGK96]. (This just means a conditional independence relationship which only holds under certain instantiations of some of the conditioning variables.) We discuss how to exploit this for inference in Section B.6.2.

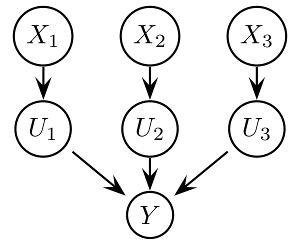


Figure A.8: A noisy-OR CPD, $P(Y|X_1, X_2, X_3)$, represented in terms of noisy hidden variables U_i and a deterministic OR gate.

Causal independence

Causal independence arises when a CPD has the special property that the contribution of each parent on the child can be computed in terms of an associative and commutative operator such as "or", "sum" or "max". A standard example is the noisy-OR distribution [Pea88]. The noisy-OR is applicable if the child Y and all its parents are binary (values in $\{0, 1\}$). A noisy-OR function is like a regular OR function, except some "connections" between the inputs (parents) and the output (child) might "fail". Failures are assumed to be independent, and to map parents which have value 1 to value 0 (to "suppress" parents which are on); q_i is the probability that the i'th input fails in this way. This is illustrated in Figure A.8. We define $P(U_i = 0|X_i = 1) = q_i$, $P(U_i = 0|X_i = 0) = 1$ and $P(y|u_1, \ldots, u_N) = \delta(y, u_1 \vee \cdots u_N)$. The overall CPD has the following distribution:

$$P(Y = 0|x_{1:N}) = \prod_{i:x_i=1} q_i = \prod_{i=1}^{N} q_i^{x_i} \tag{A.1}$$

It is common to add a "leak" term, which is a dummy parent which is always on; this represents "all other causes". Let q_0 to be the probability that the leak is suppressed. Since $P(Y = 1|X_{1:N}) = 1 - P(Y = 0|X_{1:N})$, we may write

$$P(y|x_{1:N}) = \left(1 - q_0 \prod_{i=1}^{N} q_i^{x_i}\right)^y \left(q_0 \prod_{i=1}^{N} q_i^{x_i}\right)^{1-y}$$

It is easy to generalize noisy-OR to noisy-MAX, etc. [Die93, Hec93, HB94, Sri93, MH97, RD98]. Such models have a number of parameters that is linear in the number of parents.

Log-concave CPDs

We define a log-concave CPD to be one which has the form

$$P(Y = y|x_{1:N}) = G_y(\sum_j \theta_j x_j)$$

where $\log G_y(\cdot)$ is a concave function. Such CPDs can be exploited by approximate variational inference [JJ96]. This class includes sigmoid CPDs [Nea92] and noisy-OR CPDs [Pea88], both defined for binary nodes. To see this, note that a sigmoid CPD is defined as

$$P(Y = 1|x_{1:N}) = \sigma(\sum_j \theta_j x_j)$$

[4] Parsimonious CPDs have lower sample complexity (require less data), but often have higher computational complexity (require more time). Specifically, for many parsimonious CPDs, the M step of EM cannot be solved in closed form; hence it is necessary to use iterative methods. For example, softmax CPDs can use IRLS (iteratively reweighted least squares), a kind of Newton method, and MLP (multi-layer perceptron) CPDs can use backpropagation (gradient ascent). The resulting algorithm is called generalized EM, and is still guaranteed to converge to a local maximum, but is usually much slower than regular EM. See Appendix C for details.

where $\sigma(x) = \frac{1}{1+e^{-x}}$ is the sigmoid (logistic) function [Jor95]. In other words,

$$P(Y = 1|x_{1:N}) = \left(\frac{1}{1+e^{-u}}\right)^y \left(\frac{1}{1+e^{u}}\right)^{1-y}$$

where $u = \theta' x$, since $1 - \sigma(u) = \frac{1}{1+e^u} = \sigma(-u)$. To express a noisy-OR CPD in this form, note that

$$\prod_j q_j^{x_j} = \exp \log \prod_j q_j^{x_j} = \exp \sum_j x_k \log q_j = e^{-\sum_j \theta_j x_j}$$

where we have defined $\theta_j = -\log q_j$. Hence the noisy-OR CPD can be written as

$$P(Y = 1|x_{1:N}) = (1 - e^{-u})^y (e^{-u})^{1-y}$$

Softmax

The softmax function [MN83] is a generalization of the sigmoid function to the case where the "output" has K possible values, instead of just two. (In statistics, it is called a multi-logistic regression.) We define

$$P(Y = j|U = u) = \sigma(u, W, j)$$

where

$$\sigma(u, W, j) = \frac{\exp(u' W_{:j})}{\sum_k \exp(u' W_{:k})}$$

and $W_{:k}$ is the k'th column of W. When the output is binary, the softmax function reduces to the logistic function:

$$\sigma(u, W, 1) = \frac{\exp(u' W_{:1})}{\exp(u' W_{:1}) + \exp(u' W_{:0})} = \frac{1}{1 + \exp(u'(W_{:0} - W_{:1}))} = \sigma(u' \tilde{w})$$

where $\tilde{w} = W_{:1} - W_{:0}$.

The softmax function takes a continuous vector u as input. To use it in the case of discrete parents, we must represent the values of the discrete parents as a vector. We therefore define $\hat{X}_i$ to be a bit vector of length K, with value 1 in the position corresponding to the value of X_i, and value 0 in all other positions (a one-of-K distributed representation); $\hat{X}$ is the concatentation of all the $\hat{X}_i$ vectors. Given this representation, we can represent $P(Y|X_{1:N})$ as a softmax even if some of the parents X_i are discrete [Pfl98].

Alternatively, we might want to use one or more of the discrete parents to specify what parameters to use (a conditional softmax function), e.g., $P(Y = j|X_1 = i, \tilde{X}_2 = x) = \sigma(x, W_i, j)$. In BNT, the optional parameter `dps_as_cts` to the softmax CPD specifies which discrete parents should be treated as continuous (converted to distributed form), and which should be used as indices.

Conditional linear Gaussian

We have already defined the conditional linear Gaussian (CLG) CPD to be

$$P(Y = y|U = u, X = i) = \mathcal{N}(y; W_i u + \mu_i, \Sigma_i)$$

If we have many discrete parents, we can convert them to continuous vectors: $P(Y = y|X_{1:N}) = \mathcal{N}(y; W\hat{X}, \Sigma)$. For example, suppose $N = 3$, X_1 and X_3 are binary, X_2 is ternary, and $Y \in \mathbb{R}^2$. We can think of W as the concatenation of N smaller weight matrices. If $X_1 = 1$, $X_2 = 3$, $X_3 = 2$, then $\tilde{X}_1 = \begin{pmatrix} 1 & 0 \end{pmatrix}$, $\tilde{X}_2 = \begin{pmatrix} 0 & 0 & 1 \end{pmatrix}$, and $\tilde{X}_3 = \begin{pmatrix} 0 & 1 \end{pmatrix}$, so the mean is given by

$$W\hat{X} = \begin{pmatrix} W^1_{11} & W^1_{12} & W^2_{11} & W^2_{12} & W^2_{13} & W^3_{11} & W^3_{12} \\ W^1_{21} & W^1_{22} & W^2_{21} & W^2_{22} & W^2_{23} & W^3_{21} & W^3_{22} \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} W^1_{11} \\ W^1_{21} \end{pmatrix} + \begin{pmatrix} W^2_{13} \\ W^2_{23} \end{pmatrix} + \begin{pmatrix} W^3_{12} \\ W^3_{22} \end{pmatrix}$$

Another way of writing this is
$$W\hat{X} = \sum_i W^i \hat{X}_i = \sum_i W^i(:, X_i)$$

where $W^i(:, X_i)$ is the X_i'th column of W^i. This representation was suggested for factorial HMMs in [GJ97], and is commin in the neural network field.

Other representations

It is straightforward to use any function approximator to define a CPD, e.g., multi-layer perceptrons (feed-forward neural networks) [Bis95], Gaussian processes [FN00], etc. However, it is not always easy to use them for efficient inference or learning.

A.4 Factor graphs

Factor graphs (fgraphs) [KFL01] provide a convenient representation that unifies directed and undirected graphical models. In a factor graph, there are two kinds of nodes, representing factors (local terms) and variables. Variable node X_i is connected to all factor nodes F_i which contain X_i in their domain. Hence factor graphs are bipartite. Variable nodes are represented by circles, and factor nodes by squares. See Figure A.9 for an example.

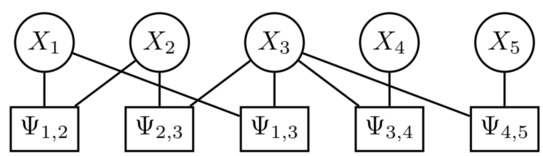


Figure A.9: Converting the MRF in Figure A.3 to a factor graph, where we have assumed $\psi(X_1, X_2, X_3) = \psi(X_1, X_2)\psi(X_2, X_3)\psi(X_1, X_3)$.

A factor graph specifies how a function of many variables can be decomposed into a set of local functions. For example, Figure A.9 specifies

$$P(X_{1:5}) \propto \Psi_{1,2}(X_1, X_2)\Psi_{2,3}(X_2, X_3)\Psi_{1,3}(X_1, X_3)\Psi_{3,4}(X_3, X_4)\Psi_{3,5}(X_3, X_5)$$

We can convert an fgraph back to an MRF by adding a link between all nodes that share the same factor. See Figure A.11 for an example. Applying this procedure to the fgraph in Figure A.9 recovers Figure A.3. Note that the MRF representation cannot graphically represent the factorized form that we have assumed for the potential on clique X_1, X_2, X_3

$$\psi(X_1, X_2, X_3) = \psi(X_1, X_2)\psi(X_2, X_3)\psi(X_1, X_3).$$

We can convert a Bayes net to an fgraph as shown in Figure A.10. This encodes the following factorization:
$$P(C, S, R, W) = P(C)P(S|C)P(R|C)P(W|S, R)$$
This fgraph encodes the conditional independence statements corresponding to the moral graph of the Bayes net, i.e., it does not graphically represent the fact that $S \perp R | C$.

Figure A.11 shows how to convert an fgraph back to a BN; this uses the following standard trick: to represent (soft or hard) constraints or relations on a set of nodes, $\psi(X_1, \ldots, X_k)$, we simply make all of them parents of a new dummy binary node Y whose CPD is $P(Y = 1|X_{1:k}) = \psi(X_{1:k})$; by clamping $Y = 1$ (i.e., making it observed), we enforce the constraint. Note that convert a BN to an fgraph and back again does not recover the original BN, because the fgraph only encodes conditional independencies that are present in the moral (undirected) graph.[5]

[5] These transformations are discussed in [YFW01].

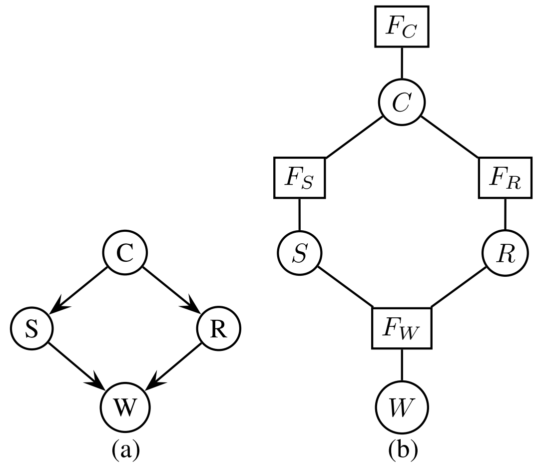


Figure A.10: Converting a Bayes net (a) to a factor graph (b). The factors are $F_W = P(W|S, R)$, etc.

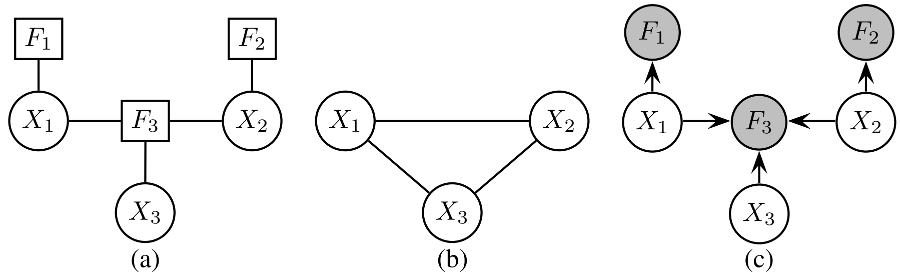


Figure A.11: Converting a factor graph (a) to an MRF (b) or a Bayes net (c).

A.5 First-order probabilistic models

Graphical models assumed a fixed-size state-space, i.e., a fixed number of random variables. Such models are called propositional, since they just assign probabilities to propositions of the form $X_{1:N} = x_{1:N}$. There has been a lot of work in the AI community on what is called first-order probabilistic logic (FOPL). Below, I will briefly review this work. (See [Pfe00, ch.9] for a more thorough review and list of references).

The three main components of a first-order language are objects, relations and quantifiers. Objects are basically groups of attributes which "belong together", c.f. a structure in a programming language, or an AI "frame". To specify the behavior of an object, it is convenient to use universal quantifiers, e.g., $\forall x.\text{human}(x) \Rightarrow \text{mortal}(x)$. This is usually modelled using classes: all objects that are instances of the human class have the property that they are mortal. Hence classes are just a way of specifying parameter tying, c.f., the way DBNs allow us to define probability distributions over semi-infinite sequences by parameter tying (quantifying over time: $\forall t.P(X_t = i|X_{t-1} = j) = A(i, j)$). Existential quantifiers (e.g., $\exists x.\text{enemy}(x) \wedge \text{nearby}(x)$) in FOPL are handled by checking whether the predicate is true for any object in the current world state.

Finally, an n-ary relation can be thought of as a binary random variable R which has n objects as parents, $X_{1:n}$; R is true iff $X_{1:n}$ satisfies the relation, e.g., sibling-of(X_1, X_2) is true iff X_1 is a sibling of X_2. Often it is very useful to find all X' s.t. $R(X, X')$ is true; this is a multi-valued function, e.g., siblings-of$(X_1) = \{X_2 : \text{sibling-of}(X_1, X_2) = \text{true}\}$. If this is guaranteed to be a single-valued function, we can write e.g., X_1.mother to mean mother-of(X_1), etc. This is sometimes called a reference slot, since its value is a pointer to another object. Structural/ relational uncertainty means that we are uncertain of the value of a reference slot. A simple example is data association ambiguity: we do not know which object caused the observation. So we can write $P(Y|Y.\text{cause} = X_i) = f(X_i)$ to denote that what we expect to see depends on the properties of the object we are actually looking at (whose identity will generally be uncertain).

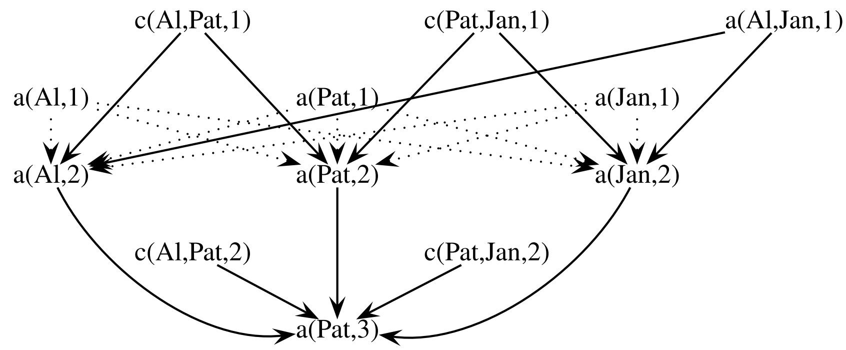


Figure A.12: A Bayes net created from a temporal knowledge base. Some lines are shown dotted merely to reduce clutter. a(X,T) is true if object X has aids at time T; c(X,Y,T) is true if X and Y had sexual contact at time T. Based on Figure 4 of [GK95].

A.5.1 Knowledge-based model construction (KBMC)

KBMC uses a knowledge base to specify set of rules which can be used to create BN structure on a case-by-case basis (see [WBG92] for an early review). Probabilistic logic programming (PLP) [NH97] is an example of KBMC.

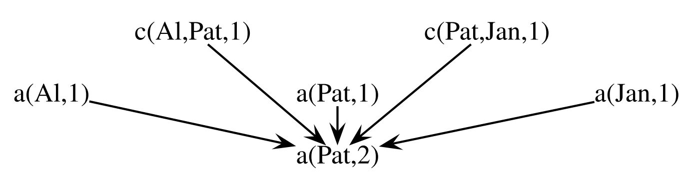
Situation calculus (see e.g., [RBKK95]) extends first order logic to temporal reasoning by adding a time argument to all predicates. A similar approach was used in [GK95] to extend PLP to the temporal case. For example, consider the rule

$$\forall X, T.\text{aids}(X, T) \Leftarrow \text{aids}(X, T-1) \vee (\exists Y.\text{aids}(Y, T-1) \wedge \text{contact}(X, Y, T-1))$$

In Prolog, this is usually written as

```
aids(X,T) :- aids(X,T-1)
aids(X,T) :- exists(Y), aids(Y,T-1), contact(X,Y,T-1)
```

Given a ground query (i.e., one which does not contain any free variables), such as `aids(Pat,2)`, the system performs backwards chaining, matching rules whose heads (left hand side) match the query. This generates two new "queries" or "proof obligations", `aids(Pat,1)` and `aids(Y,1)` $\wedge$ `contact(X,Y,1)`; the latter is then instantianted for each possible value of Y, and the corresponding nodes are added to the network. If the input facts are `aids(Al,1)=1`, `contact(Al,Pat,1)=1`, `aids(Jan,1)=0`, `aids(Pat,1)=0`, `contact(Jan,Pat,1)=0`, then we create the Bayes net shown below.



The rules which have the same head are combined in using a mechanism like noisy-OR, which can handle a variable number of parents. We can then apply any standard inference algorithm to the resulting BN. If the query becomes `aids(Pat,3)`, we create the more complicated network shown in Figure A.12.

It is clear that the models created in this way have an irregular structure, which is hard to exploit for efficient inference, especially online inference. Also, KBMC does not allow one to express structural or relational uncertainty, since the structure of the graph is automatically generated given the rules and the background facts. We will address both of these problems in the following sections.

A.5.2 Object-oriented Bayes nets

Object oriented Bayes nets (OOBNs) were introduced in [KP97]. (Similar ideas have been proposed in [LM97, BW00].) The basic idea is that each object has a set of input, output and internal (value) attributes;

given the inputs and outputs (the object's interface), the internal attributes are d-separated from the rest of the graph. This, plus the hierarchical structure of the model, allows for more efficient inference than is possible with KBMC-generated flat models c.f., [XPB93]. Furthermore, all instances of a class share the same parameters, making learning easier [LB01, BLN01].

[FKP98] introduce the concept of a dynamic object oriented Bayes net (DOOBN). Each object can either be transient or persistent; if it is persistent, it is allowed to refer to the "old" values of its internal attributes, i.e., the ones in the previous time slice. The objects can evolve at different time scales, simply by copying slowly evolving ones less often. Objects can also interact intermittently; this can be modelled by adding switching (guard condition) nodes, which effectively disable links until certain conditions are met c.f., [MP95].

The DOOBN can be flattened to a regular DBN in a straightforward way for inference purposes. Unfortunately, unlike the static case, the object oriented structure does not help speedup exact inference, since, as we shall see in Section 3.5, essentially all objects become correlated. However, the structure may be exploitable by certain approximation algorithms such as BK (see Section 4.2.1). Note that, if objects evolve at different time scales, the resulting structure will be irregular, as with KBMCs. For online inference, it will often be necessary to copy all objects at every step.

A.5.3 Probabilistic relational models

Probabilistic relational models [FGKP99, Pfe00] extend object oriented Bayes nets by allowing general relations between objects, not just "part of" relations. (In an OOBN, the inputs to an object are part of the enclosing object; the model must be strictly hierarchical.) In a PRM, each object can have attributes and reference slots, which are pointers to other objects. (A reference slot is just like a foreign key in a relational database.) An example might be course.instructor, where course is an instance of the Course class, and instructor is a pointer to an instance of the Instructor class.

Reference uncertainty means the value of the pointer (reference slot) is unknown. This can easily be modelled by add all possible objects as parents, and using a multiplexer, as in Section 2.4.6. Alternatively, we can reify a relation into an object itself, e.g., a Registration object might have two pointers, one to a Course and one to a Student. We can then define a joint probability distribution over the values of the Course and Student fields, which might depend on attributes of the Course and Student (e.g., undergrads might be less likely to take grad classes).

PRMs have not yet been applied to temporal reasoning, but it should be straightforward to do so, at least if there is no structural uncertainty. Unfortunately, the flattened graphs may be less structured than in the case of DOOBNs, potentially making inference harder. An MCMC-based approach to inference in PRMs is described in [PR01].

Appendix B

Graphical models: inference

B.1 Introduction

In this appendix, I give a tutorial on how to do exact and approximate inference in Bayesian networks. (Nearly all of the techniques are also applicable to undirected graphical models (MRFs) as well.) This tutorial brings together a lot of material which cannot be found in any one place. I start with the variable elimination algorithm, and then show how this implicitly creates a junction tree (jtree). The jtree can then be used as the basis of a message passing procedure that computes the marginals on all the nodes in a single forwards-backwards pass. I go on to discuss how to handle continuous variables and how to do approximate inference using algorithms based on message passing, including belief propagation and expectation propagation.

B.2 Variable elimination

Probabilistic inference means computing $P(X_Q|X_E = x_E)$, where X_Q is a set of query variables, and X_E is a set of evidence variables. (For instance, in medical diagnosis, X_E might be the variables representing the observed symptoms and X_Q might be the variables representing causes of these symptoms.) This can be computed from the joint distribution $P(X_1, \ldots, X_N)$ using Bayes rule:

$$P(X_Q|X_E) = \frac{P(X_Q, X_E)}{P(X_E)} = \frac{\sum_{h \not\subset Q \cup E} P(X_H = h, X_Q, X_E)}{\sum_{h \not\subset E} P(X_H = h, X_E)}$$

Hence inference boils down to marginalizing joint distributions.

If all variables are binary, then computing $\sum_h P(X_1, \ldots, X_N)$ takes $O(2^N)$ time. We would like to do this more efficiently. If the joint is represented by a Bayes net, the joint can be written in factored form:

$$P(X_1, \ldots, X_N) = \prod_{i=1}^{N} P(X_i|\mathrm{Pa}(X_i))$$

Marginalization can then be done efficiently by "pushing sums inside of products", as we show in the example below.

Consider the Bayes net in Figure B.1. The joint probability distribution can be written as

$$P(A, B, C, D, F, G) = P(A)P(B|A)P(C|A)P(D|B, A)P(F|B, C)P(G|F)$$

Suppose we want to marginalize out all the variables from this distribution. Obviously this will give the answer 1.0; however, the same idea holds for cases where we only marginalize out a subset of the variables. We can write this as follows:

$$\sum_{A,B,C,D,F,G} P(A, B, C, D, F, G) =$$
$$\sum_A P(A) \sum_B P(B|A) \sum_C P(C|A) \sum_D P(D|B, A) \sum_F P(F|B, C) \sum_G P(G|F)$$

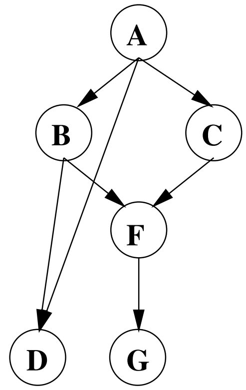


Figure B.1: A Bayesian network from [KDLC01]. The node E is omitted to avoid confusion with the evidence.

Working from right to left, we get

$$\sum_A P(A) \sum_B P(B|A) \sum_C P(C|A) \sum_D P(D|B,A) \sum_F P(F|B,C)\lambda_{G\to F}(F)$$

where $\lambda_{G\to F}(F) = \sum_G P(G|F)$. Obviously in this case $\lambda_{G\to F}(F) = 1$ for all F, but this may not be true in general. At the next step we get

$$\sum_A P(A) \sum_B P(B|A) \sum_C P(C|A)\lambda_{F\to C}(B,C) \sum_D P(D|B,A)$$

where $\lambda_{F\to C}(B,C) = \sum_F P(F|C,B)\lambda_{G\to F}(F)$.[1] Notice how we put this term next to the summation over C, "skipping" the summation over D.

In general, we try to move terms as far left as possible, so as to minimize the computational work; the restriction is that all of the variables appearing in a term must be in the scope of the appropriate sum operator. The notation $\lambda_{F\to C}(B,C)$ means this is a term arising from the summation over F and going to the summation over C; it goes to C rather than B because C is higher in the elimination (summation) ordering. It should be clear that the order in which we perform the summations determines the size of the intermediate λ terms, which can affect the computational complexity substantially; we discuss this issue in Section B.3.5. For future reference, we will call this elimination ordering π.

We can continue in this way until we have computed the desired marginal. The idea of distributing sums over products has been independently invented several times, and has various names: peeling [CTS78], symbolic probabilistic inference (SPI) [LD94], variable elimination [ZP96], bucket elimination [Dec98], etc.

The idea behind variable elimination can be generalized greatly to apply to any commutative semi-ring [AM00]. For example, we can replace sum-product with max-product to get Viterbi's algorithm for finding the most probable assignment to the variables: $x_Q^* = \arg\max_{x_Q} P(x_Q|x_E)$. Or we can replace sum-product with min-sum, and replace the local terms $F_i = P(X_i|\text{Pa}(X_i))$ with real-valued cost functions $f(X_i, \text{Pa}(X_i))$ to get a non-serial dynamic programming solution to combinatorial optimization [BB72].

We can also cast all of the following algorithms as instances of variable elimination, by using the appropriate semi-ring: the Hadamard and fast Fourier transforms [KFL01], adaptive consistency for constraint satisfaction problems (CSPs) [BMR97b], directional resolution (Davis-Putnam) for propositional satisfiability [Dec98], various grammar parsing algorithms [Goo99], etc.

[1] For discrete variables, multiplication should be interpreted elementwise, with the domains of the functions being replicated where necessary, e.g., $g_1(B,C,F) * g_2(F)$ means compute $g_1(b,c,f) * g_2(f)$ for each joint assignment to b, c, f. See [HD96] for some of the implementation details on how to perform this efficiently. (Multiplying and marginalizing multi-dimensional arrays is the main computational bottleneck in exact inference with discrete variables.) For continuous variables, see Section B.5.1.

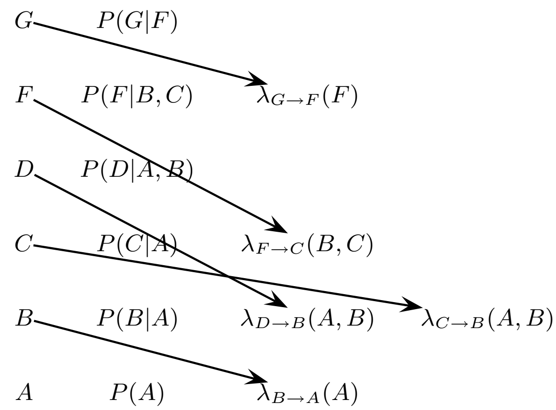


Figure B.2: Variable elimination applied to Figure B.1 using the ordering A, B, C, D, F, G. The arrow from F to C means we compute the "message" $\lambda_{F \to C}(B, C) = \sum_F P(F|C, B)\lambda_{G \to F}(F)$ and send it to C's "bucket", etc.

B.3 From graph to junction tree

Suppose we want to compute $P(X_i|X_E)$ for all $i \notin E$.[2] We could call variable elimination $O(N)$ times, once for each i, but that would be unnecessarily inefficient, since we would repeat many of the same computations. In particular, since variable elimination takes $O(N)$ time, N calls to it would take $O(N^2)$ time. We now discuss a way to compute all N marginals in $O(N)$ time.

The basic idea is to store the intermediate λ terms that we created while working right-to-left through the summations, and then to reuse them as we work left-to-right. The λ terms will be stored in a tree structured graph called a junction tree. In Section B.4, we discuss how to use this tree to compute all the marginals in $O(N)$ time by "message passing". But first we must discuss how to create the junction tree. (The presentation in this section is based in part on [KDLC01, Coz00].)

B.3.1 Elimination

The computations performed by variable elimination are depicted graphically in Figure B.2 and Figure B.3. Let us call each intermediate λ term we create a "message", and say that all the terms in the scope of the summation operator $\sum_{X_i}$ belong to "bucket" B_i. Note that the contents of the buckets change over time. However, when we eliminate bucket i (i.e., perform $\sum_{X_i}$), B_i will be "full"; let us call all the variables in B_i at this time B_i's "domain". For example the domain of bucket C is $\{A, B, C\}$, since it contains $P(C|A)$ and $\lambda_{F \to C}(B, C)$ when we sum out C.

In general, we can compute the domain of each bucket as follows.[3] First we "moralize" the DAG by connecting together unmarried parents (nodes which share a child), and then dropping the directionality on all arcs. The result is an undirected graph with an edge between i and j if there is some term (e.g., CPD) which involves both X_i and X_j; see Figure B.4 for an example. We will call this graph G. (Moralization is unnecessary if we start with an undirected graphical model.) Then we apply the algorithm in Figure B.5, which takes as input G and an elimination ordering π, and returns a set of elimination sets, $C_1, \ldots, C_N$. The domain of the bucket for X_i is $C_{\pi^{-1}(i)}$, where $\pi^{-1}(i)$ is the position of X_i in the ordering.

In Figure B.6, we show an example of eliminating with the ordering $\pi = (A, B, C, D, F, G)$. In this case, we do not need to add any fill-in arcs. However, in Figure B.7, we show an example of eliminating with the ordering $\pi = (A, F, D, C, B, G)$; when we eliminate B, we connect all its uneliminated (lower-

[2] This is useful for speech recognition, image reconstruction, decoding messages sent over noisy channels, etc., where we care about all the hidden variables. For parameter learning, we need to compute the family marginals $P(X_i, \text{Pa}(X_i)|E)$, which is straightforward given $P(X_i|E)$. For structure learning, and a few other tasks, we need to compute $P(X_S|E)$ for a potentially arbitrary subset of nodes S. The optimal way to do this, using junction trees, is described in [EH01].

[3] Pre-computing the domains is useful for two reasons. Firstly, the computational complexity depends exponentially on the size of the largest domain, so we will want to search over summation orderings to try to minimize this: see Section B.3.5. Secondly, but more prosaically, it is very useful to pre-allocate space to the buckets.

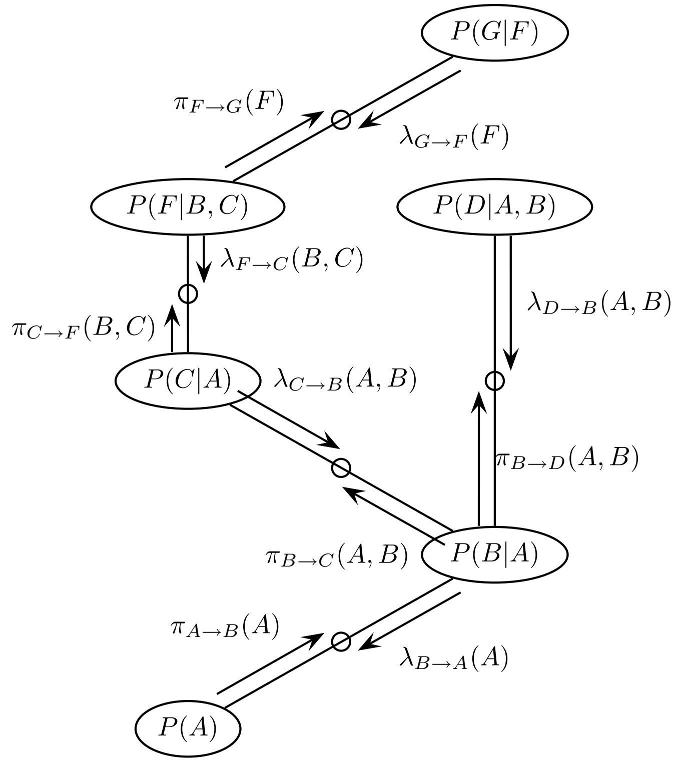


Figure B.3: The elimination tree derived from Figure B.2. Ovals represent "buckets"; the CPDs inside them are the terms they contain after initialisation. Note that the "domain" of the C bucket is $\{A, B, C\}$, even though it is initialized with $P(C|A)$. The empty circles represent separators. λ messages flow from the leaves to the root (the A node); then π messages flow from the root to the leaves.

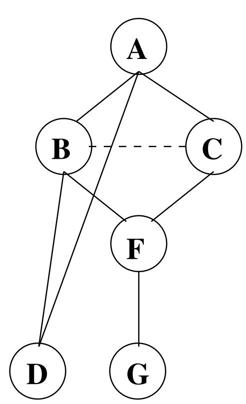


Figure B.4: The moral graph of Figure B.1. Dotted edges are moralization arcs.

- Start with all vertices unnumbered, set counter $i := N$.
- While there are still some unnumbered vertices:
  - Let $v_i = \pi(i)$.
  - Form the set C_i consisting of v_i and its (unnumbered/ uneliminated) neighbors.
  - Fill in edges between all pairs of vertices in C_i.
  - Eliminate v_i and decrement i by 1.

Figure B.5: Pseudo-code for elimination c.f., algorithm 4.13 of [CDLS99, p58]. π is the elimination ordering, which we use in reverse order.

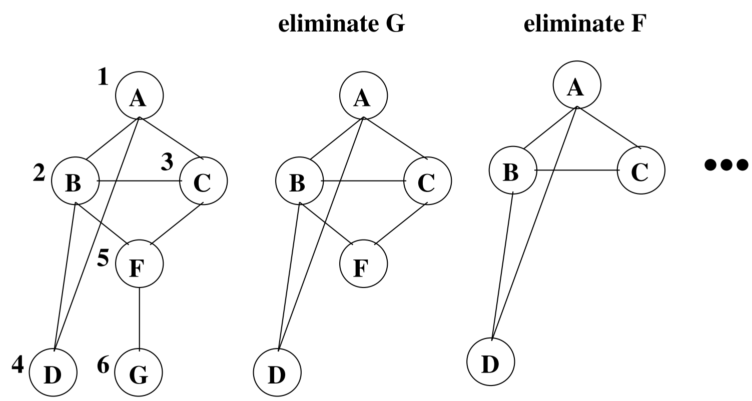


Figure B.6: Eliminating the nodes in Figure B.4, with $\pi = (A, B, C, D, F, G)$.

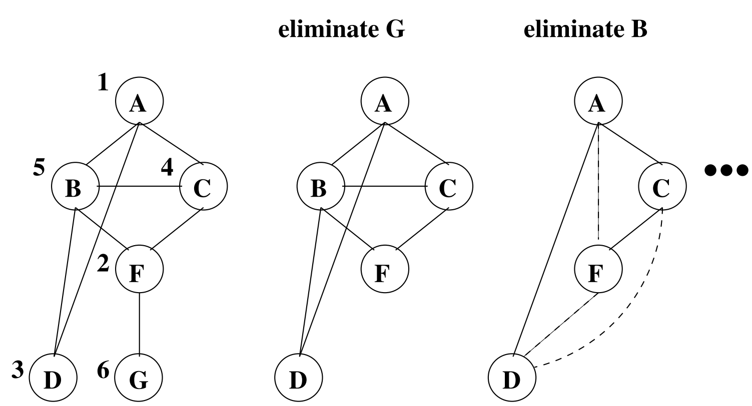


Figure B.7: Eliminating the nodes in Figure B.4, with $\pi = (A, F, D, C, B, G)$. Dotted edges are fill-in arcs. When we eliminate B, all the remaining nodes are lower in the ordering, and hence all neighbors of B get connected.

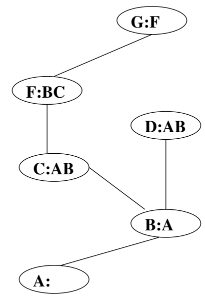


Figure B.8: An elimination tree derived from Figure B.6. The notation $C : A, B$ means, when C was eliminated, $\{A, B\}$ were the uneliminated neighbors.

numbered) neighbors, which results in the elimination set for B being $\{A, C, D, F\}$; this is easily seen from the following decomposition:

$$\sum_{A,F,D,C,B,G} P(A, B, C, D, F, G) =$$

$$\sum_A P(A) \sum_F \sum_D \sum_C P(C|A) \sum_B P(D|B, A)P(F|B, C)P(B|A) \sum_G P(G|F)$$

The size of the largest elimination set is called the induced width of the graph. In this case, the width induced by $\pi = (A, B, C, D, F, G)$ is 3, which is better than $\pi = (A, F, D, C, B, G)$, which induces a width of 4. In Section B.3.5, we discuss ways to search for a good elimination ordering (in the sense of minimizing the induced width).

B.3.2 Triangulation

The extra edges we introduce in the elimination process (if any) are called "fill-in edges". With the ordering $\pi = (A, B, C, D, F, G)$, there are no fill-in edges, but with the ordering $\pi = (A, F, D, C, B, G)$, we added fill-in edges $A - F$, $D - F$ and $C - D$ when we eliminated B. If we add these fill-in edges to the original graph G, the result is a triangulated or chordal graph, G^T. (A triangulated graph is one which does not possess any cycles of length ≥ 4 without a chord which breaks the cycle.)

A graph is triangulated iff it possess a perfect elimination ordering. This is an ordering such that if we eliminate in that ordering, we will not introduce any fill-in edges. An algorithm called maximum cardinality search (MCS) [TY84] can be used to find a perfect elimination ordering given a triangulated graph. (Although [Pea88, p112] suggests using MCS to triangulate a graph, [CDLS99, p58] claim that this often results in more fill-in edges than necessary. We discuss better ways to find elimination orderings for non-chordal graphs in Section B.3.5. Hence MCS is more suited to testing chordality than to ensuring it.)

In the current example, $\pi = (A, B, C, D, F, G)$ is a perfect elimination ordering, which tells us that the graph in Figure B.4 is already triangulated. If a graph is not triangulated, it can be made so by using the procedure in Figure B.5.

B.3.3 Elimination trees

The elimination sets created during triangulation can be arranged into a tree, called a bucket tree [Dec98] or an elimination tree (etree) [CDLS99, p.59]; see Figure B.8 for an example. We simply connect C_i to C_j, where j is the largest index of a vertex in $C_i \setminus \{V_i\}$. Let us call this procedure `etree-from-esets` (algorithm 4.14 of [CDLS99, p59]).

The nodes in the elimination tree are sets of nodes in the original graph G. In fact, they are cliques of the triangulated graph, but not necessarily maximal cliques. In Figure B.8, the elimination sets are $C_1 = \{A\}$, $C_2 = \{A, B\}$, $C_3 = \{A, B, C\}$, $C_4 = \{A, B, D\}$, $C_5 = \{B, C, F\}$ and $C_6 = \{F, G\}$, which are clearly (non-maximal) cliques of the triangulated graph in Figure B.4.

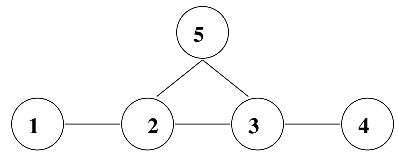


Figure B.9: The numbering of the vertices is perfect, but the cliques, numbered as $(\{1,2\},\{3,4\},\{2,3,5\})$, do not satisfy RIP; hence this numbering could not have been generated by max cardinality search. From Figure 4.9 of [CDLS99].

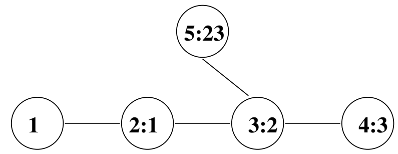


Figure B.10: Elimination tree obtained from the chordal graph of Figure B.9 using the numbering $\{1,\ldots,5\}$, i.e., we first eliminate 5, and ensure lower-numbered neighbors $\{2,3\}$ are connected, etc. From Figure 4.10 of [CDLS99].

B.3.4 Junction trees

The elimination sets satisfy the running intersection property (RIP), which is defined as follows. An ordered sequence of sets $C_1,\ldots,C_k$ is said to satisfy RIP if, for all $1 < j \leq k$, there is an $i < j$ s.t. $C_j \cap (C_1 \cup \cdots \cup C_{j-1}) \subseteq C_i$, e.g., $C_4 \cap \{C_1, C_2, C_3\} = \{A,B,D\} \cap \{A,B,C\} \subseteq C_2$. Figure B.9 shows an example of a sequence of sets that does not satisfy RIP.

Any sequence of sets $C_1,\ldots,C_k$ that satisfies RIP can be converted into a junction or join tree (jtree), whose nodes are the C_i's. A tree is said to be a junction tree, or to have the junction tree property, if $C_1 \cap C_2$ is contained in every C_i ($i \neq 1,2$) on the unique path between C_1 and C_2, for every pair C_1, C_2. In other words, all the sets containing some X_j form a connected tree: along any path, X_j cannot disappear and then reappear later.

We can construct a jtree from a sequence of sets which satisfy RIP by connecting C_j to any C_i, for $i \in \{1,\ldots,j-1\}$, s.t. $C_j \cap (C_1 \cup \cdots \cup C_{j-1}) \subseteq C_i$. Let us call this procedure `jtree-from-RIP-clqs` (algorithm 4.8 in [CDLS99, p55]; see also [Pea88, p113]). Note that this way of constructing a jtree is not valid unless the order of the sets satisfies RIP.

The elimination tree is a jtree of sets, but is not a jtree of maximal cliques. It is conventional to remove the redundant sets.[4] We can do this by a simple pruning procedure: when we create C_j, we check if there is any higher numbered (already created) clique, $C_j \subset C_i$ for $i > j$, and if so, we simply omit C_j from the output sequence. Unfortunately, the resulting sequence may no longer satisfy RIP. For example, consider the elimination tree in Figure B.10, with sequence of elimination sets $C_1 = \{1\}$, $C_2 = \{1,2\}$, $C_3 = \{2,3\}$, $C_4 = \{3,4\}$ and $C_5 = \{2,3,5\}$. If we delete C_3, we violate RIP.

There is a way to remove redundant sets and keep RIP which we shall call `RIP-prune`: see Lemma 4.16 of [CDLS99, p60] for details. Alternatively, we can use max cardinality search to find a perfect elimination ordering, and then use Algorithm 4.11 of [CDLS99, p56] (which we call `RIP-max-clqs-from-MCS`) to find the maximal cliques of the chordal graph, in an order that satisfies RIP. Either way, once we have a sequence of maximal cliques that satisfies RIP, we can use `jtree-from-RIP-clqs` to create a jtree of maximal cliques. For example, see the jtree Figure B.12; compare this with the etree in Figure B.8. In general, a jtree may be much smaller than an etree.

Fortunately, there is a much simpler way to construct a jtree from a set of maximal cliques, even if their ordering does not satisfy RIP. We first create a junction graph, in which we add an edge between i and j if

[4] It is not always a good idea to only use maximal cliques. One advantage of an etree is that $C_j \setminus C_{ij}$ is always a singleton, where C_{ij} is the intersection between two neighboring cliques. This means that we only have to marginalize one variable at a time during message passing (see Section B.4). When variables can be of mixed types, e.g., random variables and decision variables in an influence diagram, this can result in simpler algorithms: see e.g., [CDLS99, ch.8]. [GLS99] also gives cases where it is better *not* to require that the cliques be maximal, in the context of constraint satisfaction problems.

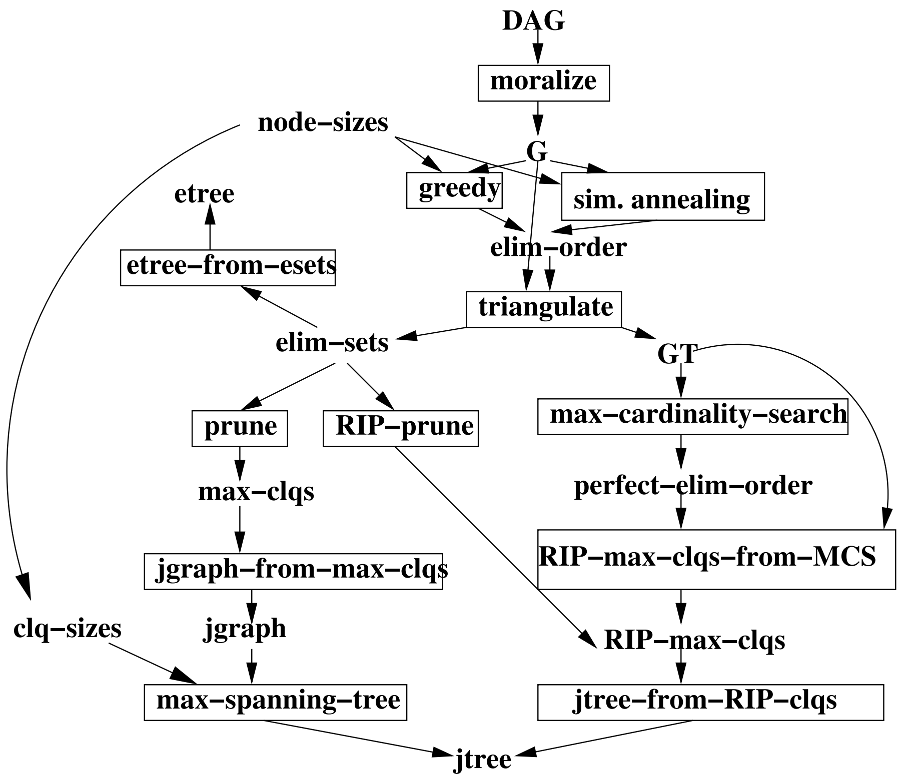


Figure B.11: Summary of the steps involved in converting a DAG to a junction tree. The cost of a node is q_i, the number of discrete values it can take on. GT is the triangulated version of G. We now specify the correspondence between some of the square boxes and algorithms in [CDLS99]. Greedy is Algorithm 4.13. Simulated annealing is another way of trying to compute an optimal elimination ordering; see Section B.3.5. `max-cardinality-search` is Algorithm 4.9. `RIP-max-clqs-from-MCS` is Algorithm 4.11. `RIP-prune` is Lemma 4.16. `jtree-from-RIP-clqs` is Algorithm 4.8. `etree-from-esets` is Algorithm 4.14. See text for details.

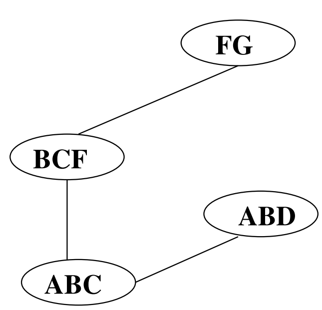


Figure B.12: A jtree for Figure B.1 constructed using the elimination ordering (A, B, C, D, F, G). Compare with the etree in Figure B.8.

$C_i \cap C_j \neq \emptyset$. We set the weight of this edge to be $|C_i \cap C_j|$, i.e., the number of variables they have in common. Any maximal weight spanning tree (MST) of the jgraph is a junction tree (see [JJ94a] or [AM00] for a proof). We can construct an MST using Kruskal's algorithm in $O(|E| \log |E|)$ time, or using Prim's algorithm in $O(N^2)$ time [CLR90]. (BNT uses Prim's algorithm, because Kruskal's requires a priority queue, which is inefficient in Matlab.) The best MST is the one which minimizes the sum of the costs on each edge, where the cost of an edge $e = (v, w)$ is defined as $\chi(e) = q_v + q_q - q_{v \cap w}$. q_v is the cost of node v in the jgraph, and is defined as $q_v = \prod_{i \in v} q_i$, where q_i is the number of discrete values X_i can take on. If we use $\chi(e)$ to break ties when constructing the MST, the result will be an optimal junction tree (for a fixed elimination ordering) [JJ94a].

The overall procedure for converting a DAG to a jtree is summarized in Figure B.11.

B.3.5 Finding a good elimination ordering

We have mentioned several times that the cost of inference is exponential in the size of the largest clique/ λ term/ bucket / elimination set. We would like to minimize the max clique size. However, this problem is NP-complete [Yan81, ACP87]. A standard greedy approximation is the following: eliminate any node which would not result in any fill-ins (i.e., all of whose uneliminated neighbors already form a clique); if there is no such node, eliminate the node which would result in the minimum number of fill-in edges. This is called the "min-fill" heuristic. An alternative that works better in practice is the "min-weight" heuristic, where we try to minimize the weight of the created cliques, where the weight of a clique C is $\prod_{i \in C} q_i$, where q_i is the cardinality of X_i. (If all nodes have the same weight, this is the same as min-fill.)

Of course, many other methods are possible. [Kja90, Kja92] compared simulated annealing with the above greedy method, and found that it sometimes works better (although it is much slower). [MJ97] approximate the discrete optimization problem by a continuous optimization problem. [BG96] present a randomized approximation algorithm. See [Ami01] for various recent constant-factor appoximation algorithms.

B.3.6 Strong junction trees

When we consider graphical models with more than one type of node we sometimes have to use a constrained elimination ordering. For example, in an influence diagram, we must max out a decision node before we sum out the random (information) variables that it depends on [JJD94]. Also, in a conditionally Gaussian (CG) Bayes net (see Section B.5), if we want to get exact answers we must integrate out a continuous node before marginalizing out the discrete parents it depends on, i.e., we write $\sum_i \int_x f(x, \mu(i), \Sigma(i))$, rather than $\int_x \sum_i f(x, \mu(i), \Sigma(i))$. In general, if we have a partial ordering, where $i \prec j$ means we must eliminate i before j, we try to find the best elimination ordering consistent with the partial ordering. A valid elimination ordering π will such that the reverse of π is a total ordering of $\prec$. (The reason why we consider the *reverse* of π is that we work backwards: recall from Section B.2 that if $\pi = (A, B, C, D, F, G)$, we first summed over G.) In the case of CG networks, we ensure that all the continuous nodes are eliminated before any of the discrete nodes. This results in a strongly triangulated graph, which can be converted to a junction tree of (not necessarily maximal) cliques in the usual way (e.g., using max spanning tree).

In addition to using a restricted elimination ordering, we must choose the root R to be a clique which is "strong". A root R is strong if, for any pair C_1, C_2 of adjacent cliques with C_1 closer to R than C_2, there exists an ordering of C_2 that respects $\prec$, and with $C_1 \cap C_2 \succ C_2 \setminus C_1$ [JJD94]. In the case of CG networks, we have $\Delta \succ \Gamma$, where Γ is the set of Gaussian variables and Δ is the set of discrete variables, so this definition can be restated as follows [Lau92]:

$$(C_2 \setminus C_1) \subseteq \Gamma \text{ or } (C_1 \cap C_2) \subseteq \Delta$$

In other words, if the residual $C_2 \setminus C_1$ (novel variables further from the root) contains a discrete variable, then the separator is purely discrete. Roughly speaking, the tree becomes more and more discrete as we move towards the root.

It can be shown that any strongly triangulated graph has a jtree with a strong root. Furthermore, the last clique (C_1) created by the standard one-step look-ahead strong triangulation routine (see Figure B.5) is a strong root, and this will obviously not get pruned out.

B.4 Message passing

We now explain how to compute all N marginals in $O(N)$ time by using the junction tree. We will present several different algorithms from the literature, all of which are essentially the same.

B.4.1 Initialization

Let clique i have domain S_i, and let $S_{ij} = S_i \cap S_j$ be the separator between cliques i and j. Let us associate a "potential" with each clique. A potential is a non-negative function of its arguments; if the arguments are discrete, we can represent potentials as multi-dimensional arrays; if the arguments are continuous, we may be able to use a parametric representation such as a Gaussian or other member of the exponentialy family; if some arguments are discrete and some are continuous, we can use mixtures of Gaussians (see Section B.5.2). We initialize potentials to be the identity element for the semi-ring we are considering, which is 1 in the case of sum-product with discrete potentials.

Each term F_i in the original problem (the CPD for X_i in the case of Bayes nets) is associated with a clique S_j that contains all of F_i's domain. In an etree, we have one clique per term (see Figure B.3), but in a jtree, we may have to associate multiple terms with the same clique, e.g., in Figure B.12, we can associate $P(A)$ and $P(B|A)$ with cliques ABC or ABD. Hence the clique ABD may contain terms $P(A)$, $P(B|A)$ and $P(D|A, B)$. All the terms associated with a clique are multiplied together elementwise to get a single initial clique potential, ψ_j.

B.4.2 Parallel protocol

Perhaps the simplest formulation of message passing, which is often called belief propagation, is as follows [AM00]. At each step, every node in the jtree in parallel collects all the messages from its neighbors and computes

$$\phi_j = \left[\prod_{i \in \text{nbr}(j)} \mu_{i \to j} \right] \psi_j$$

where $\mu_{i \to j}(S_{ij})$ is the message sent from node i to node j, and $\psi_j(S_j)$ is the term assigned to node j (the initial clique potential). Once a node has received messages from all its neighbors, it can then send a message to each of them, as follows:

$$\mu_{j \to k} = \left(\left[\prod_{i \in \text{nbr}(j), i \neq k} \mu_{i \to j} \right] \psi_j \right) \downarrow S_{jk} = \left(\frac{\phi_j}{\mu_{k \to j}} \right) \downarrow S_{jk} = \frac{(\phi_j) \downarrow S_{jk}}{\mu_{k \to j}}$$

where the notation $\phi(S) \downarrow T$ means $\sum_{S \setminus T} \phi(S)$, i.e., marginalization (projection) onto set (domain) T. (For discrete variables, division is element-wise, but we define $0/0 = 0$; for division of continuous variables, see Section B.5.1.) If we initialise all messages to the identity element (1's), each node can apply the above rules in parallel. After D steps, where $D \leq N$ is the diameter of the graph, the final ϕ_j values will be proportional to $P(S_j|e)$, where e is the evidence. (For a proof of this, see Section B.4.7.) Since the final ϕ_j values will be normalized, we are free to scale the messages arbitrarily, since the scaling factors will cancel at the end; this is very useful to prevent underflow in large networks and iterative algorithms.

B.4.3 Serial protocol

In the above parallel protocol, all nodes are performing a computation at every step, so on a serial machine, it would take $O(N^2)$ time to get the exact answers for a tree. Other message passing schedules are possible (see [AM00] for conditions). We now present a particularly simple serial protocol which can compute all marginals in two passes over the tree, i.e., in $O(N)$ time (see Figure B.13). The algorithm has two phases: collect to root, and distribute from root. (The root node can be chosen arbitrarily.) Performing collect and then distribute is sometimes called "calibrating" the jtree. In the collect phase, a node collects messages from all its children, and then passes a message to its unique parent, i.e., nodes are processed in post-order. In the

```
function [φ, μ] = collect(F)
for each j in post-order
    φ_j = ∏_{i∈ass(j)} F_i × ∏_{i∈pred(j)} μ_{i→j}
    k = parent(j)
    μ_{j→k} = φ_j ↓ S_{jk}

function [φ, μ] = distribute(φ, μ)
for each j in pre-order
    k = parent(j)
    φ_j = normalize(φ_j × μ_{k→j})
    for each child i
        μ_{j→i} = (φ_j ↓ S_{ij}) / μ_{i→j}
```

Figure B.13: Pseudo-code for the JLO/GDL algorithm using a serial updating protocol. (JLO = Jensen, Lauritzen and Olesen [JLO90]; GDL = generalized distributive law [AM00].) Calibrating the jtree means calling collect followed by distribute. We assume a product over an empty set returns 1. $\text{ass}(j)$ are the terms (CPDs) assigned to clique j. Post-order means children before parents, pre-order means parents before children; pred(j) means predecessors of j in the post-order. ϕ_j has domain S_j, and $\mu_{j\to k}$ has domain $S_{jk} = S_j \cap S_k$. This code is designed to be as similar as possible to belief propagation on a factor graph (see Figure B.20). See Figure B.14 for a slightly different implementation. Normalization is one way to prevent numerical underflow. It also ensures that, upon termination, the clique potentials can be interpreted as conditional probabilities: $\phi_j = P(S_j|e)$, where e is all the evidence.

- For each j in post-order
  - $\phi_j = \prod_{i\in \text{ass}(j)} F_i$
  - Let k be the parent of j in the jtree.
  - $\phi_{jk} = \phi_j \downarrow S_{jk}$
  - $\phi_k = \phi_k * \phi_{jk}$
- For each j in pre-order
  - For each child i of j
    - $\phi_i = \frac{\phi_i}{\phi_{ij}}$
    - $\phi_{ij} = \phi_i \downarrow S_{ij}$
    - $\phi_i = \phi_i * \phi_{ij}$
- Normalize each ϕ_j.

Figure B.14: An alternative implementation of the JLO algorithm. (This is how it is implemented in BNT.) The disadvantage of this method is that node j does not update its local potential ϕ_j, but instead updates the potentials of its parent ϕ_k or children ϕ_i. Also, the similarity to BP on a factor graph (see Figure B.20) is obscured in this version.

distribute phase, a node collects a message from its unique parent, and passes it to all its children, i.e., nodes are processed in pre-order.

Consider a generic node j with children i and i' and unique parent k. In the collect phase, $\mu_{k\to j} = 1$, since all messages are initialized to 1's. Hence we can skip the division by $\mu_{k\to j}$, so the update rules simplifies as follows:

$$\phi_j = \left[\prod_{i\in \text{ch}(j)} \mu_{i\to j}\right] \psi_j$$

$$\mu_{j\to k} = \phi_j \downarrow S_{jk}$$

This corresponds exactly to the variable elimination algorithm: computing ϕ_j is what happens when we combine all the terms in bucket j, and computing $\mu_{j\to k}$ corresponds to marginalizing out $S_j \setminus S_{jk}$, which, for an etree, is always a single variable, namely X_j.

In the distribute phase, node j computes its new potential

$$\phi_j^* = \phi_j \times \mu_{k\to j}$$

where k is the parent of j. Then it sends the following message to its child i:

$$\mu_{j\to i} = \left(\left[\prod_{i'\in \text{ch}(j), i'\neq i} \mu_{i'\to j}\right] \mu_{k\to j}\psi_j\right) \downarrow S_{ij} = \left(\frac{\phi_j^*}{\mu_{i\to j}}\right) \downarrow S_{ij} = \frac{\left(\phi_j^*\right) \downarrow S_{ij}}{\mu_{i\to j}}$$

The overall algorithm is summarized in Figure B.13.

B.4.4 Absorption via separators

The most common presentation of message passing in junction trees (see Figure B.14) looks different from what we have just seen, but we now show that it is equivalent. The basic operation is absorption: if node (clique) i is connected to j via a separator s, then we say j absorbs from i if

$$\phi_j^* = \phi_j \frac{\phi_s^*}{\phi_s}$$

where $\phi_s^* = \phi_i \downarrow s$. (The asterisk denotes updated potential.)

The potential on a separator is equivalent to what we have been calling a message. The potential on S_{ij} stores $\mu_{i\to j}$ if it was just updated by i, or $\mu_{j\to i}$ if it was just updated by j.

Suppose we want to update ϕ_j to reflect the new message arriving from i, $\mu^*_{i\to j}$. The new potential can be computed as

$$\phi_j^* = \left[\prod_{i'\in \text{nbr}(j), i'\neq i} \mu_{i\to j}\right] \mu^*_{i\to j}\psi_j = \frac{\phi_j}{\mu_{i\to j}}\mu^*_{i\to j} = \phi_j \frac{\mu^*_{i\to j}}{\mu_{i\to j}}$$

This is what is usually meant by the "junction tree algorithm", also known as clustering, the Hugin algorithm (the name of a famous commercial program that implements it), the JLO algorithm [JLO90], "clique tree propagation", etc. However, it is useful to distinguish the junction tree as a data structure from the kind of message passing that is performed on it. Also, it is useful to distinguish between a jtree of maximal cliques (as used by Hugin), a jtree of elimination sets (an etree), binary jtrees [She97], jtrees of maximal prime subgraphs [OM99], etc.

B.4.5 Hugin vs Shafer-Shenoy

The version of message passing discussed in Section B.4.4 is often known as the "Hugin architecture". It stores the product of messages at each clique and uses division to implement the product of all-but-one of the messages. An alternative is known as the "Shafer Shenoy architecture" (SS) [SS88, SS90a, SS90b, She92], which does not does not store the product of messages at each clique and does not involve a division operator.

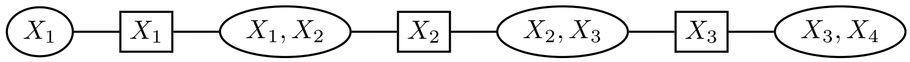


Figure B.15: A junction tree for a Markov chain $X_1 \to X_2 \to X_3 \to X_4$; square nodes represent separators, ovals represent cliques. The X_1 clique and separator is not strictly necesssary, but is included to ensure a 1:1 correspondence between separators and nodes in the original graph. The clique potentials are $P(X_1)$, $P(X_2|X_1)$, $P(X_3|X_2)$, $P(X_4|X_3)$, etc.

(It might seem that division for Gaussian distributions is not well-defined, but in fact it is: see Section B.5.1. However, mixtures of Gaussians cause problems: see Section B.5.2.)

Hugin uses more space (because it must store clique and separator potentials), but less time (because it does not need to repeatedly remultiply messages). The need to repeatedly multiply messages in SS can be ameliorated using binary join trees [She97, SS98]. This supposedly makes SS about as fast as Hugin. See [LS98] for a detailed comparison of the SS and Hugin architectures.

"Lazy" junction trees [MJ99] only multiply together those terms which are assigned to a clique on a demand-basis, i.e., if the terms contain a variable which is about to be marginalized out. In other words, lazy Hugin maintains clique potentials as a product of factors. Division is implemented by multiplying all-but-one of the relevant factors, as in SS.

B.4.6 Message passing on a directed polytree

Belief propagation was first presented in [Pea88] for directed polytrees. (A polytree is a directed tree with more than one root; hence a node can have several parents as well as several children, but the overall graph has no cycles, directed or undirected.) We now show that this is equivalent to the message passing scheme we have just presented, in the case that the original graph is a (directed) chain; the general polytree case follows analogously.

If the DAG is a chain, the cliques of the corresponding jtree will be families (nodes plus parents), and the separators will be nodes. See Figure B.15 for an example. In the JLO/GDL algorithm, cliques sent messages to cliques via separators. Now we will let separators send messages to separators via cliques. Messages from children to parents are called λ messages, and the messages from parents to children are called π messages (see Figure B.3).

Consider a node j with unique child i and unique parent k. Since $\phi_j = P(j|k)$ and $S_{jk} = \{k\}$, the λ message is

$$\lambda_{j\to k}(k) = \sum_k \lambda_{i\to j}(j) P(j|k)$$

and the π message is

$$\pi_{j\to i}(j) = \sum_k \pi_{k\to j}(k) P(j|k)$$

Note that this is equivalent to the forwards-backwards algorithm for HMMs, where $\alpha \equiv \pi$ and $\beta \equiv \lambda$ [SHJ97]. Also, note that the λ and π messages can be computed independently of each other. However, this is only true for a chain: for a tree, the π's depend on the λ's (computed in a bottom up pass), and for a polytree, the λ's and π's both depend on each other in the following way:

$$\begin{aligned}
\lambda_{j\to k}(k) &= \prod_i \lambda_{i\to j}(j) \prod_{k'\neq k} \pi_{k'\to j}(k') P(j|k,k') \downarrow S_{jk} \\
&= \sum_{j,k'} \lambda_j(j) P(j|k,k') \prod_{k'\neq k} \pi_{k'\to j}(k')
\end{aligned}$$

where $\lambda_j(j) = \prod_{i\in \text{ch}(j)} \lambda_{i\to j}(j)$, and

$$\begin{aligned}
\pi_{j\to i}(j) &= \prod_{i'\neq i} \lambda_{i'\to j}(j) \prod_k \pi_{k\to j}(k)\; P(j|k,k') \downarrow S_{i,j} \\
&= \pi_j(j) \prod_{i'\neq i} \lambda_{i'\to j}(j)
\end{aligned}$$

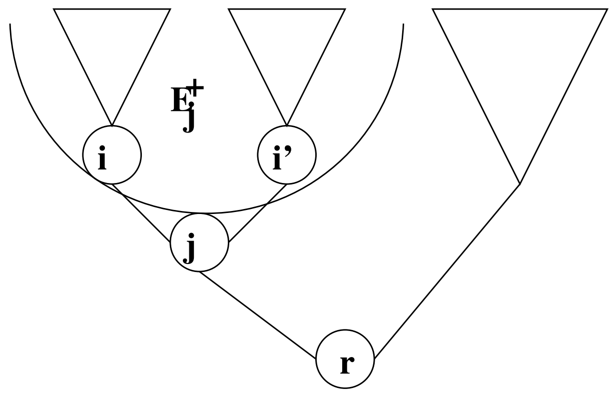


Figure B.16: Conditioning on X_j divides the tree into the two sets, one which contains evidence in the subtree rooted at X_j, which we call E_j^+, and the other which contains all the other evidence, which we call E_j^-.

where $\pi_j(j) = \sum_{k,k'} P(j|k,k') \prod_k \pi_{k \to j}(k)$.

The final beliefs are given by $P(X_j|e) \propto \lambda_j \pi_j$. See [Pea88] for a proof.

Cutset conditioning

One of the earliest approaches to exact inference in DAGs with undirected cycles was cutset conditioning [Pea88]. This works by finding a set of nodes, called a cutset, which, when removed, turns the remaining graph into one or more polytrees. Exact inference can be performed by instantiating the cutset to each of its possible values, and running message passing in each polytree, and combining the results [PS91]. Although intuitively appealing (this algorithm is similar to reasoning by cases), it takes time exponential in the size of the cutset, which is always at least as large as the tree width (max clique size). Hence cutset conditioning is never any faster than message passing in a junction tree [SAS94], and is not widely used.

Applying Pearl's algorithm to jtrees

To apply Pearl's algorithm to a jtree, we root the tree, and treat the cliques as (mega-)nodes. To do this, we must compute $P(j|k)$ for neighboring cliques j and k. For discrete nodes, this can be done by ensuring $P(j|k) = 0$ if j and k are inconsistent on shared values, and otherwise multiplying together the CPDs assigned to clique j (see [Zwe98] for details). Note that this transformation may destroy any structure (apart from 0s) in the local CPDs. Also, it is not clear how to do this for linear-Gaussians CPDs.

B.4.7 Correctness of message passing

We have already remarked that the collect-evidence procedure is identical to variable elimination, which is "obviously" correct. The question remains of why message passing in general is correct.

If the original Bayes net is a polytree (i.e., has no undirected cycles), Pearl [Pea88] proved that message passing is correct by using d-separation properties of the DAG. The basic idea is that by conditioning on a node X_j, we partition the tree into two sets, one of which contains the evidence in the subtree rooted at X_j, call it E_j^+, and the other which contains the remaining evidence, E_j^-: see Figure B.16. Now $P(X_j|E_j^+, E_j^-) \propto P(X_j|E_j^+)P(E_j^-|X_i)$, so we can compute $\pi_j = P(X_j|E_j^+)$ and $\lambda_j = P(E_j^-|X_j)$ separately and then combine them.

For junction trees, things are trickier, because the nodes represent *sets* of random variables. However, the fact that we only assign each term F_i to a unique clique means that we do not overcount information. Also, the junction tree property ensures that when two or more subtrees send a message to a node, their information can be fused consistently. See [AM00, app.A] for a simple inductive proof of correctness. Historically, the first proof of correctness for general semi-rings was [SS88] for the SS architecure and [LJ97] for the Hugin architecture.

B.4.8 Handling evidence

The standard implementation of the junction tree algorithm (see e.g., [CDLS99]) is to compute the initial clique potentials ψ_j by multiplying all the CPDs assigned to clique j, and then to calibrate the tree. The clique potentials now represent unconditional joint distributions: $\phi_j = P(S_j)$ (marginals over the variables in the clique).

Now suppose (hard) evidence arrives on node X_i, i.e., we observe $X_i = x_i$. We find any clique that contains X_i, and set the elements of the clique potential that are inconsistent with the evidence to 0, e.g., if we have a clique containing two binary nodes, X_1 and X_2, and we observe $X_2 = 1$, then we set $\phi(x_1, x_2) = 0$ if $x_2 \neq 1$. Note that many of the potentials may end up being sparse. (If we have soft (virtual) evidence, we simply reweight the terms.) We then perform another calibration to propagate the evidence.

When dealing with Gaussian distributions (see Section B.5.1), adding evidence to a node reduces the size of the Gaussian potential, so we must add the evidence to every potential that contains an observed node (since we can't have 0s in the covariance matrix). (This can be done in the discrete case too, but is unnecessary, since 0s will be propagated.)

The above scheme will fail if we have a model with conditional-only CPDs, e.g., a regression model with two nodes, X and Y, where we specify $P(Y|X)$ but not the prior $P(X)$, since we cannot evaluate $P(Y|X)$ until we know X. However, there is a simple solution (first proposed in [Mur99]): delay converting the CPDs to potential form until after the evidence arrives. Hence we eliminate the initial calibration phase completely. This has the additional advantage that we know that discrete observed variables effectively only have one possible value, and continuous observed values are "zero-dimensional points" (i.e., just scale factors); this means we can allocate potentials of just the right size: no need for zero compression or shrinking of Gaussian potentials.

B.5 Message passing with continuous random variables

So far, we have assumed that potentials are represented as tables (multi-dimensional arrays); multiplication and division are defined pointwise, and marginalization is just summation over the appropriate dimension(s). We now discuss what to do if some of the variables are continuous. We start by assuming all CPDs are linear-Gaussian, in which case the joint distribution is just a large, sparse multivariate Gaussian. In this case, exact inference always takes at most $O(N^3)$ time, since we can just create the corresponding Gaussian. However, N might be very large, rendering this approach impractical.

Then we consider the case where some CPDs are conditional linear Gaussian (CLG), i.e., of the form $P(Y = y|U = u, X = i) = \mathcal{N}(y; W_i u + \mu_i, \Sigma_i)$. We also allow discrete CPDs (of any kind), but only if they have no continuous parents. The result is just a a mixture of sparse Gaussians, where the mixture distribution is factorized as in a discrete Bayes net, and the sparsity pattern can vary between mixture components. This is called a conditional Gaussian (CG) distribution [LW89, Lau92, Ole93, CDLS99].

Finally we consider arbitrary combinations of CPDs, where exact inference is usually not possible.

B.5.1 Pure Gaussian case

Moment and canonical characteristics

We can represent a Gaussian distribution in moment form or in canonical (information) form. In moment form we have

$$\phi(x; p, \mu, \Sigma) = p \times \exp\left(-\tfrac{1}{2}(x-\mu)'\Sigma^{-1}(x-\mu)\right)$$

where $p = (2\pi)^{-n/2}|\Sigma|^{-\frac{1}{2}}$ is the normalizing constant that ensures $\int_x \phi(x; p, \mu, \Sigma) = 1$. ($n$ is the dimensionality of X.) Expanding out the quadratic form and collecting terms we get the canonical form:

$$\phi(x; g, h, K) = \exp\left(g + x'h - \tfrac{1}{2}x'Kx\right)$$

where

$$\begin{aligned} K &= \Sigma^{-1} \\ h &= \Sigma^{-1}\mu \\ g &= \log p - \tfrac{1}{2}\mu' K\mu \end{aligned}$$

K is often called the precision matrix. We can write the above in scalar form:

$$\phi(x; g, h, K) = \exp\left(g + \sum_i h_i x_i - \tfrac{1}{2}\sum_i \sum_k K_{ij} x_i x_j\right)$$

Now $X \perp Y | Z$ iff the density factors as $P(X, Y, Z) = g(X, Z)h(Y, Z)$ for some functions g and h. Hence we see that $X_i \perp X_j|\text{rest}$ iff $K_{ij} = 0$.

Note that we can only convert from canonical to moment form if K is positive semi definite (psd), which it need not be in general (see Section B.5.1). Similarly, we can only convert from moment to canonical form if Σ is psd; but this will always be the case, unless we have deterministic linear CPDs (in which case $\Sigma = 0$). Note that potentials need not be probability distributions, and need not be normalizable (integrate to 1). We keep track of the constant terms (p or g) so we can compute the likelihood of the evidence.

Outline of algorithm

For each node X_i, we convert X_i's CPD to a potential, F_i; if there is evidence on any of X's family, we enter it into F_i; we then multiply F_i onto a clique potential containing X's family. Then we calibrate the jtree as before, but using the new definitions of multiplication, division and marginalization.

We must represent the initial potentials in canonical form, because they may represent conditional likelihoods, not probability distributions (c.f., the distinction between β and γ in an HMM). For example, consider Figure B.15. The initial clique potential for $1, 2$ will be $P(X_1)P(X_2|X_1) = P(X_1, X_2)$, which is a joint pdf, but the initial potential for $2, 3$ will be $P(X_3|X_2)$, which is a conditional likelihood. Conditional likelihoods can be represented in canonical form, but not in moment form, since the (unconditional) mean does not exist. (We can only compute $E[X_3]$ from $P(X_3|X_2)$ once $P(X_2)$ becomes available.) This will be explained in more detail below.

Since the potentials are initially created in canonical form, we perform all the other operations (multiplication, division and marginalization) in this form, too. However, at the end, we usually convert to moment form, since that is easier for the user to understand, and is also a more convenient form for learning (we simply sum the means and covariances). Note that when this algorithm is applied to a Kalman Filter Model, the result is identical to the information form of the Kalman smoother.

Converting a linear-Gaussian CPD to a canonical potential

For a vector node, the conditional distribution has the form

$$\begin{aligned} f(x|z) &= c\exp\left[-\tfrac{1}{2}\left((x-\mu-B^T z)^T\Sigma^{-1}(x-\mu-B^T z)\right)\right] \\ &= \exp\left[-\tfrac{1}{2}\begin{pmatrix} x & z\end{pmatrix}\begin{pmatrix} \Sigma^{-1} & -\Sigma^{-1}B^T \\ -B\Sigma^{-1^T} & B\Sigma^{-1}B^T\end{pmatrix}\begin{pmatrix} x \\ z\end{pmatrix} + \begin{pmatrix} x & z\end{pmatrix}\begin{pmatrix} \Sigma^{-1}\mu \\ -B\Sigma^{-1}\mu\end{pmatrix} - \tfrac{1}{2}\mu^T\Sigma^{-1}\mu + \log c\right] \end{aligned}$$

where $c = (2\pi)^{-n/2}|\Sigma|^{-\frac{1}{2}}$. Hence we set the canonical characteristics to

$$\begin{aligned} g &= -\tfrac{1}{2}\mu^T\Sigma^{-1}\mu - \frac{n}{2}\log(2\pi) - \tfrac{1}{2}\log|\Sigma| \\ h &= \begin{pmatrix} \Sigma^{-1}\mu \\ -B\Sigma^{-1}\mu\end{pmatrix} \\ K &= \begin{pmatrix} \Sigma^{-1} & -\Sigma^{-1}B^T \\ -B\Sigma^{-T} & B\Sigma^{-1}B^T\end{pmatrix}. = \begin{pmatrix} I \\ -B\end{pmatrix}\Sigma^{-1}\begin{pmatrix} I & -B\end{pmatrix} \end{aligned}$$

This generalizes the result in [Lau92] to the vector case. In the scalar case, $\Sigma^{-1} = 1/\sigma$ (so σ represents the variance, not the standard deviation), $B = b$ and $n = 1$, so the above becomes

$$\begin{aligned} g &= \frac{-\mu^2}{2\sigma} - \tfrac{1}{2}\log(2\pi\sigma) \\ h &= \frac{\mu}{\sigma}\begin{pmatrix} 1 \\ -b \end{pmatrix} \\ K &= \frac{1}{\sigma}\begin{pmatrix} 1 & -b^T \\ -b & bb^T \end{pmatrix}. \end{aligned}$$

Once we have the canonical characteristics, we can compute the initial potentials for each clique by multiplying together the potentials associated with each variable which is assigned to this clique. Unfortunately, we cannot convert these canonical characteristics to moment characteristics because K is not of full rank, and hence is not invertible.

Entering evidence into a canonical potential

If we observe that a continuous variable y takes on a specific value y, we must modify the potentials of all the cliques/separators that contain y, since their dimensionality will be reduced. Consider a clique with domain (x, y). The new potential is

$$\begin{aligned} \phi^*(x) &= \exp[g + \begin{pmatrix} x^T & y^T \end{pmatrix}\begin{pmatrix} h_X \\ h_Y \end{pmatrix} - \tfrac{1}{2}\begin{pmatrix} x^T & y^T \end{pmatrix}\begin{pmatrix} K_{XX} & K_{XY} \\ K_{YX} & K_{YY} \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} \\ &= \exp[\left(g + h_Y^T y - \tfrac{1}{2}y^T K_{YY} y\right) + x^T(h_X - K_{XY}y) - \tfrac{1}{2}x^T K_{XX} x] \end{aligned}$$

This generalizes the corresponding equation in [Lau92] to the vector case.

Multiplication and division of canonical potentials

In the discrete case, we use multiplication and division to update potentials when new evidence arrives: $\Pr^*_W = \Pr_W \frac{\Pr^*_S}{\Pr_S}$, where S is a separator and W is a clique. Notice that $\frac{\Pr_W}{\Pr_S} = \Pr(W|S)$, so we are really computing a conditional distribution "on the fly" and multiplying in new information.

We can define multiplication and division in the Gaussian case in terms of canonical characteristics, as follows. To multiply $\phi_1(x_1, \ldots, x_k; g_1, h_1, K_1)$ by $\phi_2(x_{k+1}, \ldots, x_n; g_2, h_2, K_2)$, we extend them both to the same domain $x_1, \ldots, x_n$ by adding zeros to the appropriate dimensions, and compute

$$(g_1, h_1, K_1) * (g_2, h_2, K_2) = (g_1 + g_2, h_1 + h_2, K_1 + K_2)$$

Division is defined as follows:

$$(g_1, h_1, K_1)/(g_2, h_2, K_2) = (g_1 - g_2, h_1 - h_2, K_1 - K_2)$$

Marginalization of a canonical potential

Let ϕ_W be a potential over a set W of variables. We can compute the potential over a subset $V \subset W$ of variables by marginalizing, denoted $\phi_V = \sum_{W \setminus V} \phi_W$. Let

$$y = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}, \qquad h = \begin{pmatrix} h_1 \\ h_2 \end{pmatrix}, \qquad K = \begin{pmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{pmatrix},$$

with y_1 having dimension p and y_2 having dimension q. It can be shown that

$$\int_{y_1} \phi(y_1, y_2; g, h, K) = \phi(y_2; \hat{g}, \hat{h}, \hat{K})$$

where

$$\begin{aligned} \hat{g} &= g + \tfrac{1}{2}\left(p\log(2\pi) - \log|K_{11}| + h_1^T K_{11}^{-1} h_1\right) \\ \hat{h} &= h_2 - K_{21}K_{11}^{-1}h_1 \\ \hat{K} &= K_{22} - K_{21}K_{11}^{-1}K_{12} \end{aligned}$$

B.5.2 Conditional Gaussian case

In the CG case, potential functions can be represented as a list of canonical characteristics, one for each possible combination of discrete values. In addition to g, h and K, we store $\chi(i)$, which is a multiplicative constant outside of the $\exp$ operator; if $\chi(i) = 0$, it means the i'th element has probability 0; otherwise we set $\chi(i) = 1$. All the operations go through as before, except for marginalization, which we discuss below.

Strong marginalization

If we marginalize out over some continuous nodes, we can proceed as in Section B.5.1 above, once for each value of the discrete nodes, i.e.,

$$\int_{y_1} \phi(i, y_1, y_2; g(i), h(i), K(i)) = \phi(i, y_2; \hat{g}(i), \hat{h}(i), \hat{K}(i))$$

If we marginalize out over some discrete variables, say j, we distinguish two cases. First, if h and K do not depend on j, i.e., $h(i, j) = h(i)$ and $K(i, j) = K(i)$, then we define

$$\begin{aligned} \sum_j \phi(i, j, y) &= \sum_j \chi(i, j) \exp\left(g(i, j) + h(i)'y - \tfrac{1}{2} y' K(i) y\right) \\ &= \exp\left(h(i)'y - \tfrac{1}{2} y' K(i) y\right) \times \sum_j \chi(i, j) \exp g(i, j) \end{aligned}$$

Hence

$$\hat{g}(i) = \log \sum_{j : \chi(i,j)=1} \exp g(i, j), \quad \hat{h}(i) = h(i), \quad \hat{K}(i) = K(i).$$

If h or K depends on j, we need to perform "weak marginalization", which we discuss next.

Weak marginalization

CG potentials are not closed under addition. For example, suppose i and j have K possible values each, so $\phi(y, i, j)$ is a mixture of K^2 Gaussians. When we marginalize out j, we are still left with a mixture of K^2 Gaussians:

$$\sum_j \phi(x, i, j; \mu_{i,j}, \Sigma_{i,j}) \propto \sum_j \phi(x, i; \mu_{i,j}, \Sigma_{i,j})$$

This cannot be simplified any further, and must be kept as a list of terms. If this is then multiplied by another mixture of K Gaussians, we end up with K^3 terms. One way to stop this exponential explosion is to "collapse" the mixture of K^2 Gaussians to a mixture of K Gaussians. We would like to do this in an optimal way, i.e., minimize the KL distance between the true distribution, $\sum_j \phi(x, j, i)$, and the approximate distribution, $\tilde{\phi}(x, i)$, on a case by case basis (i.e., for each i). We can do this by moment matching (for each i). This requires that we convert ϕ to moment form, and hence requires that $K(i, j)$ be psd. We then proceed as follows.

$$\begin{aligned} \hat{p}_i &= \sum_j p_{ij} \\ \hat{p}_{i|j} &= p_{ij} / \hat{p}_i \\ \hat{\mu}_i &= \sum_j \mu_{ij} \hat{p}_{i|j} \\ \hat{\Sigma}_i &= \sum_j \Sigma_{ij} \hat{p}_{i|j} + \sum_j (\mu_{ij} - \hat{\mu}_i)(\mu_{ij} - \hat{\mu}_i)^T \hat{p}_{i|j} \end{aligned}$$

If we use a strong junction tree (see Section B.3.6), then all marginalizations on the collect phase are strong, hence we do not "inherit" any errors. Hence the above operation will give the "correct" mean and variance. However, even though the first two moments are correct (and hence this is the optimal single Gaussian approximation), they may not be meaningful if the posterior is multimodal. An (expensive) alternative

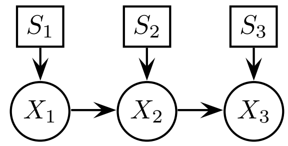


Figure B.17: A switching Markov chain. Square nodes are discrete, round nodes are continuous.

is to use computer algebra to compute the exact posterior symbolically [CGH97, Cur97]. (The paper [CF95], whose title is "Symbolic Probabilistic Inference with both Discrete and Continuous Variables", sounds like it's doing this, but in fact SPI is just another name for the variable elimination algorithm, which is numerical.)

Note also that a special case of these rules were derived for Pearl's polytree algorithm in [DM95].

Computational complexity of exact CG inference

[LSK01] prove that inference in CG networks is NP-hard, even if the structure is a (poly)tree, such as the one in Figure B.17. The proof is by reduction from subset sum, but the intuition is as follows: the prior $P(X_1)$ is a mixture of K Gaussians, depending on S_1; when we marginalize out S_1, we are still left with a mixture of K Gaussians; each one of these gets "passed through" K different matrices, depending on the value of S_2, resulting in a mixture of K^2 Gaussians, etc; in general, the belief state at time t is a mixture of K^t Gaussians.

The problem above arose because we used a strong jtree. This requires us to eliminate all the discrete nodes before any of the continuous nodes; hence we are forced to form one clique which contains all the discrete nodes.[5] Unfortunately, in the case of hybrid DBNs, the need to eliminate all the continuous nodes before their discrete ancestors clashes with our desire to eliminate all the nodes in slice t before we eliminate any in slice $t+1$. Hence it is common to remove the strong triangulation requirement, resulting in the jtree in Figure 4.13. In this case, both the collect (forwards) and the distribute (backwards) passes will involve weak marginalization. This is equivalent to the standard moment-matching approximation for filtering in switching Kalman filter models (see e.g., [TSM85, BSL93, Kim94, WH97]). We discuss how to improve on this using expectation propagation in Section B.7.2.

Numerically stable CG inference

The scheme outlined above, whether exact (using a strong jtree) or approximate (using a regular jtree or even an untriangulated graph), is subject to numerical errors because of the need to convert between canonical and moment form. In addition, the fact that we represent initial clique potentials in canonical form means we cannot use deterministic linear CPDs (with $\Sigma = 0$), which can be useful for some models. A solution to both of these problems is proposed in [LJ01]. Unfortunately, the incorporation of evidence is quite different from the above framework, and requires fairly significant changes to the architecture.

B.5.3 Arbitrary CPDs

Exact inference algorithms have (so far) only been derived for networks which satisfy the following restrictions: all hidden nodes must be discrete with discrete parents, or have conditionally linear Gaussian (CLG) CPDs (which of course includes as special cases linear Gaussian and unconditional Gaussian distributions). Observed nodes may have any distribution if all their parents are discrete or observed, but must have have CLG CPDs if they have any hidden continuous parents.

These restrictions rule out many combinations, but a particularly useful one would be discrete nodes with hidden continuous parents (e.g., using a softmax CPD); this can be used to model switching/threshold behavior as well as for dimensionality reduction of discrete data [Tip98]. I proposed a variational approximation for this case in [Mur99], based on [JJ00]; [Wie00] extended this to a mixture variational approximation. [LSK01] proposed a hybrid jtree/numerical integration algorithm for this problem.

[5] For example, consider Figure B.17. With the elimination ordering $X_1, \ldots, X_T, S_1, \ldots, S_T$, the cliques are $\{S_1, S_2, X_1, X_2\}, \ldots, \{S_{1:t}, X_{t-1}, X_t\}, \ldots, \{S_{1:T}, X_{T-1}, X_T\}$.

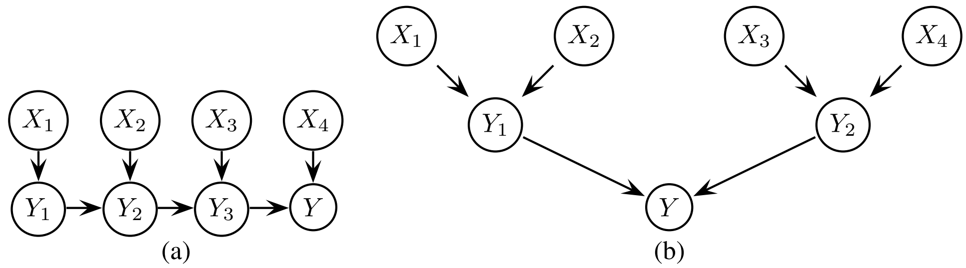


Figure B.18: A noisy-OR CPD for $P(Y|X_{1:4})$ represented in a more computationally revealing form. (a) The temporal transformation. (b) Parent divorcing.

In general, the current state of the art for general continuous CPDs is to use sampling methods (see Section B.7.4). (Of course, it is always possible to discretize all the continuous variables, or to apply numerical integration, but such approaches usually suffer from the curse of dimensionality.)

B.6 Speeding up exact discrete inference

There are at least three kinds of indepencies we can exploit to speed up exact inference. The first is conditional independence, which is reflected in the graph structure; we have already discussed how to exploit this. The second is causal independence (e.g., noisy-OR), and the third is context specific independence (CSI). We discuss these, and other tricks, below.

B.6.1 Exploiting causal independence

CPDs with causal independence were introduced in Section A.3.2. The canonical example is noisy-OR. Pearl [Pea88] showed how to exploit the structure of the noisy-OR CPD to compute the λ and π messages in time linear in the number of parents; [ZP96, RD98] showed how to exploit it in variable elimination, and [Hec89] showed how to exploit in BN20 (binary node 2-level noisy-OR) networks such as QMR-DT using the Quickscore algorithm.

To exploit causal independence in the jtree algorithm, we have to make graphically explicit the local conditional independencies which are "hidden" inside the CPD. The two standard techniques for this are the temporal transformation [Hec93, HB94] and parent divorcing [OKJ$^+$89]: see Figure B.18. We introduce extra hidden nodes Y_i which accumulate a partial-OR of previous parents. In general, to benefit from such a transformation, we must eliminate parents before children, otherwise we end up creating a clique out of the original family. However, sometimes this is not the best ordering. See [RD98] for a discussion of this point; see also [ZY97, MD99].

Interestingly, it is not always possible to exploit causal independence if we are doing max-product (Viterbi), as opposed to sum-product, because we must always sum out the dummy hidden nodes; however, the max and sum operators do not commute, so this imposes a restriction on the possible orderings (c.f., Section B.3.6) which can eliminate any potential gains.

B.6.2 Exploiting context specific independence (CSI)

CPDs with CSI were introduced in Section A.3.2. The canonical example is a tree-CPD. Such CPDs can be exploited for inference by the jtree algorithm using a network transformation [BFGK96], as illustrated in Figure B.19. This is analogous to the transformations introduced to exploit causal independence (see Section B.6.1). More aggressive optimizations are possible if we use the variable elimination algorithm [Zha98b, ZP99]. A way of exploiting CSI in the jtree algorithm is discussed in [Pfe01].

Tree-CPDs have been widely used for many years and are especially popular in the decision making community, e.g., see [BDG01] for a review of the work on factored Markov decision processes (MDPs). [Kim01, ch5] discusses how to use trees to do "structured linear algebra". Recently the MDP community has started to investigate algebraic decision diagrams (ADDs) [BFG$^+$93], which are very computationally

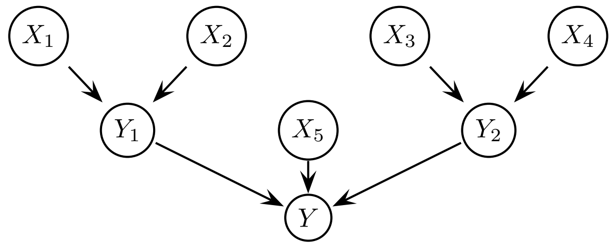


Figure B.19: A tree-structured CPD for $P(Y|X_{1:5})$ represented in a more computationally useful form. Let all nodes be binary for simplicity. We assume Y is conditionally independent of $X_{3:4}$ given that $X_5 = 1$, i.e., $P(Y|X_{1:4}, X_5 = 1) = P(Y_1|X_1, X_2) = f(Y, X_1, X_2)$. Similarly, assume $P(Y|X_{1:4}, X_5 = 2) = P(Y_2|X_3, X_4) = g(Y, X_3, X_4)$. Hence X_5 is a switching parent: $P(Y|Y_1, Y_2, X_5 = i) = \delta(Y, Y_i)$. This tree requires 2^3 parameters, whereas a table would require 2^5; furthermore, the clique size is reduced from 6 to 4. Obviously the benefits are (potentially) greater for nodes with more parents and for nodes with bigger domains.

efficient ways of representing and manipulating real-valued functions of binary variables. These are more efficient than trees because they permit sharing of substructure; just as important, there is a fast, freely-available program for manipulating them called CUDD.[6] ADDs have also been used for speeding up exact inference in discrete Bayes nets [NWJK00].

B.6.3 Exploiting deterministic CPDs

Deterministic discrete CPDs are quite common, as we saw in the HHMM models in Chapter 2. (Multiplexer nodes are also deterministic.) This means the resulting CPD has lots of zeros. The technique of "zero compression" removes 0s from the potentials so that multiplication and marginalization can be performed more efficiently, without any loss in accuracy [JA90, HD96].

[Zwe98] develops a technique for exploiting deterministic CPDs in Pearl's directed message passing algorithm on a jtree. Specifically, he requires the jtree satisfy what he calls IRP (the immediate resolution property), which says that, in a preorder traversal of the tree (root to leaves), the first time a deterministic node appears in a clique, its parents must also be present. This can be ensured by eliminating parents before children, i.e., in some total ordering of the DAG: see Section B.3.6. The advantage of this is that it is possible to detect which elements of $P(C_c|C_p)$ will be zero, where C_c is the child clique and C_p is its parent (one or the other of these can be separators). It is not clear if this technique is relevant to the undirected form of message passing, which does not require computation of terms like $P(C_c|C_p)$. In particular, it seems that using sparse potentials should achieve the same effect.

B.6.4 Exploiting the evidence

Sometimes, with deterministic CPDs, evidence on the child or one or more parents can render the remaining family members "effectively observed", e.g., in an OR-gate, if the child is off, all the parents must be off, or if any of the parents is on, the child must be on. Such constraints can be exploited by running arc consistency before probabilistic message passing, to reduce the effective domain size of each node. This technique is widely used in genetic linkage analysis [FGL00].

B.6.5 Being lazy

In general there is a tradeoff between being lazy, i.e., waiting until the query, evidence, structure and parameters have all been specified, and being eager, i.e., precomputing stuff as soon as possible, so the cost can be amortized across many future operations (e.g., we would not want to create a jtree from scratch every time the evidence or query changed). Usually the jtree is constructed based only on the graph structure. However, by constructing the jtree later in the pipeline, we can avail of the following kinds of information:

[6] http://vlsi.colorado.edu/ fabio/CUDD/cuddIntro.html

- If we know how many values each node can take on, we know how "heavy" the cliques will be; this will affect the search for the elimination ordering.
- If we know what kinds of CPD each node has, we can perform network transformations if necessary. Also, we know if we need to construct a strong jtree or not.
- If we know which nodes will be observed, it will affect how many values each node can have, the kinds of CPDs it can support, whether we need to build a strong jtree, etc.
- If we know the parameter values, we might be able to perform optimizations such as zero compression.
- If we know which nodes will be queried and which nodes will be observed, we can figure which nodes are relevant for answering the query.
- If we know the values of the evidence nodes, we might be able to do some pre-processing (e.g., constraint satisfaction) before running probabilistic inference.

Most inference routines choose an elimination ordering based on the first two or three pieces of information. Variable elimination can easily exploit the remaining information, as can lazy Hugin [MJ99]. Query-DAGs [DP97] are precompiled structures for answering specific queries, and can be highly optimized.

B.7 Approximate inference

Exact inference in discrete networks is exponential in the size of the largest clique. The techniques in Section B.6 can help for certain networks, but are generally no use for large structured networks such as grids. (The max clique for an $N = n \times n$ grid has size $O(\sqrt{N})$.) In the worsr case, exact inference is exponentially hard (in fact exact inference is known to be #P-hard, i.e., strictly harder than NP-hard [DL93]). We therefore need to resort to approximations.

Even if the graph is tree structured, exact inference in networks with hidden continuous variables is only possible in very restricted circumstances, as mentioned in Section B.5.3. We therefore need to resort to approximations in this case, too.

Approximate inference is a huge subject. In this section, I concentrate on methods that are very similar (algorithmically speaking) to the exact message passing schemes we have just discussed; this makes them efficient and easy to implement. For completeness, I briefly mention some other approaches at the end.

B.7.1 Loopy belief propagation (LBP)

Belief propagation is a way of computing exact marginal posterior probabilities in graphs with no undirected cycles (loops); essentially it generalises the forwards-backwards algorithm to trees (see Section B.4). To be sure of getting exact results, we must convert a graph with loops into a junction tree (jtree) (see Section B.3), and run belief propagation on the resulting jtree. Unfortunately, the nodes in the jtree (which are cliques in the original graph) may be exponentially large, making this procedure prohibitively slow.

One natural approximation, first suggested in [Pea88], is to apply belief propagation to the original graph, even if it has loops; this is often called "loopy belief propagation" (LBP). In principle, this runs the risk of double counting information, and of either not converging or of converging to the wrong answer. However, in practice the method often works surprisingly well. The most important success story has been decoding messages sent over noisy channels at rates near to the Shannon limit (see [MMC98] and references therein), but the technique has also been succesfully applied to general Bayes nets [MWJ99], 2D lattice MRFs [FPC00, Wei01], etc. Various improvements to the algorithm have been made, including versions that always converge (e.g., [Yui00, WT01]) and versions that yield higher accuracy by considering embedded trees [WJW01] or higher-order interactions (beyond pairwise) [YFW01] (the Kikuchi method).

A number of theoretical results have now been proved about LBP, explaining why it gives exact answers for networks in which all nodes are Gaussian [WF99], for networks in which there is only a single loop [Wei00], and for general networks but using the max-product (Viterbi) version instead of the sum-product version of the algorithm [FW00]. Connections to variational methods have also been made [WT02, TJ02]. In general, however, all we can say is that, if the algorithm converges, it will reach a stationary point (in practice, usually a local minimum) of the Bethe free energy [YFW00]. (This ensures things are locally, but perhaps not globally, consistent.)

The LBP algorithm

To apply LBP to a Bayes net, we can use Pearl's formulation in terms of λ and π messages. But how do we apply LBP to an MRF? After all, the nodes do not have local terms associated with them. For pairwise MRFs, it is simple to formulate the message passing rules in terms of matrix-vector multiplication [Wei00]; the pairwise Gaussian case is also easy [WF99]. Any MRF with potentials defined on larger sets of nodes can be converted into a pairwise MRF by creating mega-nodes [WF99], but this is not always desirable, since the state-space of the mega-nodes will be much larger, and hence computing the messages may be exponentially slower. It is possible to derive the rules for arbitrary MRFs, but they are a bit messy. Besides, we would like to use the same code for Bayes nets and MRFs. We therefore convert both BNs and MRFs to factor graphs (see Section A.4). We now derive the LBP algorithm for factor graphs (fgraphs).

The basic "rules" for message passing in an fgraph are as follows. Each variable node x sends a message to each neighboring factor node f of the form

$$\mu_{x \to f}(x) = \prod_{g \neq f} \mu_{g \to x}(x)$$

and each factor node f sends a message to each neighboring variable node x of the form

$$\mu_{f \to x}(x) = \sum_{u \backslash x} f(u) \prod_{y \neq x} \mu_{y \to f}(y)$$

where we have assumed u is the domain of f. Obviously we could update all nodes in parallel, but in Figure B.20, I give a more efficient serial updating scheme. If the fgraph is tree structured, one forwards and one backwards sweep suffices to compute the exact marginals at every node. The serial protocol uses an arbitrary, but fixed, order. For chains, the "natural" ordering is left-to-right, for trees it is leaves-to-root (pre-order), and for DAGs, it is a topological ordering. (Undirected graphs don't have a natural ordering, so we can pick any one.) Each factor node divides its neighboring variables into its predecessors (lower in the ordering), and its successors (higher in the ordering). On the forwards sweep, a node combines all the incoming messages from its predecessors, and puts a message on its out arcs to each of its successors. On the backwards sweep, a node combines all its incoming messages from its successors, and puts a message on its out arcs to each of its predecessors.

Damping

LBP can result in posterior "marginals" that oscillate. Previous experience [MWJ99] has shown that one effective solution to this problem is to only update the messages "partially" towards their new value at each iteration. That is, we use a convex combination

$$(1 - m)\mu^t + m\mu^{t-1}$$

of the new message and its previous value, where $0 \leq m \leq 1$ is a "momentum" term and μ represents a message. $m = 0$ corresponds to the usual LBP algorithm; $m = 1$ corresponds to no updating at all. It is easy to see that fixed points of this modified algorithm are also fixed points of the original system (since if $\mu^t = \mu^{t-1} = \mu$, then $(1 - m)\mu^t + m\mu^{t-1} = \mu$).

If (damped) LBP does not converge, we can use one of several alternative algorithms that directly minimizes the Bethe free energy and which always converge [WT01]. However, empirically it has been found that in such cases, the accuracy of the answers is poor, i.e., oscillation seems to be a sign that the Bethe/LBP approximation is a bad one [WT02].

```
function [φ, μ] = BP-fgraph(F, order, max-iter, tol)
iter = 1
converged = false
initialize: φ_j = ∏_{i∈ass(j)} F_i, φ_x = 1, μ_{f→x} = 1, μ_{x→x} = 1
while (not converged) and (iter ≤ max-iter)
        [φ, μ] = sweep(φ, μ, order)
        [φ, μ] = sweep(φ, μ, reverse(order))
        converged = test-convergence(φ, μ, tol)
        iter = iter + 1

function [φ, μ] = sweep(φ^old, μ^old, order)
for each factor f in order
    φ_f = φ_f^old
    for each variable x in pred(f, order)
        μ_{x→f} = (φ_x^old)/(μ_{f→x}^old)
        φ_f = φ_f × (μ_{x→f})/(μ_{x→f}^old)
    for each variable x in succ(f,order)
        μ_{f→x} = (φ_f ↓ x)/(μ_{x→f}^old)
        φ_x = φ_x^old × (μ_{f→x})/(μ_{f→x}^old)
```

Figure B.20: Pseudo-code for belief propagation on a factor graph using a serial updating protocol.

Efficient computation of messages

Computing a message from a factor to a variable takes $O(K^{F_{in}})$ time, where K is the number of discrete values a node can take on, and F_{in} is the fan-in of the factor (one plus the number of parents of a node in the case of a BN, or the number of terms in the clique potential in the case of an MRF). It is sometimes claimed (e.g., [Bar01]) that undirected message passing is therefore more efficient, but in fact it has exactly the same time complexity: when we convert the Bayes net to an MRF, we need to moralize the parents, creating a clique of size $K^{F_{in}+1}$. In general, message passing with tabular representations of CPDs/potentials always takes time exponential in the clique size.

[Pea88] shows how to exploit the structure of the noisy-OR CPD to compute the λ and π messages in time linear in the number of parents, instead of exponential. This method can be generalized to any CPD that enjoys the property of "causal independence" [RD98], including multiplexer nodes. In the fgraph context, this means that certain kinds of factor nodes will have specialized "methods" for computing messages; furthermore, there will be an asymmetry between messages sent to children and message sent to parents. [Bar01] shows how to compute the messages for any CPD which is a linear function of its parents using a Fourier transform, which can sometimes be approximated using saddle point techniques.

If the factor is a mixture of Gaussians, we may have to use weak marginalization (see Section B.5.2) when computing the message.

If the fgraph was derived from a jtree, the cost of computing a message is $O(K^{w^*})$ in the worst case, where w^* is the induced width of the graph (i.e., maximal clique size). Furthermore, since potentials (factors) may be products of many different types of CPDs, it is much harder to exploit local structure, unless it is exposed graphically (see Section B.6.2).

LBP on a jtree

If the fgraph is a tree, LBP will give the exact answers. Any graph can be converted to a (junction) tree, as we discussed in Section B.3. If we could convert this jtree to an fgraph and run LBP, we would get exact answers. This would allow us to use a single inference algorithm (shown in Figure B.20) for both exact and approximate inference. We can convert a jtree to an fgraph by making the following correspondence: the factors correspond to cliques and the variables correspond to separators.[7]

In the case of a jtree, a separator (variable) is always connected to exactly two cliques (factors). This means we can simplify the algorithm. For example, suppose j is a variable node (separator) and i and k are neighboring factors (cliques), The variable node (separator) computes its product of incoming messages, $\phi_j = \mu_{i\to j} \times \mu_{k\to j}$, and then sends out a new message, $\mu_{j\to k} = \phi_j/\mu_{k\to j} = \mu_{i\to j}$. Factor node (clique) k then absorbs this message. This is clearly equivalent to factor node (clique) i sending $\mu_{i\to j}$ to factor node (clique) k directly. Hence it is sufficient for cliques to send messages to each other directly: separators do not need to send messages; instead, they simply store the potentials (messages). This simplification results in the JLO algorithm in Figure B.13.

B.7.2 Expectation propagation (EP)

Expectation propagation (EP) [Min01] is like belief propagation except it requires that the posteriors (beliefs) on each variable have a restricted form. Specifically, the posterior must be in the exponential family, i.e., of the form $q(\theta) \propto \exp(\gamma' f(\theta))$, where θ is a variable. This ensures that beliefs can be represented using a fixed number of sufficient statistics. We choose the parameters of the beliefs s.t.

$$\gamma^* = \arg\min_{\gamma} D(p(\theta)||q_\gamma(\theta))$$

where

$$p(\theta) = \frac{q^{prior}(\theta) \times t(\theta)}{Z}$$

is the exact posterior, $Z = \int_\theta q^{prior}(\theta) \times t(\theta)$ is the exact normalizing constant, $t(\theta)$ is the likelihood term (message coming in from a factor) and $q_\gamma(\theta)$ is the approximate posterior. This can be solved by moment matching. When we combine sequential Bayesian updating with this approximation (projection) after every update step, the result is called Assumed Density Filtering (ADF), or the probabilistic editor [SM80]. It is clear that the BK algorithm (see Section 4.2.1) is an example of ADF, as is the standard GBP algorithm for switching Kalman filters (see Section 4.3).

One drawback with ADF is its dependence on the order in which the updates are performed. EP, which is a batch algorithm, reduces the sensitivity to ordering by iterating. Intuitively, this allows EP to go back and reoptimize each belief in the revised context of all the other updated beliefs. This requires that we store the messages so their effect can be later "undone" by division. In EP, rather than approximating the messages directly (which is hard, since they represent conditional likelihoods), we approximate the posterior using moment matching, and then infer the corresponding equivalent message. We explain this below.

To explain the difference between EP and BP procedurally, consider the simple factor graph shown in Figure B.22. We send a message from f to x, and then update x's belief, as follows:

$$\begin{aligned}
\phi_x^{prior} &= \phi_x/\mu_{f\to x}^{old} = \mu_{g\to x}^{old} \\
\phi_f^{prior} &= \phi_f/\mu_{x\to f}^{old} = f(x,y)\mu_{y\to f}^{old}(y) \\
\mu_{f\to x} &= \phi_f^{prior} \downarrow x = \int_y f(x,y)\mu_{y\to f}^{old}(y) \\
\phi_x &= \phi_x^{prior} \times \mu_{f\to x} = \mu_{g\to x}^{old} \times \mu_{f\to x}
\end{aligned}$$

The terms after the second equality on each line are for this particular example.

In EP, we compute the approximate posterior ϕ_x first, and then derive the message $\mu_{f\to x}$, which, had it been combined with the prior ϕ_x^{prior}, would result in the same approximate posterior:

$$\begin{aligned}
\phi_x^{prior} &= \phi_x^{old}/\mu_{f\to x}^{old} \\
\phi_f^{prior} &= \phi_f/\mu_{x\to f}^{old} \\
(\phi_x, Z) &= ADF(\phi_x^{prior} \times \phi_f^{prior} \downarrow x) \\
\mu_{f\to x} &= (Z\phi_x)/(\phi_x^{prior})
\end{aligned}$$

[7] As far as I know, this is a novel observation.

function $[\phi, \mu]$ = sweep(ϕ^{old}, μ^{old}, order)
for each factor f in order
    $\phi_f = \phi_f^{old}$
    for each variable x in pred(f, order)
        $\mu_{x \to f} = (\phi_x^{old}) / (\mu_{f \to x}^{old})$
        $\phi_f = \phi_f \times (\mu_{x \to f}) / (\mu_{x \to f}^{old})$
    for each variable x in succ(f,order)
        $\phi_x^{prior} = \phi_x^{old} / \mu_{f \to x}^{old}$
        $\phi_f^{prior} = \phi_f / \mu_{x \to f}^{old}$
        $(\phi_x, Z) = ADF(\phi_x^{prior} \times \phi_f^{prior} \downarrow x)$
        $\mu_{f \to x} = (Z\phi_x) / (\phi_x^{prior})$

Figure B.21: Pseudo-code for expectation propagation on a factor graph using a serial updating protocol. This sweep function is called iteratively, as in Figure B.20.

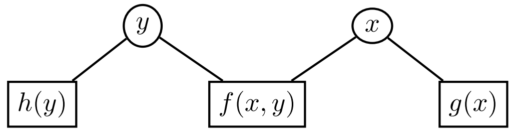


Figure B.22: A simple factor graph. Square nodes are factors, round nodes are variables.

where $(q, Z) = ADF(p)$ produces the best approximation q to p within a specified family of distributions. The code for EP is shown in Figure B.21.

For example, consider the switching Markov chain in Figure B.17. The factor graph for the first few slices is shown in Figure B.22, where the variables are the separators, $y = (S_1, X_1)$, $x = (S_2, X_2)$, etc., and the factors are the clique potentials, $h(y) = P(S_1)P(X_1|S_1)$, $f(x, y) = P(S_2|S_1)P(X_2|X_1, S_1)$, etc. When f sends a message to x, it needs to compute

$$\phi_x(X_2, S_2) = ADF\left(\int_{X_1} \sum_{S_1} \phi_x^{prior}(X_2, S_2) \times \phi_f^{prior}(X_2, S_2, X_1, S_1)\right)$$

which can be done by weak marginalization (moment matching for a Gaussian mixture).

In the special case in which we only have a single variable node (as Minka typically assumes), things simplify, since the factor nodes do not contain any other variables, (e.g., $\phi_f^{prior} = f(x)$, as opposed to $\phi_f^{prior} = f(x, y) \times \mu_{y \to f}(y)$), and hence the projection onto x is unnecessary. In this case, the ADF step simplifies to

$$(\phi_x, Z) = ADF(\phi_x^{prior} \times f(x))$$

To map this to Minka's notation, use: $x = \theta$, $\phi_x = q(\theta)$, and $f(x) = t_i(\theta)$.

We now discuss how to implement $(q(\theta), Z) = ADF(q^{prior}(\theta) \times t(\theta))$ for different kinds of distribution $q(\theta)$ and likelihood terms $t(\theta)$.

EP with a Gaussian posterior

We can compute the parameters of an approximate Gaussian posterior $q(\theta)$, as follows:

$$\begin{aligned} m_{post} &= E_p[\theta] = m + V\nabla_m \\ V_{post} &= E_p[\theta\theta'] - [E_p\theta][E_p\theta]' = V - V(\nabla_m \nabla_m' - 2\nabla_V)V \end{aligned}$$

where $m = m_{prior}$, $V = V_{prior}$, $\nabla_m = \nabla_m \log Z(m, V)$, $\nabla_V = \nabla_V \log Z(m, V)$ and $Z(m, V) = \int_\theta t(\theta) q^{prior}(\theta; m, V)$ [Min01, p15,p45]. These equations are properties of the Gaussian distribution and hold for any $t_i(\theta)$. Minka works the details out for the example of a likelihood term that is a mixture of Gaussians (modelling a point buried in clutter):

$$t_i(\theta) = (1-w)\mathcal{N}(x_i; \theta, I) + w\mathcal{N}(x_i; 0, 10I)$$

for a fixed, known w.

Once we have approximated the beliefs, we can compute the new message by division. In the case of Gaussians, the easiest way to do this is to convert the beliefs to canonical form, and then divide. Note that this division might result in a negative or infinite variance term (since division in canonical form is implemented by subtracting precision matrices). A negative variance represents a function that curves upwards (like a U), rather than the usual downward (bell-shaped) curve of a Gaussian; an infinite variance corresponds to a flat (uniform) distribution; zero variance corresponds to a constant. It is okay if messages have negative variance, but a belief with a negative variance is a problem. In this case, one crude approximation is to to replace a negative variance with an infinite variance, which means the absorbing variable will ignore this message (since it is completely uncertain). This will always result in convergence, but [Min01, p22] says that "when EP does converge with negative v_i's, the result is always better than having forced $v_i > 0$" (where the variance is $v_i I$ in the case of a spherical Gaussian).

EP with a Dirichlet posterior

It is possible to use EP to compute a Dirichlet approximation to $P(\theta|y)$ even when the likelihood functions have the form of a mixture, $P(y_i|\theta) = \sum_z P(z|\theta)P(y_i|z)$. This has been applied to a model in which the y's represent words, the z's represent latent topics, and θ is a distribution over topics for a particular document [ML02].

EP with mixed types of posterior

It is not clear how to use EP for arbitrary factor graphs, where the belief on each factor (and hence the corresponding outgoing messages) might be a different member of the exponential family. This is a subject for future research.

EP with a fully factorized posterior is LBP

Minka points out that LBP is a special case of EP when we make the approximation that the posterior is fully factorized: $q(x) = \prod_{i=1}^N q_i(x_i)$. To see this, consider combining this factored prior with a term $t_i(x) = P(X_i|\text{Pa}(X_i))$ to yield to posterior $p(x)$. We now seek the distribution q s.t., $D(p(x)||q(x))$ is minimized, subject to the constraint that q be fully factorized. This means

$$q_i(x_i) = p(x_i) = \sum_{x \setminus x_i} p(x)$$

i.e., the marginals must match. These are expectation constraints:

$$E_q[\delta(x_i - v)] = E_p[\delta(x_i, v)]$$

for all values v and all nodes i. This corresponds to running LBP on a factor graph with no need for the ADF step to approximate messages.

B.7.3 Variational methods

The simplest example of a variational method is the mean-field approximation, which, roughly speaking, exploits the law of large numbers to approximate large sums of random variables by their means. In particular, we essentially decouple all the nodes, and introduce a new parameter, called a variational parameter, for each node, and iteratively update these parameters so as to minimize the cross-entropy (KL distance) between the approximate and true probability distributions, i.e., we seek to minimize $D(q||p)$, where q is the approximate distribution. (EP, by contrast, (locally) minimizes $D(p||q)$.) Updating the variational parameters becomes a proxy for inference. The mean-field approximation produces a lower bound on the likelihood.

It is possible to combine variational and exact inference; this is called a structured variational approximation. See [JGJS98, Jaa01] for good general tutorials.

B.7.4 Sampling methods

Stochastic (sampling) algorithms for (static) Bayesian networks are usually based on importance sampling (likelihood weighting) or Monte Carlo Markov Chain (MCMC) (see e.g., [Nea93, GRS96, Mac98] for good introductions), which includes Gibbs sampling and simulated annealing as special cases.

Sampling algorithms have several advantages over deterministic approximation algorithms: they are easy to implement, they work on almost any kind of model (all kinds of CPDs can be mixed together, the state-space can have variable size, the model structure can change), they can convert heuristics into provably correct algorithms by using them as proposal distributions, and, in the limit of an infinite number of samples, they are guaranteed to give the exact answer.

The main disadvantage of sampling algorithms is speed: they are often significantly slower than deterministic methods, often making them unsuitable for large models and/or large data sets. However, it is possible to combine exact and stochastic inference. The basic idea is to "integrate out" some of the variables using exact inference, and apply sampling to the remaining ones; this is called Rao-Blackwellisation [CR96].

B.7.5 Other approaches

There are a variety of other approaches we have not mentioned, most of which are designed for discrete state-spaces.[8]

- Truncating small numbers: simply force small numbers in the clique potentials to zero, and then use zero-compression. If done carefully, the overall error introduced by this procedure can be bounded (and computed) [JA90].
- Structural simplifications: make smaller cliques by removing some of the edges from the triangulated graph [Kja94]; again the error can be bounded. A very similar technique is called "mini buckets" [Dec98], but has no formal guarantees.
- Bounded cutset conditioning. Instead of instantiating exponentially many values of the cutset, simple instantiate a few of them [HSC88, Dar95]. The error introduced by this method can sometimes be bounded. Alternatively, we can sample the cutsets jointly, a technique known as blocking Gibbs sampling [JKK95].

[8] The following web page contains a list of approximate inference methods c. 1996. `camis.stanford.edu/people/pradhan/approx.html`

Appendix C

Graphical models: learning

C.1 Introduction

There are many different kinds of learning. We can distinguish between the following "axes":

- Parameter learning or structure learning. For linear-Gaussian models, these are more or less the same thing (see Section 2.4.2), since 0 weights in a regression matrix correspond to absent directed edges, and 0 weights in a precision matrix correspond to absent undirected edges. For HMMs, "structure learning" usually refers to learning the structure of the transition matrix, i.e., identifying the 0s in the CPD for $P(X_t|X_{t-1})$. We consider this parameter learning with a sparseness prior, c.f., entropic learning [Bra99a]. In general, structure learning refers to learning the graph topology no matter what parameterization is used. Structure learning is often called model selection.

- Fully observed or partially observed. Partially observed refers to the case where the values of some of the nodes in some of the cases are unknown. This may be because some data is missing, or because some nodes are latent/ hidden. Learning in the partially observed case is much harder; the likelihood surface is multimodal, so one usually has to settle for a locally optimal solution, obtained using EM or gradient methods.

- Frequentist or Bayesian. A frequentist tries to learn a single best parameter/ model. In the case of parameters, this can either be the maximum likelihood (ML) or the maximum a posteriori (MAP) estimate. In the case of structure, it must be a MAP estimate, since the ML estimate would be the fully connected graph. By contrast, a Bayesian tries to learn a distribution over parameters/ models. This gives one some idea of confidence in one's estimate, and allows for predictive techniques such as Bayesian model averaging. Although more elegant, Bayesian solutions are usually more expensive to obtain.

- Directed or undirected model. It is easy to do parameter learning in the fully observed case for directed models (BNs), because the problem decomposes into a set of local problems, one per CPD; in particular, inference is not required. However, parameter learning for undirected models (MRFs), even in the fully observed case, is hard, because the normalizing term Z couples all the parameters together; in particular, inference is required. (Of course, parameter learning in the partially observed case is hard in both models.) Conversely, structure learning in the directed case is harder than in the undirected case, because one needs to worry about avoiding directed cycles, and the fact that many directed graphs may be Markov equivalent, i.e., encode the same conditional independencies.

- Static or dynamic model. Most techniques designed for learning static graphical models also apply to learning dynamic graphical models (DBNs and dynamic chain graphs), but not vice versa. In this chapter, we only talk about general techniques; we reserve discussion of DBN-specific techniques to Chapter 6.

- Offline or online. Offline learning refers to estimating the parameters/ structure given a fixed batch of data. Online learning refers to sequentially updating an estimate of the parameters/ structure as each data point arrives. (Bayesian methods are naturally suited to online learning.) Note that one can learn a static model online and a dynamic model offline; these are orthogonal issues. If the training set is huge, online learning might be more efficient than offline learning. Often one uses a compromise, and processes "mini batches", i.e., sets of training cases at a time.

- Discriminative or not. Discriminative training is very useful when the model is going to be used for classification purposes [NJ02]. In this case, it is not so important that each model be able to explain/ generate all of the data; it only matters that the "true" model gets higher likelihood than the rival models. Hence it is more important to focus on the differences in the data from each class than to focus on all of the characteristics of the data. Typically discriminative training requries that the models for each class all be trained simultaneously (because of the sum-to-one constraint), which is often intractable. Various approximate techniques have been developed. Note that discriminative training can be applied to parameter and/or structure learning.

- Active or passive. In supervised learning, active learning means choosing which inputs you would like to see output labels for, either by selecting from a pool of examples, or by asking arbitrary questions from an "oracle" (teacher). In unsupervised learning, active learning means choosing where in the sample space the training data is drawn from; usually the learner has some control over where it is in state-space, e.g., in reinforcement learning. The control case is made harder because there is usually some cost involved in moving to unexplored parts of the state space; this gives rise to the exploration-exploitation tradeoff. (The optimal Bayesian solution to this problem, for the case of discrete MDPs, is discussed in [Duf02].) In the context of causal models, active learning means choosing which "perfect interventions" [Pea00, SGS00] to perform. (A perfect intervention corresponds to setting a node to a specific value, and then cutting all incoming links to that node, to stop information flowing upwards.[1] A real-world example would be knocking out a gene.)

In this chapter, we focus on the following subset of the above topics: passive, non-discriminative, offline, static, and directed. That leaves three variables: parameters or structure, full or partial observability, and frequentist or Bayesian. For the cases that we will not focus on here, here are some pointers to relevant papers or sections of this thesis.

- Discriminative parameter learning: [Jeb01] discuss maximum entropy discrimination for the exponential family, and reverse Jensen/EM to handle latent variables; [EL01] discuss the TM algorithm for maximizing a conditional likelihood function from fully observed data; [RR01] discuss deterministic annealing applied to discriminative training of HMMs.

- Discriminative structure learning: [Bil98, Bil00] learns the interconnectivity between observed nodes in a DBN for isolated word speech recognition.

- Online parameter learning: see Sections 4.4.2 and C.4.5.

- Online structure learning: [FG97] discuss keeping a pool of candidate BN models, and updating it sequentially.

- Dynamic models: see Chapter 6.

- Undirected parameter learning (using IPF, IIS and GIS, etc.): see e.g., [JP95, Ber, Jor02].

- Undirected structure learning: see e.g., [Edw00, DGJ01].

- Active learning of BN parameters: [TK00].

[1] For example, consider the 2 node BN where smoking $\rightarrow$ yellow-fingers; if we observe yellow fingers, we may assume it is due to nicotine, and infer that the person is a smoker; but if we paint someone's fingers yellow, we are not licensed to make that inference, and hence must sever the incoming links to the yellow node, to reflect the fact that we forced yellow to true, rather than observed that it was true.

- Active learning of BN structure: [TK01, Mur01a, SJ02].

Note that the case most relevant to an autonomous life-long learning agent is also the hardest: online, active, discriminative, Bayesian structure learning of a dynamic chain-graph model in a partially observed environment. We leave this case to future work.

C.2 Known structure, full observability, frequentist

We assume that the goal of learning in this case is to find the maximum likelihood estimates (MLEs) of the parameters of each CPD, i.e., the parameter values which maximize the likelihood of the training data, which contains M cases (assumed to be independent). The log-likelihood of the training set $D = \{D_1, \ldots, D_M\}$ is a sum of terms, one for each node:

$$L = \log \prod_{m=1}^{M} \Pr(D_m|G) = \sum_{i=1}^{n} \sum_{m=1}^{M} \log P(X_i|\text{Pa}(X_i), D_m)$$

where $\text{Pa}(X_i)$ are the parents of X_i. We see that the log-likelihood scoring function decomposes into a series of terms, one per node. (It is simple to modify the above to handle tied (shared) parameters: we simply pool the data from all the nodes whose parameters are tied.)

All that remains is to specify how to estimate the parameters of each type of CPD given its local data $\{D_m(X_i, \text{Pa}(X_i))\}$. (If the CPD is in the exponential family, this set can be summarized in terms of finite (i.e., independent of M) sufficient statistics: $\sum_m D_m(X_i, \text{Pa}(X_i))$.) A huge arsenal of techniques from supervised learning can be applied at this point, since we are just estimating a function from inputs (parent values) to outputs (child probability distribution).

When we consider the case of parameter learning with partial observability, we will replace the sufficient statistics with their expectation. Hence we phrase the results below in this more general way, in preparation for EM.

C.2.1 Multinomial distributions

In the case of tabular CPDs, where we define $\theta_{ijk} \stackrel{\text{def}}{=} P(X_i = k|\text{Pa}(X_i) = j)$, the log-likelihood becomes

$$\begin{aligned} L &= \sum_i \sum_m \log \prod_{j,k} \theta_{ijk}^{I_{ijkm}} \\ &= \sum_i \sum_m \sum_{j,k} I_{ijkm} \log \theta_{ijk} \\ &= \sum_{ijk} N_{ijk} \log \theta_{ijk} \end{aligned} \tag{C.1}$$

where $I_{ijkm} \stackrel{\text{def}}{=} I(X_i = k, \text{Pa}(X_i) = j|D_m)$ is 1 if the event $(X_i = k, \text{Pa}(X_i) = j)$ occurs in case D_m, and hence $N_{ijk} \stackrel{\text{def}}{=} \sum_m I(X_i = k, \text{Pa}(X_i) = j|D_m)$ is the number of times the event $(X_i = k, \text{Pa}(X_i) = j)$ was seen in the training set. (For a Markov chain, this corresponds to counting transitions.) It is easy to show (taking derivatives and using a Lagrange multiplier to ensure $\sum_k \theta_{ijk} = 1$ for all i, j) that the MLE is

$$\hat{\theta}_{ijk} = \frac{N_{ijk}}{\sum_{k'} N_{ijk'}} \tag{C.2}$$

C.2.2 Conditional linear Gaussian distributions

Now consider a conditional linear Gaussian CPD for node Y with continuous parent X and discrete parent Q, i.e.,

$$p(y|x, Q = i) = c|\Sigma_i|^{-\frac{1}{2}} \exp\left(-\tfrac{1}{2}(y - B_i x)'\Sigma_i^{-1}(y - B_i x)\right)$$

where $c = (2\pi)^{-d/2}$ is a constant and $|y| = d$. The j'th row of B_i is the regression vector for the j'th component of y given that $Q = i$. To allow for an offset, we shall assume the last component of x is fixed to 1, and the last column of B_i is μ_i.

Special cases of this CPD occur in the following common models:

- Mixture of Gaussians. X does not exist, Q is hidden, and Y is observed. (The temporal version of this is an HMM with Gaussian outputs: $Q = Q_t$ is the discrete hidden state, and $Y = Y_t$ is the Gaussian observation.)

- Factor analysis. Q does not exist, Σ is assumed diagonal, X is hidden and Y is observed. (The temporal version of this is the Kalman filter model; for the observation CPD, $X = X_t$ and $Y = Y_t$; for the transition CPD, $X = X_{t-1}$ and $Y = X_t$, so X and Y will be hidden.)

- Mixture of factor analyzers [GH96a]. Σ_i is diagonal, Q and X are hidden, Y is observed. (The temporal version of this is a switching Kalman filter; for the observation CPD, $X = X_t$, $Q = Q_t$ and $Y = Y_t$; for the transition CPD, $X = X_{t-1}$, $Q = Q_t$ and $Y = X_t$, so the whole family will be hidden.)

Since we are interested in estimating time-invariant dynamic models (such as Kalman filters models whose parameters are tied across all time slices), we express all the estimates in terms of expected sufficient statistics, whose size is independent of the number of samples (time steps). This is different from the usual presentation, which states the formulas in terms of the raw data matrix.

The log-likelihood is

$$\log \prod_{m=1}^{M} \prod_{i=1}^{K} [\Pr(y_m | x_m, Q_m = i, D_m)]^{q_m^i}$$

where the indicator variable $q_m^i = 1$ if Q has value i in the m'th data case, and 0 otherwise. Since Q, X and Y may all be unobserved, we compute the expected complete-data log likelihood as follows

$$L = -\tfrac{1}{2} \sum_m E\left[\sum_i q_m^i \log |\Sigma_i| + q_m^i (y_m - B_i x_m)' \Sigma_i^{-1} (y_m - B_i x_m) \mid D_m \right]$$

We note that

$$E[q_m^i x_m x_m' | D_m] = E[q_m^i | D_m] E[x_m x_m' | Q_m = i, D_m] \stackrel{\text{def}}{=} w_m^i \langle x_m x_m' \rangle_i$$

where the posterior probabilities $w_m^i = \Pr(Q = i | D_m)$ are weights, and $\langle x_m x_m' \rangle_i$ is a conditional second moment.

Since a full covariance matrix has $\frac{d(d+1)}{2}$ parameters, we are often interested in placing restrictions on the form of this matrix. We consider the following kinds of restrictions.

- Σ_i is tied across all states, i.e., $\Sigma_i = \Sigma$ for all i.

- Σ_i is diagonal.

- Σ_i is spherical (isotropic), i.e., $\Sigma_i = \sigma_i^2 I$.

It turns out that the ML estimate of a diagonal covariance matrix is the same as the ML estimate of the full matrix, with the off-diagonal entries set to 0; hence we will not consider this case further. This leaves us with four combinations: full or spherical, tied or not. In addition, it is useful to consider the special case where there is no parent except the constant $X = 1$, so $B_i = \mu_i$; this gives 8 formulas. These are summarized in Table C.1. (These formulae are derived in Section C.11.) Although they look hairy, we would like to emphasize the generality of these results: it subsumes the derivation of the M step for a large class of popular models, including HMMs with (mixtures of) Gaussian outputs, (mixtures of) factor analysis models, (switching) Kalman filter models, etc.

We can see from the table that we need the following expected sufficient statistics:

- $\sum_m w_m^i$, $\sum_m w_m^i \langle y_m y_m' \rangle_i$

- If X exists: $\sum_m w_m^i \langle x_m y_m' \rangle_i$, $\sum_m w_m^i \langle x_m x_m' \rangle_i$.

Eqn	Cov.	B/μ	Tied	Formula
1	Full	B	U	$\Sigma_i = \frac{\sum_m w_m^i \langle y_m y_m' \rangle_i}{\sum_m w_m^i} - \frac{B_i \sum_m w_m^i \langle x_m y_m' \rangle_i}{\sum_m w_m^i}$
2	Full	B	T	$\Sigma = \frac{1}{M} \left(\sum_i \sum_m w_m^i \langle y_m y_m' \rangle_i \right) - \frac{1}{M} \left(\sum_i B_i \sum_m w_m^i \langle x_m y_m' \rangle_i \right)$
3	Full	μ	U	$\Sigma_i = \frac{\sum_m w_m^i \langle y_m y_m' \rangle_i}{\sum_m w_m^i} - \mu_i \mu_i'$
4	Full	μ	T	$\Sigma = \frac{1}{M} \left(\sum_i \sum_m w_m^i \langle y_m y_m' \rangle_i \right) - \frac{1}{M} \left(\sum_i (\sum_m w_m^i) \mu_i \mu_i' \right)$
5	Sph.	B	U	$\sigma_i^2 = \frac{1/d}{\sum_m w_m^i} \mathrm{Tr} \left[\sum_m w_m^i \langle y_m y_m' \rangle_i + \sum_m w_m^i B_i' B_i \langle x_m x_m' \rangle_i \right.$
				$\left. -2 \sum_m w_m^i B_i \langle x_m y_m' \rangle_i \right]$
6	Sph.	B	T	$\sigma^2 = \frac{1}{Md} \mathrm{Tr} \left[\sum_i (\sum_m w_m^i \langle y_m y_m' \rangle_i) + \sum_i (\sum_m w_m^i B_i' B_i \langle x_m x_m' \rangle_i) \right.$
				$\left. -2 \sum_i (\sum_m w_m^i B_i \langle x_m y_m' \rangle_i) \right]$
7	Sph.	μ	U	$\sigma_i^2 = \frac{1}{d} \left(\frac{\sum_m w_m^i \langle y_m' y_m \rangle_i}{\sum_m w_m^i} - \mu_i' \mu_i \right)$
8	Sph.	μ	T	$\sigma^2 = \frac{1}{Md} \left(\sum_m w_m^i \langle y_m' y_m \rangle_i - (\sum_m w_m^i) \mu_i' \mu_i \right)$
9	-	B	-	$B_i = \left(\sum_m w_m^i \langle y_m x_m' \rangle_i \right) \left(\sum_m w_m^i \langle x_m x_m' \rangle_i \right)^{-1}$
10	-	μ	-	$\mu_i = \frac{\sum_m w_m^i \langle y_m \rangle_i}{\sum_m w_m^i}$

Table C.1: Parameter estimation for a CLG CPD $P(Y|X, Q = i) = \mathcal{N}(Y; B_i X, \Sigma_i)$. The covariance matrix can be full or spherical (sph.), tied (T) or untied (U), and the regression matrix can be a full matrix (B), or just a mean column vector (μ) (if $X = 1$). M is the number of training cases, d is the dimensionality of Y.

- If X does not exist: $\sum_m w_m^i \langle y_m \rangle_i$.
- If X exists and $\Sigma_i = \sigma_i^2 I$: $\sum_m w_m^i (B_i' B_i) \langle x_m x_m' \rangle_i$, $\sum_m w_m^i B_i i \langle x_m y_m' \rangle_i$.

Note that, if, as is commonly the case, Y is observed, we can make the following simplifications: $\langle x_m y_m' \rangle_i \to \langle x_m \rangle_i y_m'$, $\langle y_m y_m' \rangle_i \to y_m y_m'$, and $\langle y_m' y_m \rangle_i \to y_m' y_m$.

C.2.3 Other CPDs

Noisy-OR One way to estimate these is to represent the CPDs as mini Bayes nets, as in Figure A.8. The deterministic link from U_i to Y is of course fixed; the link from X_i to U_i is just a binomial (binary multinomial) distribution, representing the "reliability" of the link; its parameters can be estimated using EM [MH97] (we need EM since the U_i are hidden).

Trees MLEs of trees can be computed using the standard greedy tree-growing methods such as CART and C4.5.

Softmax Parameter estimating for softmax CPDs can be done using the iteratively reweighted least squares (IRLS) algorithm (see e.g., [JJ94b, App.A]), or gradient descent.

Feedforward neural networks Multilayer perceptrons can be fit using gradient descent (often called "back-propagation" in this context; see e.g., [Bis95]).

C.3 Known structure, full observability, Bayesian

If there are a small number of training cases compared to the number of parameters, we should use a prior to regularize the problem. In this case, we call the estimates maximum *a posterori* (MAP) estimates, as opposed to maximum likelihood estimates. Since the model is fully observable, the posterior over parameters will be unimodal. If we use a conjugate prior, we can compute this posterior in closed form.

C.3.1 Multinomial distributions

For discrete nodes, it is very common to assume the local CPDs are multinomial, i.e., represented as a table of the form $\Pr(X_i = k|\Pi_i = j) = \theta_{ijk}$, for $k = 1, \ldots, r_i$ and $j = 1, \ldots, q_i$, where r_i is the number of values node i can take on, and $q_i = \prod_{l \in \Pi_i} r_l$ is the number of values node i's parents, Π_i, can take on. These parameters satisfy the constraints $0 \leq \theta_{ijk} \leq 1$ and $\sum_k \theta_{ijk} = 1$.

Following common practice, we will make two assumptions. First, global parameter independence: $P(\theta) = \prod_{i=1}^n P(\theta_i)$, where $\theta_i = \{\theta_{ijk}, j = 1, \ldots, q_i, k = 1, \ldots, r_i\}$ are the parameters for node i. Second, local parameter independence: $P(\theta_i) = \prod_{j=1}^{q_i} \theta_{ij}$, where $\theta_{ij} = \{\theta_{ijk}, k = 1, \ldots, r_i\}$ are the parameters for the j'th row of X_i's table (i.e., parameters for the j'th instantiation of X_i's parents). Given a factored prior and complete data, the posterior over parameters will also be factored [SL90]. We will give the form of this posterior below. (Note that, if we have missing data, the parameter posterior will no longer be factored. Hence assuming parameter independence is equivalent to assuming that one's prior knowledge was derived from a fully observed "virtual database".)

[GH97, RG01] prove that the assumptions of global and local parameter independence, plus an additional assumption called likelihood equivalence[2], imply that the prior must be Dirichlet. Fortunately, the Dirichlet prior is the conjugate prior for the multinomial [Ber85], which makes analysis easier, as we will see below. (For this reason, the Dirichlet is often used even if the assumption of likelihood equivalence is violated.) Note that, in the case of binary nodes, the multinomial becomes the Bernoulli distribution, and the Dirichlet becomes the Beta.

Given global and local independence, each CPD $P(X_i|U_i = j) = \theta_{ij}$ is a multinomial random variable with r_i possible values. The Dirichlet prior, $\theta_{ij} \sim \mathcal{D}(\alpha_{ij1}, \ldots, \alpha_{i,j,r_i})$, is defined as

$$P(\theta_{ij}|\alpha_{ij}) = \prod_{k=1}^{r_i} \theta_{ijk}^{\alpha_{ijk}-1} \times \frac{1}{B(\alpha_{ij1}, \ldots, \alpha_{i,j,r_i})}$$

The normalizing constant is the r_i-dimensional Beta function

$$B(\alpha_1, \ldots, \alpha_r) = \frac{\Gamma(\sum_k \alpha_k)}{\prod_{k=1}^r \Gamma(\alpha_k)}$$

where $\Gamma(\cdot)$ is the gamma function; for positive integers, $\Gamma(n) = (n-1)!$.

The hyperparameters, $\alpha_{ijk} > 0$, have a simple interpretation as pseudo counts. The quantity $\alpha_{ijk} - 1$ represents the number of imaginary cases in which event $(X_i = k, \Pi_i = j)$ has already occured (in some virtual prior database). Upon seeing a database D in which the event $(X_i = k, \Pi_i = j)$ occurs N_{ijk} times, the parameter posterior becomes

$$\theta_{ij}|D \sim \mathcal{D}(\alpha_{ij1} + N_{ij1}, \ldots, \alpha_{i,j,r_i} + N_{i,j,r_i})$$

The posterior mean is

$$E[\theta_{ijk}|D] = \frac{\alpha_{ijk} + N_{ijk}}{\sum_{l=1}^{r_i} \alpha_{ijl} + N_{ijl}} \tag{C.3}$$

and the posterior mode (MAP estimate) is

$$\arg\max P[\theta_{ijk}|D] = \frac{\alpha_{ijk} + N_{ijk} - 1}{\sum_{l=1}^{r_i} \alpha_{ijl} + N_{ijl} - r_i} \tag{C.4}$$

[2] Two graph structures are likelihood equivalent if they assign the same marginal likelihood to data, i.e., $P(D|G_1) = P(D|G_2)$. This is weaker than the assumption of Markov (hypothesis) equivalence, which says two graphs are equivalent if they encode the same set of conditional independence assumptions. Hypothesis equivalence is clearly violated if we adopt a causal interpretation of BNs. In addition, likelihood equivalence is violated if we have interventional data. See [Hec95] for a discussion.

and the predictive distribution is

$$\begin{aligned} P(x|\theta) &= \prod_{i=1}^{n}\prod_{j=1}^{q_i}\int\prod_{k=1}^{r_i}\theta_{ijk}^{1_{ijk}(x)}P(\theta_{ijk})d\theta_{ijk} \\ &= \prod_{i=1}^{n}\prod_{j=1}^{q_i}\prod_{k=1}^{r_i}E(\theta_{ijk})^{1_{ijk}(x)} \end{aligned} \tag{C.5}$$

where $1_{ijk}(x)$ is an indicator function that is 1 if the event $(X_i = k, \Pi_i = j)$ occurs in case x, and is 0 otherwise.

To compute the marginal likelihood for a database of M cases, $D = (x^1, \ldots, x^M)$, we can use sequential Bayesian updating [SL90]:

$$\begin{aligned} P(D|G) &= P(x^1|\theta_0)P(x^2|\theta_0, x^1)\ldots P(x^M|\theta_0, x^{1:M-1}) \\ &= P(x^1|\theta_0)P(x^2|\theta_1)\ldots P(x^M|\theta_{M-1}) \end{aligned}$$

where $\theta_0 = \alpha$ is our prior, and θ_t is the result of updating θ_{t-1} with x^t. ([HGC95] call $P(D|G)P(G)$ the Bayesian Dirichlet (BD) score for a model.)

Assessing Dirichlet priors

It is clearly impossible for the user to specify parametric priors for $O(2^{n^2})$ graph structures. [HGC95] show that, under some assumptions[3], it is possible to derive the Dirichlet parameters for an arbitrary graph from a single prior network, plus a confidence factor, α. Specifically, $\alpha_{ijk} = \alpha P(X_i = k, \Pi_i = j|G_c)$, where G_c is the complete graph. When priors are derived in this manner, the BD score is called BDe (BD with likelihood equivalence). Unfortunately, it might be difficult to parameterize the prior network (especially because of the counterintuitive conditioning on G_c: see [HGC95] for a discussion). In addition, computing the parameter priors for an arbitrary graph structure from such a prior network requires running an inference algorithm, which can be slow. [SDLC93] suggest a similar way of computing Dirichlet priors from a prior network.

A much simpler alternative is to use a non-informative prior. A natural choice is $\alpha_{ijk} = 0$, which corresponds to maximum likelihood. (In the binary case, this is called Haldane's prior.) However, this is an improper prior. More importantly, this will cause the log-likelihood to explode if a future case contains an event that was not seen in the training data.

If we set $0 < \alpha_{ijk} < 1$, we encourage the parameter values θ_{ijk} to be near 0 or 1, thus encoding near-deterministic distributions. This might be desirable in some domains. [Bra99a] explicitely encodes this bias using an "entropic prior" of the form

$$P(\theta_{ij}) \propto e^{-H(\theta_{ij})} = \prod_k \theta_{ijk}^{\theta_{ijk}}.$$

Unfortunately, the entropic prior is not a conjugate prior, which means we must use iterative techniques to find the MAP estimate.

[CH92] suggest the uniform prior, $\alpha_{ijk} = 1$. This is a non-informative prior since it does not affect the posterior (although it does affect the marginal likelihood). Unfortunately, it is not entirely uninformative, because it is not transformation invariant. The fully non-informative prior is called a Jeffrey's prior. For the special case of a binary root node (i.e., a node with no parents and a beta distribution), the Jeffrey's prior is just $\alpha_{ijk} = \frac{1}{2}$. Computing a Jeffrey's prior for an arbitrary BN is hard: see [KMS$^+$98].

[Bun91] suggests the prior $\alpha_{ijk} = \alpha/(r_i q_i)$, where α is an equivalent sample size. This induces the following distribution

$$P(X_i = k|\Pi_i = j) = E[\theta_{ijk}] = \frac{\alpha/(r_i q_i)}{\alpha/q_i} = \frac{1}{r_i}$$

[3] The assumptions are global and local parameter independence, likelihood equivalence, parameter modularity and structural possibility. Parameter modularity says that $P(\theta_{ij})$ is the same for any two graph structures in which X_i has the same set of parents. Structural possibility says that all complete (fully connected) graph structures are possible a priori.

This is a special case of the BDe metric where the prior network assigns a uniform distribution to the joint distribution; hence [HGC95] call this the BDeu metric. (Not surprisingly, experiments reported in [HGC95] suggest that, for small databases, it is better to use the uninformative prior (BDeu metric) that an informative, but incorrect, prior (BDe metric).)

A more sophisticated approach is to use a hierarchical prior, where we place a prior on the hyperparameters themselves. For example, let $\alpha_{ijk} = \alpha_{ij0}/r_i$ for each i, j, where α_{ij0} is the prior precision for α_{ij}. α_{ij0} is itself an r.v., and can be given e.g., a gamma distribution. Unfortunately, we can no longer compute the marginal likelihood in closed form using such hierarchical priors, and must resort to sampling, as in [GGT00, DF99]. A more efficient method, known as empirical Bayes or maximum likelihood type II, is to estimate the hyperparameters from data: see [Min00b] for details.

C.3.2 Gaussian distributions

If $P(Y)$ is a Gaussian, then the ML estimates of μ and Σ are the sample mean and covariance of the observations:

$$\hat{\Sigma}^{ML} = E[YY'] = \frac{1}{M}\sum_m y_m y_m'$$

A suitable prior is the Normal-Wishart [DeG70, Min00c]. In the special case of a zero-mean, diagonal Wishart, this amounts to adding λ_i to the diagonal elements of the empirical covariance matrix:

$$\hat{\Sigma}_{MAP} = \frac{1}{M}\left(M\hat{\Sigma}^{ML} + \Lambda\right)$$

where $\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_d)$.

C.3.3 Conditional linear Gaussian distributions

See [Min00a] for an up-to-date treatment of multivariate Bayesian linear regression, which uses the matrix-normal distribution.

C.3.4 Other CPDs

Noisy-OR For noisy-OR, we can simply put an independent Dirichlet prior on the suppression probabilities of each parent.

Trees Bayesian approaches to decision trees are discussed in [Bun89].

Softmax There is no conjugate prior for the softmax function, so one must resort to approximations, e.g., [JJ00].

Feedforward neural networks Bayesian estimation of the parameters of MLPs is discussed in e.g., [Bis95].

C.4 Known structure, partial observability, frequentist

In the partially observable case, the log-likelihood is

$$\begin{aligned} L &= \sum_m \log P(D_m) \\ &= \sum_m \log \sum_h P(H = h, V = D_m) \qquad \text{(C.6)} \end{aligned}$$

where the innermost sum is over all the assignments to the hidden nodes H, and $V = D_m$ means the visible nodes take on the values specificied by case D_m. Unlike the fully observed case, this does *not* decompose into a sum of local terms, one per node (although it does decompose across training cases). We will see two different solutions to this below, based on gradient ascent and EM, which we will then compare experimentally in terms of speed.

C.4.1 Gradient ascent

The obvious way to maximize likelihood is to do gradient ascent. Following [BKRK97], we show below that the gradient is a sum of family marginals, which can be computed using an inference engine (since a node and its parents will always be in the same clique due to moralization). This turns out to be very similar to EM (see Section C.4.2). (This similarity was also noted in [BC94] for the case of gradient ascent HMM training.) Consider first the case of tabular CPDs. Let $w_{ijk} = P(X_i = j|\text{Pa}(X_i) = k)$ (so $\sum_j w_{ijk} = 1$).

$$\begin{aligned}\frac{\partial L}{\partial w_{ijk}} &= \sum_{m=1}^{M} \frac{\partial \log P_w(D_m)}{\partial w_{ijk}} \\ &= \sum_{m=1}^{M} \frac{\partial P_w(D_m)/\partial w_{ijk}}{P_w(D_m)}\end{aligned}$$

which at least decomposes into a sum of terms, one per data case (because we assumed they were independent). The key trick is that we can simplify the numerator of the term inside the sum by conditioning on the parents:

$$\begin{aligned}\frac{\partial P_w(D_m)}{\partial w_{ijk}} &= \frac{\partial}{\partial w_{ijk}} \sum_{j',k'} P_w(D_m|X_i = j', \text{Pa}(X_i) = k') P_w(X_i = j'|\text{Pa}(X_i) = k') P_w(\text{Pa}(X_i) = k) \\ &= P_w(D_m|X_i = j, \text{Pa}(X_i) = k) P_w(\text{Pa}(X_i) = k)\end{aligned}$$

since the derivative "plucks out" the term involving $j = j'$ and $k = k'$, and since $P_w(X_i = j|\text{Pa}(X_i) = k) = w_{ijk}$. Hence

$$\begin{aligned}\frac{\partial P_w(D_m)/\partial w_{ijk}}{P_w(D_m)} &= \frac{P_w(D_m|X_i = j, \text{Pa}(X_i) = k) P_w(\text{Pa}(X_i) = k)}{P_w(D_m)} \\ &= \frac{P_w(X_i = j, \text{Pa}(X_i) = k|D_m) P_w(D_m) P_w(\text{Pa}(X_i) = k)}{P_w(X_i = j, \text{Pa}(X_i) = k) P_w(D_m)} \\ &= \frac{P_w(X_i = j, \text{Pa}(X_i) = k|D_m)}{w_{ijk}}\end{aligned}$$

where the second line follow from Bayes' rule. The numerator, $P_w(X_i = j, \text{Pa}(X_i) = k|D_m)$, can be computed using inference with the evidence set to D_m.

The extension of the above to non-tabular CPDs follows by using the chain rule:

$$\frac{\partial \log P_w(D)}{\partial \lambda_p} = \sum_{i,j,k} \frac{\partial \log P_w(D)}{\partial w_{ijk}} \times \frac{\partial w_{ijk}}{\partial \lambda_p}$$

where λ_p are the parameters of the CPD.

Handling constraints on parameters

Generally we have constraints on the parameters. For example, for an HMM transition matrix, we require $0 \leq A(i,j) \leq 1$ for all i, j and $\sum_j A(i,j) = 1$ for all i. Hence we must work inside the unit cube to ensure $0 \leq A(i,j) \leq 1$ and project onto the constraint surface to ensure $\sum_j A(i,j)$. It is possible to reparameterize this problem using a softmax transform, $\tilde{A}(i,j) = \frac{\exp(A(i,j))}{\exp(\sum_k A(i,k))}$, to eliminate both of these requirements. However, reparameterization is generally not recommended for constrained optimization problems, because it changes the nature of the surface [GMW81].

Sometimes the constraints are more complicated. For example, when estimating Gaussian distributions, we must ensure the covariance matrix is positive semi-definite. In this case, we can reparameterize and do unconstrained optimization of $\Sigma^{\frac{1}{2}}$.

C.4.2 EM algorithm

The basic idea behind EM is to apply Jensen's inequality [CT91] to Equation C.6 to get a lower bound on the log-likelihood, and then to iteratively maximize this lower bound. Jensen's inequality says that, for any concave function f,

$$f\left(\sum_j \lambda_j y_j\right) \geq \sum_j \lambda_j f(y_j)$$

where $\sum_j \lambda_j = 1$. Informally, this says that f of the average is bigger than the average of the f's, which is easily seen by drawing f. Since the log function is concave, we can apply Jensen's to Equation C.6 to get

$$\begin{aligned}
L &= \sum_m \log \sum_h P_\theta(H = h, V_m) \\
&= \sum_m \log \sum_h q(h|V_m) \frac{P_\theta(H = h, V_m)}{q(h|V_m)} \\
&\geq \sum_m \sum_h q(h|V_m) \log \frac{P_\theta(H = h, V_m)}{q(h|V_m)} \\
&= \sum_m \sum_h q(h|V_m) \log P_\theta(H = h, V_m) - \sum_m \sum_h q(h|V_m) \log q(h|V_m)
\end{aligned}$$

where q is a function s.t. $\sum_h q(h|V_m) = 1$ and $0 \leq q(h|V_m) \leq 1$, but is otherwise arbitrary.[4] Maximizing the lower bound with respect to q gives

$$q(h|V_m) = P_\theta(h|V_m)$$

This is called the E (expectation) step, and makes the bound tight. Approximate inference increases this bound, but may not maximize it [NH98].

Maximizing the lower bound with respect to the free parameters θ' is equivalent to maximizing the expected complete-data log-likelihood

$$\langle l_c(\theta') \rangle_q = \sum_m \sum_h q(h|V_m) \log P_{\theta'}(H = h, V_m)$$

This is called the M (maximization) step. This is efficient if the corresponding complete-data problem is tractable, and q has a tractable form.

If we use $q(h|V_m) = P_\theta(h|V_m)$, as in exact EM, then the above is often written as

$$Q(\theta'|\theta) = \sum_m \sum_h P(h|V_m, \theta) \log P(h, V_m|\theta')$$

Dempster et al. [DLR77] proved that choosing θ' s.t. $Q(\theta'|\theta) > Q(\theta|\theta)$ is guaranteed to ensure $P(D|\theta') > P(D|\theta)$, i.e., increasing the expected complete-data log-likelihood will increase the actual (partial) data log-likeliood. This is because using $q(h|V_m) = P_\theta(h|V_m)$ in the E step makes the lower bound touch the actual log-likelihood curve, so raising the lower bound at this point will also raise the actual log-likelihood curve. If we do an approximate E step, we do not have such a guarantee.

In the case of multinomial CPDs, the expected complete-data log-likelihood becomes (based on Equation C.1)

$$Q(\theta'|\theta) = \sum_{ijk} E[N_{ijk}] \log \theta'_{ijk}$$

where $EN_{ijk} = \sum_m P(X_i = k, \text{Pa}(X_i) = j|D_m, \theta)$, so the M-step, $\theta := \arg\max_{\theta'} Q(\theta'|\theta)$, becomes (based on Equation C.2)

$$\hat{\theta}_{ijk} = \frac{EN_{ijk}}{\sum_{k'} EN_{ijk'}}$$

[4] In physics, $-L(q, \theta)$ is called the variational free energy, and consists of the expected energy, $\langle -\log P_\theta(H = h, V_m) \rangle_q$, and the entropy, $H(q)$. Loopy belief propagation (see Section B.7.1) approximates $H(q)$ by considering only single and pairwise terms, resulting in the Bethe approximation to the free energy.

This is a generalization of the EM algorithm for HMMs (often called Baum-Welch), which was already described in Section 1.2.3. This idea can be applied to any Bayes net [Lau95]: compute the expected sufficient statistics (ESS), $\sum_m P(X_i, \text{Pa}(X_i)|D_m, \theta_{\text{old}})$, using an inference algorithm, and use these in the M step (Section C.2) as if they were actually sufficient statistics computed from the data, and then repeat.

Much has been written about EM. Please see [MvD97, MK97, UN98, NH98, Mar01] for details. For articles specifically about EM for Bayes nets, see [Lau95, BKS97, Zha96].

C.4.3 EM vs gradient methods

The title of this section is a little misleading, since EM is implicitly a gradient method, as we show below. However, it is still an interesting question whether one should explicitely try to minimize the gradient, or just use EM.[5]

We shall restrict our attention to deterministic algorithms with the following form of additive update rule[6]:

$$\Theta^{(t+1)} = \Theta^{(t)} + \lambda_t \mathbf{d}_t, \tag{C.7}$$

where $\mathbf{d}_t$ is the direction in which to move at iteration t, and λ_t is the step size. Even within the confines of this form, many variations are possible. For example, how do we choose the direction? How do we choose the step size? How do we maintain the parameter constraints? Are λ_t and $\mathbf{d}_t$ just functions of the t'th training case (i.e., an online algorithm), or can they depend on all the training data (i.e., a batch algorithm)? We discuss these and other issues below.

The performance of the algorithms can be measured in two ways: the quality of the local optimum which is reached, and the speed with which it is reached. Clearly, both answers will depend on the topology of the space, the starting point, and the direction of the walk. The topology of the space may depend on the network structure, the amount of missing data, and the number of training cases. In our experimental setup, we vary all three of these factors to see how robust our conclusions are. For any fixed space, we start all algorithms off at the same point, and use the same stopping criterion. An algorithm which converges faster is always better, since, in any fixed amount of time, we can afford to try restarting from many different points; the final "answer" can then be the best point visited, or some combination of all of them. We vary the starting point to test the robustness of our conclusions.

The direction

The most obvious choice for the direction vector is the gradient of the log-likelihood function

$$\mathbf{g}_t = \left(\frac{\partial l}{\partial \theta_1}, \cdots, \frac{\partial l}{\partial \theta_n}\right)|_{\Theta=\Theta^{(t)}}.$$

As we saw in Section C.4.1, for tabular CPDs with parameters $\theta_{ijk} \stackrel{\text{def}}{=} P(X_i = k|\text{Pa}(X_i) = j) = w_{ikj}$, this is given by

$$\frac{\partial \log \Pr(V|\Theta)}{\partial \theta_{ijk}} = \sum_{m=1}^{M} \frac{\Pr(X_i = k, \Pi_i = j|V_m)}{\theta_{ijk}} \tag{C.8}$$

Another choice is the generalized gradient $\tilde{\mathbf{g}}_t = \Pi(\Theta^{(t)})\mathbf{g}_t$, where Π is some negative definite projection matrix. It can be shown [JJ93] that this is essentially what EM is doing, where $\Pi(\Theta^{(t)}) \approx -\ddot{Q}(\hat{\Theta}, \hat{\Theta})^{-1}$, and

$$[\ddot{Q}(\hat{\Theta}, \hat{\Theta})]_{i,j} = \frac{\partial^2 Q(\Theta', \hat{\Theta})}{\partial \theta'_i \partial \theta'_j}|_{\Theta'=\hat{\Theta}}$$

[5] This subsection is based on my class project for Stats 200B in 1997.

[6] Multiplicative update rules (exponentiated gradient methods) are also possible, but [BKS97] has shown them to be inefficient in this context.

is the Hessian of Q evaluated at $\hat{\Theta}$, some interior point of the space, and Q is the expected complete-data log-likelihood

$$\begin{aligned} Q(\Theta'|\Theta) &= \sum_h \Pr(h|V,\Theta)\log\Pr(V,h|\Theta') \\ &= E_h\left[\log\Pr(V,H|\Theta')|V,\Theta\right] \end{aligned}$$

with H being the hidden variables and V the visibles. Thus EM is a quasi-Newton (variable metric) optimization method.

In [XJ96] and [JX95] they give exact formulas for Π for a mixture of Gaussians model and a mixture of experts model, but in general Π will be unknown. However, we can still compute the generalized gradient as follows

$$\tilde{\mathbf{g}}_t = U(\Theta^{(t)}) - \Theta^{(t)} \tag{C.9}$$

where the EM update operator is

$$\Theta^{(t+1)} = U(\Theta^{(t)}) = \arg\max_{\Theta} Q(\Theta|\Theta^{(t)}).$$

Conjugate directions

It is well known that making the search directions conjugate can dramatically speed up gradient descent algorithms. The Fletcher-Reeves-Polak-Ribiere formula [PVTF88] is

$$\begin{aligned} \mathbf{d}_0 &= \mathbf{g}_0 \\ \mathbf{d}_{t+1} &= \mathbf{g}_{t+1} + \gamma_t \mathbf{d}_t \\ \gamma_t &= \frac{\mathbf{g}_{t+1}^T \mathbf{g}_{t+1}}{\mathbf{g}_t^T \mathbf{g}_t} \end{aligned}$$

It is also possible to compute conjugate generalized gradients. Jamshidian and Jennrich [JJ93] propose the following update rule:

$$\begin{aligned} \mathbf{d}_0 &= \tilde{\mathbf{g}}_0 \\ \mathbf{d}_{t+1} &= \tilde{\mathbf{g}}_{t+1} + \gamma_t \mathbf{d}_t \\ \gamma_t &= \frac{\tilde{\mathbf{g}}_{t+1}^T (\mathbf{g}_t - \mathbf{g}_{t+1})}{\mathbf{d}_t^T (\mathbf{g}_{t+1} - \mathbf{g}_t)} \end{aligned}$$

They show that this method is faster than EM for a variety of non-belief net problems. Thiesson [Thi95] has applied this method to belief nets, but does not report any experimental results.

The step size

By substituting equation C.9 into equation C.7, we can rewrite the EM update rule as

$$\Theta^{(t+1)} = \Theta^{(t)} + \lambda_t U(\Theta^{(t)}) - \lambda_t \Theta^{(t)} = (1-\lambda_t)\Theta^{(t)} + \lambda_t U(\Theta^{(t)}). \tag{C.10}$$

We shall call this the EM(λ) rule. (In [BKS97], they derive this rule from an on-line learning perspective.)

The line minimization method suggests choosing a step size of $\lambda_t = \arg\max_\lambda l(\Theta^{(t)} + \lambda \mathbf{d}_t)$. Since this can be quite slow, a simplification is to use a constant step size $\lambda_t = \lambda$. For $\lambda = 1$, this corresponds to the standard EM rule, and is guaranteed to converge to a local maximum, and to satisfy the positivity and summation constraints. Bauer et al. [BKS97] show that sometimes the optimal learning rate is given by $\lambda > 1$; however, if $\lambda > 1$, the algorithm is only guaranteed to converge to a local maxmimum if it starts close enough to that maximum; and if $\lambda > 2$, there are no convergence guarantees at all. In the experiments of [JX95] on mixtures of experts, they found that using $\lambda > 1$ often led to divergence, whereas in the experiments of [BKS97], on relatively large belief nets (the Alarm network and the Insurance network), they found that using $\lambda = 1.8$ always led convergence, presumably because of the greater number of local maxima. This rule is also studied in [RW84] and elsewhere.

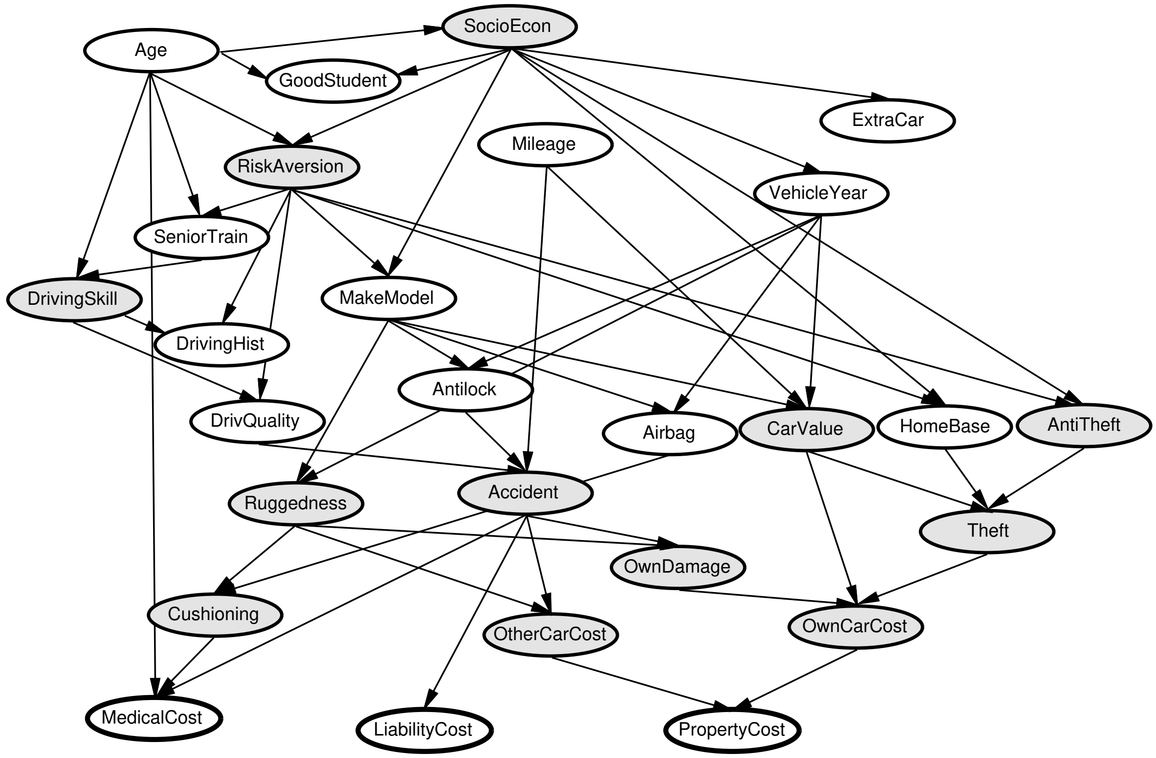


Figure C.1: A simple model of the car insurance domain, from [RN95]. Shaded nodes are hidden.

Experimental comparison

We generated 25 training cases from a small Bayes net (Figure C.1) with random tabular CPDs; we then randomly hid 20% and 50% of the nodes, and timed how long it took each algorithm to find a MLE.[7] Some results are shown in Tables C.2 and C.2. It can be seen that all methods reach similar quality local minima, but that conjugate generalized gradient without linesearch is the fastest, and non conjugate generalized gradient without linesearch (which is equivalent to EM) is the second fastest. Conjugate methods are always faster, but, perhaps surprisingly, linesearching never seems to pay off, at least in this experiment.

Note that a direct implementation of EM (i.e., one which does not incur the overhead of explicitly computing the generalized gradient) is about $1.6\times$ faster than the results shown in the tables, which eliminates much of the performance advantage of the conjugate method. ([JJ93, Thi95] suggest starting with EM, and then only switching over to a conjugate gradient method when near a local optimum.)

On balance, it seems that EM has several advantages over gradient ascent.

- There is no step-size parameter.
- It automatically takes care of parameter constraints.
- It can handle (conjugate) priors easily.
- It can handle deterministic constraints (e.g., $w_{ijk} = 0$).
- It is simple to implement.

[7] The gradient descent code is closely based on the implementation given in [PVTF88]. Constraints were ensured by reparameterizing as follows: $\theta_{ijk} = \frac{\beta_{ijk}2}{\sum_{k'} \beta_{ijk'}}$; gradients were computed wrt β_{ijk} using the chain rule. The stopping criterion was when the percentage change in the negative log likelihood dropped below 0.1%. The belief net inference package was written by Geoff Zweig in C++. All experiments were performed on a Sun Ultra Sparc, where 1 CPU second corresponds to roughly 1 real (wall clock) second (this was 1997!).

Linemin.?	Gen. grad.?	Conj.?	final NLL	#iter	CPU/s	#fn.
$\times$	$\times$	$\times$	300.0	105	74.5	210
$\times$	$\times$	$\surd$	300.4	115	88.2	230
$\times$	$\surd$	$\times$	294.8	70	29.6	70
$\times$	$\surd$	$\surd$	297.9	33	14.0	66
$\surd$	$\times$	$\times$	297.1	62	229.6	677
$\surd$	$\times$	$\surd$	296.3	26	50.0	301
$\surd$	$\surd$	$\times$	294.0	60	83.6	547
$\surd$	$\surd$	$\surd$	296.4	28	41.3	129

Table C.2: Performance of the various algorithms for a single trial with 20% hidden data and 25 training cases. Remember, the goal is to *minimize* the negative log-likelihood (NLL). "#iter" counts the number of iterations of the outer loop. "#fn" counts the number of times the likelihood function was called. The step size is fixed at 1.0. EM corresponds to the third line.

Linemin.?	Gen. grad.?	Conj.?	final NLL	#iter	CPU/s	#fn.
$\times$	$\times$	$\times$	210.5	129	121.8	258
$\times$	$\times$	$\surd$	210.4	122	112.6	244
$\times$	$\surd$	$\times$	202.5	80	44.3	160
$\times$	$\surd$	$\surd$	215.3	23	12.6	46
$\surd$	$\times$	$\times$	215.4	53	215.3	580
$\surd$	$\times$	$\surd$	210.3	46	100.3	524
$\surd$	$\surd$	$\times$	201.4	79	143.0	737
$\surd$	$\surd$	$\surd$	204.1	55	102.4	528

Table C.3: Performance of the various algorithms with 50% hidden data. The starting point is the same as the previous experiment. Surprisingly, the best NLL is lower in this case. EM corresponds to the third line.

On the other hand, if the CPD does not have a closed form MLE, it is necessary to use gradient methods in the M step (this is called generalized EM). In this case, the argument in favor of EM is somewhat less compelling. Unfortunately, we did not compare the algorithms in this case.

C.4.4 Local minima

The posterior over parameters in the partially observable case is heavily multi-modal. This means that local search algorithms, like gradient ascent and EM, are prone to get stuck in local optima. A simple solution is multiple restarts. Another popular solution is simulated annealing [KJV94]; unfortunately, this can be slow, because of the wasteful "propose, evaluate, reject" cycle.

Deterministic annealing (DA) [Ros98] is a faster alternative to simulated annealing, and works by enforcing a certain level of entropy (noise) in the system, which is gradually reduced. Recall from Section C.4.2 that the variational free energy is the expected energy, $\langle -\log P_\theta(H = h, V_m)\rangle_q$, plus the entropy, $H(q)$. The idea behind DA is to multiply the entropy by a temperature term T; initially the temperature is high, which "smooths out" the energy surface, so it is easy to find the maximum; then the temperature is gradually reduced to $T = 1$, which corresponds to the original problem. If the temperature is reduced slowly enough, it is possible to "track" the global optimum; this is called a "continuation method".

In an EM context [UN98], pre-multiplying $H(q)$ by T changes the E step: high temperatures essentially make assignments to discrete variables "softer". (The M step is unchanged.) For clustering problems, there is a single discrete hidden variable, so we can change the E step in the following straightforward way: the posterior $P(H|v) = P(H, v)/\sum_h P(h, v)$ becomes

$$f(H|v) = \frac{P(H, v)^\beta}{\sum_h P(h, v)^\beta}$$

```
while not converged
    for each case m
        for each node i
            oldESS(i, m) := ESS(i, m)
            ESS(i, m) := computeESS(D_m(X_i, Pa(X_i)), θ_i)
            ESS(i) += ESS(i, m) - oldESS(i, m)
            θ_i := maximizeParams(ESS(i), θ_i)
```

Figure C.2: Pseudo-code for online EM for a Bayes net where all CPDs are in the exponential family, and hence have finite expected sufficient statistics (ESS), based on [NH98].

where $\beta = 1/T$ is an inverse temperature. In the case of more complex graphical models, the posterior is over joint assignments; controlling the entropy of this distribution requires modifying the inference algorithm. For example, [RR01] explains how to apply DA to discriminatively train a bank of HMMs; in addition to the α and β, two new quantities need to be computed in the forwards and backwards passes.

A very simple and recent approach is presented in [ENFS02]. This exploits the fact that in many machine learning problems, the score of the hypothesis decomposes into a sum of terms, one per training case. By randomly reweighting the training cases, the search algorithm can be encouraged to explore new parts of space. This is different from boosting, which generates an ensemble of models which are then averaged together. The reweighting method is simple to implement and demonstrated big performance increases on BN structure and parameter learning problems.

C.4.5 Online parameter learning algorithms

The formula for computing the gradient of the log-likelihood and the formula for computing the ESS needed for the EM update both involve summing over all training cases. It is a simple modification to consider a single training example at a time, and thus to derive an online algorithm: see Figure C.2. This makes sense since initially the parameters will be unreliable, so the ESS will be inaccurate; hence it is best to update the parameters as soon as possible. This is sometimes called online or incremental EM [NH98].

The algorithm in Figure C.2 requires storing ESS for each CPD (which might be less than the number of nodes, if there is parameter tying) and for each data-case. An approximation is to use an exponentially decaying average of recently visited data points:

$$\text{ESS}(i) := \gamma \text{ESS}(i) + \text{computeESS}(D_m(X_i, \text{Pa}(X_i)), \theta_i),$$

where $0 < \gamma < 1$. This will not converge to the right answer, at least not if γ is held fixed, but using $\gamma \approx 1$ often gives a good approximation, and uses constant space (essential if the data is streaming in).

Instead of updating the parameters after every case, which can be slow, we can update after every B cases, where B is the batch size. $B \sim 10$ if often much faster than $B = 1$ and $B = M$ (which is batch EM) [Sto94, p40]. A way to compute the optimal batch size is given in [TMH01]. For large data sets, it is often unnecessary to use all the data [MTH01, HD02], which can result in huge time savings.

An alternative online learning algorithm for Bayes nets is presented in [BKS97] (see also [SW96] for the HMM case), based on the online learning framework of [KW97]. In this framework the new set of parameters $\Theta^{(t+1)}$ is chosen so that it maximises the normalized log-likelihood, but is not too different from the old parameters $\Theta^{(t)}$. More precisely, we choose $\Theta^{(t+1)}$ to maximise

$$F(\Theta) = \lambda l(\Theta) - d(\Theta, \Theta^{(t)}).$$

To make the problem more tractable, $l(\Theta)$ is replaced by a linear approximation. They show that by choosing the distance function d to be the L_2 norm, they get the projected gradient ascent rule of [BKRK97]; by choosing d to be χ^2, they get the EM(λ) rule; and by choosing d to be KL-distance, they get a generalized exponentiated gradient rule EG(λ). Their experimental results show that EM(λ) is much better than EG(λ), and that EM(λ) for $\lambda > 1$ converges much faster than for standard EM, which corresponds to $\lambda = 1$, as discussed. They also suggest using small batches to compute the expected sufficient statistics in the EM update step, but do not test the performance of this experimentally. That is, the rule becomes

$$\theta_{ijk}^{(t+1)} = \lambda \frac{E_{\Theta^{(t)}}(X_i = k, \text{Pa}(X_i) = j|D)}{E_{\Theta^{(t)}}(\text{Pa}(X_i) = j|D)} + (1-\lambda)\theta_{ijk}^{(t)}$$

where

$$E_{\Theta^{(t)}}(x_i^k, \pi_i^j|D) = \frac{1}{B}\sum_{m=1}^{B} P_{\Theta^{(t)}}(X_i = k, \text{Pa}(X_i) = j|D_m)$$

is a sample-based average. This has the advantage over the Neal and Hinton method of not needing to store many tables ESS[m], since we are adding the (normalized) ESS for the current case to the old parameter values.

It is folk wisdom in the neural network community that online (i.e., stochastic) gradient descent[8] is not only faster than batch gradient descent, but also works better, perhaps because by following an approximation to the total gradient, the algorithm is less likely to get stuck in local minima. Hence even in an offline setting, it can pay to do online updates.

For a Bayesian approach to online parameter learning, see Section 4.4.2.

C.5 Known structure, partial observability, Bayesian

EM and gradient ascent return a point estimate of the parameters. For many purposes, it is useful to return a distribution over the parameters. The most common method for achieving this is to use Gibbs sampling [GG84, GRS96], which can be thought of as a stochastic version of EM. One can also use variational Bayes [JJ00, Att00, GB00], which is based on assuming a factorized variational posterior of the form $P(\theta, H|V) \approx q(\theta|V)q(H|V)$. An alternative is expectation propagation [Min01], which is an iterative version of the classical moment matching approach which is common in online Bayesian inference (see Section B.7.2). See also [RS98], which uses a moment matching technique, where the weights correspond to possible completions of the dataset (hence this technique is only applicable to discrete data). The deterministic approximations usually assume unimodal posteriors; hence they are essentially MAP estimates with error bars.

C.6 Unknown structure, full observability, frequentist

When learning structure, we must consider the following issues:

- What is the hypothesis space? We can imagine searching the space of DAGs, equivalence classes of DAGs (PDAGs), undirected graphs, trees, node orderings, etc.
- What is the evaluation (scoring) function? i.e., how do we decide which model we prefer?
- What is the search algorithm? We can imagine local search (e.g., greedy hill climbing, possibly with multiple restarts) or global search (e.g., simulated annealing or genetic algorithms).

We discuss these issues in more detail below.

C.6.1 Search space

Trees

If we are willing to restrict our hypothesis space to trees, we can find the optimal ML tree in $O(MN^2)$ time (where N is the number of nodes and M is the number of data cases) using the famous Chow-Liu algorithm [CL68, Pea88]. We can use EM to extend this to mixtures of trees [MJ00]. Interestingly, learning the optimal ML path (a spanning tree in which no vertex has degree higher than two) is NP-hard [Mee01].

[8] The term stochastic EM usually refers to using a sampling algorithm in the E step [Mar01].

DAGs

The most common approach is to search through the space of DAGs. Unfortunately, the number of DAGs on N variables is $2^{O(N^2 \log N)}$ [Rob73, FK00]. (A more obvious upper bound is $O(2^{n^2})$, which is the number of $N \times N$ adjacency matrices.) For example, there are 543 DAGs on 4 nodes, and $O(10^{18})$ DAGs on 10 nodes. This means that attempts to find the "true" DAG, or even just to explore the posterior modes, are doomed to failure, although one might be able to find a good local maximum, which should be sufficient for density estimation purposes.

PDAGs

A PDAG (partially directed acyclic graph), also called a pattern or essential graph, represents a whole class of Markov equivalent DAGs. Two DAGs are Markov equivalent if they imply the same set of (conditional) independencies. For example, $X \to Y \to Z$, $X \leftarrow Y \to Z$ and $X \leftarrow Y \leftarrow Z$ are Markov equivalent, since they all represent $X \perp Z|Y$. In general, two graphs are Markov equivalent iff they have the same structure ignoring arc directions, and the same v-structures [VP90]. (A v-structure consists of converging directed edges into the same node, such as $X \to Y \leftarrow Z$.) Since we cannot distinguish members of the same Markov equivalence class if we only have observational data [CY99, Pea00], it makes sense to search in the space of PDAGs, which is smaller than the space of all DAGs (about 3.7–14 times smaller [GP01, Ste00]). We discuss methods to do this in Section C.6.2. Of course, using PDAGs is not appropriate if we have experimental (interventional) as well as observational data. (An intervention means setting/ forcing a node to a specific value, as opposed to observing that it has some value [Pea00].)

Variable orderings

Given a total ordering $\prec$, the likelihood decomposes into a product of terms, one per family, since the parents for each node can be chosen independently (there is no global acyclicity constraint). The following equation was first noted in [Bun91]:

$$\begin{aligned} P(D|\prec) &= \sum_{G \in \mathcal{G}_\prec} \prod_{i=1}^{n} \text{score}(X_i, \text{Pa}_G(X_i)|D) \\ &= \prod_i \sum_{U \in \mathcal{U}_{\prec,i}} \text{score}(X_i, U|D) \end{aligned} \tag{C.11}$$

$\mathcal{G}_\prec$ is the set of graphs consistent with the ordering $\prec$, and $\mathcal{U}_{\prec,i}$ is the set of legal parents for node i consistent with $\prec$. If we bound the fan-in (number of parents) by k, each summation in Equation C.11 takes $\binom{n}{k} \leq n^k$ time to compute, so the whole equation takes $O(n^{k+1})$ time.

Given an ordering, we can the find the best DAG consistent with that ordering using greedy selection, as in the K2 algorithm [CH92], or more sophisticated variable selection methods (see Section 6.1.2).

If the ordering is unknown, we can search for it, e.g, using MCMC [FK00] (this is an example of Rao-Blackwellisation). Not surprisingly, they claim this mixes much faster than MCMC over DAGs (see Section C.7). However, the space of orderings has size $N!$, which is still huge.

Interestingly, interventions give us some hints about the ordering. For example, in a biological setting, if we knockout gene X_1, and notice that genes X_2 and X_3 change from their "wildtype" state, but genes X_4 and X_5 do not, it suggests that X_1 is the ancestor of X_2 and X_3. This heuristic, together with a set covering algorithm, was used to learn acyclic boolean networks (i.e., binary, deterministic Bayes nets) from interventional data [ITK00].

Undirected graphs

When searching for undirected graphs, it is common to restrict attention to decomposable undirected graphs, so that the parameters of the resulting model can be estimated efficiently. See [DGJ01] for a stepwise selection approach, and [GGT00] for an MCMC approach.

Choose G somehow
While not converged
  For each G' in $\mathrm{nbd}(G)$
    Compute $\mathrm{score}(G')$
  $G^* := \arg\max_{G'} \mathrm{score}(G')$
  If $\mathrm{score}(G^*) > \mathrm{score}(G)$
    then $G := G^*$
    else converged := true

Figure C.3: Pseudo-code for hill-climbing. $\mathrm{nbd}(G)$ is the neighborhood of G, i.e., the models that can be reached by applying a single local change operator.

C.6.2 Search algorithm

For local search (whether deterministic or stochastic), the operators that move through space are usually adding, deleting or reversing a single arc; this defines the neighborhood of a graph, $\mathrm{nbd}(G)$. ([KC01] consider a richer set of local DAG transformations that gives better results.) In addition, we must specify a starting point (initial graph); this could be chosen using the PC algorithm (see below). As an example of a local search algorithm, the code for hill climbing is shown in Figure C.3.

When we make a local change to a model, we would like the change in its score to be local (see Section C.6.3); that way, evaluating the cost of many neighbors is fast. Similarly, if the current graph has a certain required property (e.g., acyclicity), we would like the cost of checking if each of the neighbors has this property to be constant time (i.e., independent of the model size). [GC01] present a method for checking acyclicity in constant time by using an auxiliary data structure called the ancestor matrix, and [GGT00] give efficient ways to check for decomposability of undirected graphs.

Global search comes in two flavors: stochastic local search, where we allow "downhill" moves (e.g., MCMC, which includes simulated annealing as a special case), and search algorithms that make non-local changes such as genetic algorithms.

The PC algorithm

We can find the globally optimal PDAG in $O(N^{k+1} N_{\mathrm{train}})$ time, where there are N nodes, N_{train} data cases, and each node has at most k neighbors, using the PC algorithm [SGS00, p84]. (This is an extension of the IC algorithm [PV91, Pea00], which takes $O(N^N N_{\mathrm{train}})$ time). This algorithm, an instance of the "constraint based approach", works as follows: start with a fully connected undirected graph, and remove an arc between X and Y if there is some set of nodes S s.t., $X \perp Y | S$ (we search for such separating subsets in increasing order of size); at the end, we can orient some of the undirected edges, so that we recover all the v-structures in the PDAG.

The PC algorithm will provably recover the generating PDAG if the conditional independency (CI) tests are all correct. For continuous data, we can implement the CI test using Fisher's z test; for discrete data, we can use a χ^2 test [SGS00, p95]. Testing if $X \perp Y | S$ for discrete random variables requires creating a table with $O(K^{|S|+2})$ entries, which requires a lot of time and samples. This is one of the main drawback of the PC algorithm. [CGK$^+$02] contains a more efficient algorithm. The other difficulty with the PC algorithm is how to implement CI tests on non-Gaussian data (e.g., mixed discrete-continuous). (The analogous problem for the the search & score methods is how to define the scoring function for complex CPDs.) One approach is discussed in [MT01]. A way of converting a Bayesian scoring metric into a CI test is given in the appendix of [Coo97].

A more sophisticated version of the PC algorithm, called FCI (fast causal inference), can handle the case where there is confounding due to latent common causes. However, FCI cannot handle models such as the one in Figure C.7, where there is a non-root latent variable. (The ability to handle arbitrary latent-variable models is one of the main strengths of the search and score techniques.)

C.6.3 Scoring function

If the search space is restricted (e.g., to trees), maximum likelihood is an adequate criterion. However, if the search space is any possible graph, then maximum likelihood would choose the fully connected (complete) graph, since this has the greatest number of parameters, and hence can achieve the highest likelihood. In such a case we will need to use other scoring metrics, which we discuss below.

A well-principled way to avoid this kind of over-fitting is to put a prior on models. By Bayes' rule, the MAP model is the one that maximizes

$$\Pr(G|D) = \frac{\Pr(D|G)\Pr(G)}{\Pr(D)}$$

where $P(D)$ is a constant independent of the model. If the prior probability is higher for simpler models (e.g., ones which are "sparser" in some sense), the $P(G)$ term has the effect of penalizing complex models.

Interestingly, it is not necessary to explicitly penalize complex structures through the structural prior. The marginal likelihood (sometimes called the evidence),

$$P(D|G) = \int_\theta P(D|G,\theta)P(\theta|G)$$

automatically penalizes more complex structures, because they have more parameters, and hence cannot give as much probability mass to the region of space where the data actually lies, because of the sum-to-one constraint. In other words, a complex model is more likely to be "right" by chance, and is therefore less believable. This phenomenon is called Ockham's razor (see e.g., [Mac95]). Of course, we can combine the marginal likelihood with a structural prior to get

$$\text{score}(G) \stackrel{\text{def}}{=} P(D|G)P(G)$$

If we assume all the parameters are independent, the marginal likelihood decomposes into a product of local terms, one per node:

$$\begin{aligned} P(D|G) &= \prod_{i=1}^n \int_{\theta_i} P(X_i|\text{Pa}(X_i),\theta_i)P(\theta_i) \\ &\stackrel{\text{def}}{=} \prod_{i=1}^n \text{score}(\text{Pa}(X_i), X_i) \end{aligned}$$

Under certain assumptions (global and local parameter independence (see Section C.3.1) plus conjugate priors), each of these integrals can be performed in closed form, so the marginal likelihood can be computed very efficiently. For example, in Section C.6.3, we discuss the case of multinomial CPDs with Dirichlet priors. See [GH94] for the case of linear-Gaussian CPDs with Normal-Wishart priors, and [BS94, Bun94] for a discussion of the general case.

If the priors are not conjugate, one can try to approximate the marginal likelihood. For example, [Hec98] shows that a Laplace approximation to the parameter posterior has the form

$$\log \Pr(D|G) \approx \log \Pr(D|G,\hat{\theta}_G) - \frac{d}{2}\log M$$

where M is the number of samples, $\hat{\theta}_G$ is the ML estimate of the parameters and d is the dimension (number of free parameters) of the model. This is called the Bayesian Information Criterion (BIC), and is equivalent to the Minimum Description Length (MDL) approach. The first term is just the likelihood and the second term is a penalty for model complexity. (Note that the BIC score is independent of the parameter prior.) The BIC score also decomposes into a product of local terms, one per node. For example, for multinomials, we have (following Equation C.1)

$$\begin{aligned} \text{BIC-score}(G) &= \sum_i \sum_m \log P(X_i|\text{Pa}(X_i),\hat{\theta}_i, D_m) - \frac{d_i}{2}\log M \\ &= \sum_i \sum_{jk} N_{ijk}\log\theta_{ijk} - \frac{d_i}{2}\log M \qquad \text{(C.12)} \end{aligned}$$

where $d_i = q_i(r_i - 1)$ is the number of parameters in X_i's CPT.

A major advantage of the fact that the score decomposes is that graphs that only differ by a single link have marginal likelihoods that differ by at most two terms, since all the others cancel. For example, let G_1 be the chain $X_1 \rightarrow X_2 \rightarrow X_3 \rightarrow X_4$, and G_2 be the same but with the middle arc reversed: $X_1 \rightarrow X_2 \leftarrow X_3 \rightarrow X_4$. Then

$$\begin{aligned} \frac{P(D|G_2)}{P(D|G_1)} &= \frac{\text{score}(X_1)\ \text{score}(X_1, X_2, X_3)\ \text{score}(X_3)\ \text{score}(X_3, X_4)}{\text{score}(X_1)\ \text{score}(X_1, X_2)\ \text{score}(X_2, X_3)\ \text{score}(X_3, X_4)} \\ &= \frac{\text{score}(X_1, X_2, X_3)\ \text{score}(X_3)}{\text{score}(X_1, X_2)\ \text{score}(X_2, X_3)} \end{aligned}$$

In general, if an arc to X_i is added or deleted, only score$(X_i|\Pi_i)$ needs to be evaluated; if an arc between X_i and X_j is reversed, only score$(X_i|\Pi_i)$ and score$(X_j|\Pi_j)$ need to be evaluated. This is important since, in greedy search, we need to know the score of $O(n^2)$ neighbors at each step, but only $O(n)$ of these scores change if the steps change one edge at a time.

The traditional limitation to local search in PDAG space has been the fact that the scoring function does not decompose into a set of local terms. This problem has recently been solved [Chi02].

Computing the marginal likelihood for the multinomial-Dirichlet

For the case of multinomial CPDs with Dirichlet priors, we discussed how to compute the marginal likelihood sequentially in Section C.3.1. To compute this in batch form, we simply compute the posterior means of the parameters (Equation C.3), and plug these expected values into the sample likelihood equation:

$$P(D|G) = P(D|\overline{\theta}, G) = \prod_{i=1}^{n} \prod_{j=1}^{q_i} \prod_{k=1}^{r_i} \overline{\theta}_{ijk}^{N_{ijk}} \tag{C.13}$$

Alternatively, this can be written as follows [CH92]:

$$\begin{aligned} P(D|G) &= \prod_{i=1}^{n} \prod_{j=1}^{q_i} \frac{B(\alpha_{ij1} + N_{ij1}, \ldots, \alpha_{i,j,r_i} + N_{i,j,r_i})}{B(\alpha_{ij1}, \ldots, \alpha_{i,j,r_i})} \\ &= \prod_{i=1}^{n} \prod_{j=1}^{q_i} \frac{\Gamma(\alpha_{ij})}{\Gamma(\alpha_{ij} + N_{ij})} \cdot \prod_{k=1}^{r_i} \frac{\Gamma(\alpha_{ijk} + N_{ijk})}{\Gamma(\alpha_{ijk})} \end{aligned} \tag{C.14}$$

([HGC95] call this the Bayesian Dirichlet (BD) score.)

For interventional data, Equation C.14 is modified by defining N_{ijk} to be the number of times $X_i = k$ is *passively observed* in the context $\Pi_i = j$, as shown by [CY99]. (The intuition is that *setting* $X_i = k$ does not tell us anything about how likely this event is to occur "by chance", and hence should not be counted). Hence, in addition to D, we need to keep a record of which variables were clamped (if any) in each data case.

C.7 Unknown structure, full observability, Bayesian

Since we cannot hope to return the posterior over all models (there are way too many of them), we try to evaluate the probability that certain features exist:

$$P(f|D) = \sum_{G \in \mathcal{G}} P(G|D) f(G)$$

for some indicator function f of interest (e.g., $f(G) = 1$ if graph G contains a certain edge). This is called Bayesian model averaging [HMRV99]. For example, we can summarize the posterior by creating a graph where the weight of an edge is proportional to the marginal probability that it exists.

Rather than trying to sum over all legal graphs, we can restrict our attention to those graphs that are relatively likely (relative to the most probable model found so far); this is called "Occam's window" [MR94].

```
Choose G somehow
While not converged
        Pick a G' u.a.r. from nbd(G)
        Compute R = P(G'|D)q(G|G') / P(G|D)q(G'|G)
        Sample u ~ Unif(0,1)
        If u < min{1, R}
           then G := G'
```

Figure C.4: Pseudo-code for the MC^3 algorithm. u.a.r. means uniformly at random.

In both cases, we need to compute

$$P(G|D) = \frac{P(D|G)P(G)}{\sum_{G'} P(D|G')P(G')}$$

The normalizing constant $P(D) = \sum_{G'} P(D|G')P(G')$ is intractable to compute, because there a super-exponential number of graphs. To avoid this intractability, we can use the Metropolis-Hastings (MH) algorithm, which only requires that we be able to compute the posterior odds between the current candidate model, G_1, and the proposed new model, G_2:

$$\frac{P(G_2|D)}{P(G_1|D)} = \frac{P(G_2)}{P(G_1)} \times \frac{P(D|G_2)}{P(D|G_1)} \tag{C.15}$$

The ratio of the evidences, $\frac{P(D|G_2)}{P(D|G_1)}$, is called the Bayes factor, and is the Bayesian equivalent of the likelihood ratio test.

The idea of applying the MH algorithm to graphical models was first proposed in [MY95], who called the technique MC^3, for MCMC Model Composition. The basic idea is to construct a Markov Chain whose state space is the set of all DAGs and whose stationary distribution is $P(G|D)$. We achieve this as follows. Define a transition matrix or kernel, $q(G'|G)$. (The only constraints on q are that the resulting chain should be irreducible and aperiodic.) We sample a new state G' from this proposal distribution, $G' \sim q(\cdot|G)$, and accept this new state with probability $\min\{1, R\}$, where R is the acceptance rate:

$$R = \frac{P(G'|D)}{P(G|D)} \times \frac{q(G|G')}{q(G'|G)}$$

(If the kernel is symmetric, so $q(G|G') = q(G'|G)$, the last term cancels, and the algorithm is called the Metropolis algorithm.) The idea is to sample from this chain for "long enough" to ensure it has converged to its stationary distribution (this is called the burn-in time) and throw these samples away; any further samples are then (non-independent) samples from the true posterior, $P(G|D)$, and can be used to estimate many quantities of interest, such as $P(f|D)$. This algorithm is given in pseudo-code in Figure C.4.

The issue of diagnosing convergence of MCMC algorithms is discussed in [GRS96]. The number of samples needed after reaching convergence depends on how rapidly the chain "mixes" (i.e., moves around the posterior distribution). To get a ballbark figure, [GC01] use MC^3 to find a distribution over the 3,781,503 DAGs with 6 binary nodes (of course, many of these are Markov equivalent), using a fully observed dataset with 1,841 cases. They used $T = 100,000$ iterations with no burn-in, but it seemed to converge after "only" 10,000 iterations. To scale up to larger problems, it will obviously be necessary to use Rao-Blackwellisation, such as searching over orderings instead of graphs (see Section C.6.1).

C.7.1 The proposal distribution

[MY95] suggested the following kernel. Define the neighborhood of the current state, nbd(G), to be the set of DAGs which differ by 1 edge from G, i.e., we can generate nbd(G) by considering all single edge additions, deletions and reversals, subject to the acyclicity constraint. Then let $q(G'|G) = 1/|\mathrm{nbd}(G)|$, for $G' \in \mathrm{nbd}(G)$, and $q(G'|G) = 0$ for $G' \notin \mathrm{nbd}(G)$, so

$$R = \frac{|\mathrm{nbd}(G)|P(G')P(D|G')}{|\mathrm{nbd}(G')|P(G)P(D|G))}$$

The main advantage of this proposal distribution is that it is efficient to compute R when G and G' only differ by a single edge (assuming complete data): see Section C.6.3.

The single-edge-change proposal can lead to high acceptance rates, but slow mixing, because it takes such small steps through the space. An alternative would be to propose large changes, such as swapping whole substructures, as in genetic programming. If one is not sure which proposal to use, one can always create a mixture distribution. The weights of this mixture are parameters that have to be tuned by hand. [GRG96] suggest that the kernel should be designed so that the average acceptance rate is 0.25.

A natural way to speed up mixing is to reduce the size of the search space. Suppose that, for each node, we restrict the maximum number of parents to be k, instead of n. (This is reasonable, since we expect the fan-in to be small). This reduces the number of parent sets we need to evaluate from $O(2^n)$ to $\binom{n}{k} \leq n^k$. Some heuristics for choosing the set of k potential parents are given in [FNP99]. As long as we give non-zero probability to all possible edge changes in our proposal, we are guaranteed to get the correct answer (since we can get from any graph to any other by single edge changes, and hence the chain is irreducible); heuristics merely help us reach the right answer faster. The heuristics in [FNP99] change with time, since they look for observed dependencies that cannot be explained by the current model. Such adaptive proposal distributions cause no theoretical problems for convergence.

C.8 Unknown structure, partial observability, frequentist

In the partially observable case, computing the marginal likelihood is intractable, requiring that we sum out all the latent variables Z as well as integrate out all the parameters θ:

$$P(X|G) = \sum_Z \int_\theta P(X, Z|G, \theta)P(\theta|G)$$

This score is hard to compute, and does not decompose into a product of local terms.

There are two approaches to this problem: try to approximate the marginal likelihood, and embed the approximation into any of the search spaces/algorithms discussed above, or use a different scoring function (the expected complete-data marginal likelihood) which does decompose. We will discuss each in turn.

C.8.1 Approximating the marginal likeihood

One accurate way of approximating the marginal likelihood, known as the Candidate's method [Chi95, Raf96], is as follows. We pick an arbitrary value θ^*_G (e.g., the MAP value), and compute

$$P(D|G) = \frac{P(D|\theta^*_G, G)P(\theta^*_G|G)}{P(\theta^*_G|D, G)}$$

Computing $P(\theta^*_G|G)$ is trivial, and computing $P(D|\theta^*_G, G)$ can be done using any BN inference algorithm. The denominator can be approximated using Gibbs sampling: see [CH97] for details. (See also [SNR00] for a related approach, based on the harmonic mean estimator.)

Various large sample approximations to the marginal likelihood, which are computationally cheaper, are compared in [CH97]. The conclusion is that the Cheeseman-Stutz (CS) approximation, which is a variation of BIC (see Section C.6.3), is the most accurate, at least in the case of naive-Bayes (mixture) models. However, the accuracy of these techniques for small samples is suspect. Even for large samples, these approximations are inaccurate at estimating $P(D|G_1)/P(D|G_2)$, where G_1 is the most probable model, and G_2 is the second most probable. That is, these approximations can be useful for model selection, but less so for model averaging.

Although the BIC and C-S scores decompose, local search algorithms are still expensive, because we need to run EM at each step to compute $\hat{\theta}$. Instead of doing EM inside of search, we can do search inside of EM; this turns out to be much faster: see Section C.8.2.

```
Choose G somehow
Choose θ somehow
While not converged
    Improve θ using parametric EM
    For each G' in nbd(G)
        Compute ESS(G') using G, θ [E step]
        Compute θ̂(G') using ESS(G')
        Compute Escore(G') using ESS(G'), θ̂(G')
    G* := arg max_{G'} Escore(G')
    If Escore(G*) > Escore(G)
        then G := G* [structural M step]
            θ := θ(G*) [parametric M step]
        else converged := true
```

Figure C.5: Pseudo-code for Structural EM. Compare with Figure C.3.

C.8.2 Structural EM

The key insight behind structural EM [Fri97] is that the expected complete-data log-likelihood, and hence the BIC score, does decompose. The algorithm works as follows: use the current model and parameters, (G, θ), to compute the expected sufficient statistics (ESS) for all models in the neighborhood of G; use ESS(G') to evaluate the expected BIC score of each neighbor G'; pick the best neighbor; iterate until convergence. See Figure C.5. (The algorithm can also be extended to use an (approximate) BDe score instead of BIC [Fri98].)

The analysis of the algorithm relies on the auxiliary Q function, just as in parametric EM (see Section C.4.2). In particular, define

$$Q(\theta', G'|\theta, G) = \sum_h P(h|V, \theta, G) \log P(V, h|\theta', G') + \text{penalty}(\theta', G')$$

The penalty depends on the dimensionality of G'. (The dimensionality of a model with latent variables is less than the number of its free parameters [GHM96].) [Fri97] proves the following theorem, by analogy with parametric EM:

Theorem [Friedman]. If $Q(G', \theta'|G, \theta) > Q(G, \theta|G, \theta)$, then $S(G', \theta') > S(G, \theta)$, where $S(G, \theta) = \log P(D|G, \theta) + \text{penalty}(G, \theta)$.

In other words, increasing the expected BIC score is guaranteed to increase the actual BIC score. In the case of multinomials, we define the expected BIC score by analogy with Equation C.12 to be

$$\text{EBIC-score}(G') = \sum_i \sum_{jk} \langle N_{ijk} \rangle \log \hat{\theta}_{ijk} - \frac{d_i}{2} \log M \tag{C.16}$$

where $d_i = q_i(r_i - 1)$ is the number of parameters in X_i's CPT in G', $\hat{\theta}$ are the MLE parameters for G' derived from $\langle N_{ijk} \rangle$, and

$$\langle N_{ijk} \rangle = \sum_m P(X_i = k, \text{Pa}_{G'}(X_i) = j | D_m, \theta, G)$$

This expected BIC score decomposes just like the regular BIC score.

Since we may add extra edges, G' might have families that are not in G. This means we may have to compute joint probability distributions on sets of nodes that are not in any clique: [EH01] describes efficient ways to do this using the jtree algorithm.

It is difficult to choose the initial graph structure. We cannot use the PC algorithm because we have (non-root) latent variables. On the other hand, we cannot choose, say, the empty graph, because our initial estimate of θ will be poor. We therefore assume we have some good initial guess based on domain knowledge.

Choose G_1 somehow
Compute $\hat{\theta}$ using EM
Sample $H_1 \sim P(H|G_1, \hat{\theta})$
for $t = 1, 2, \ldots$
  Pick a G' u.a.r. from nbd(G_t)
  Compute $\hat{B} = \frac{P(D,H_t|G')}{P(D,H_t|G_t)}$
  Compute $R = \frac{|\text{nbd}(G_t)|P(G')}{|\text{nbd}(G')|P(G_t)}\hat{B}$
  Sample $u \sim \text{Unif}(0, 1)$
  If $u < \min\{1, R\}$
    then $G_{t+1} = G'$
      Compute $P(\theta_{t+1}|D, H_t, G_{t+1})$ using Bayesian updating
      Compute $\overline{\theta}_{t+1}$ from $P(\theta_{t+1}|D, H_t, G_{t+1})$
      Sample $H_{t+1} \sim P(H|D, G_{t+1}, \overline{\theta}_{t+1})$ using Gibbs sampling

Figure C.6: Pseudo-code for the MC^3 algorithm modified to handle missing data. Choosing the initial graph structure is hard, as discussed in Section C.8.2.

C.9 Unknown structure, partial observability, Bayesian

Since we are already using MCMC to do Bayesian model averaging, it is natural to extend the state-space of the Markov chain so that it not only searches over models, but also over the values of the unobserved nodes [YMHL95]. The idea is to use Gibbs sampling, where we alternate between sampling a new model given the current completed data set, and sampling a completed data set given the current model. (This is basically an extension of the IP algorithm for data augmentation [TW87], and is similar to structural EM.)

At a high level, the algorithm cycles through the following steps (where V is the observed data and H is the hidden data):

1. Sample $G_{t+1} \sim P(G|D, H_t) \propto P(G)P(D, H_t|G)$
2. Compute $P(\theta_{t+1}|D, H_t, G_{t+1})$
3. Sample $Z_{t+1} \sim P(H|D, G_{t+1}, \overline{\theta}_{t+1})$

To avoid computing the normalizing constant implicit in the first step, we use the MH algorithm, and approximate the Bayes factor by using the current completed dataset. This is an approximation to the expected complete-data Bayes factor [Bun94, Raf96]:

$$\frac{P(D|G')}{P(D|G_t)} = E\left[\frac{P(D, H|G')}{P(D, H|G_t)} \mid D, G_t\right] \approx \frac{P(D, H_t|G')}{P(D, H_t|G_t)} \tag{C.17}$$

The second step can be performed in closed form for conjugate distributions (see Section C.3.1 for the multinomial-Dirichlet case). Finally, we sample from the predictive distribution, $P(H|D, G_{t+1})$, by using the mean parameter values, $\overline{\theta}_{t+1}$ (see Section C.3.1), with Gibbs sampling. The overall algorithm is sketched in Figure C.6. As far as I know, this has never been tried, probably because this algorithm is not only trying to search the space of all DAGs, but also the space of all assignments to the hidden variables.

C.10 Inventing new hidden nodes

So far, structure learning has meant finding the right connectivity between pre-existing nodes. A more challenging problem is inventing hidden nodes on demand. Hidden nodes can make a model much more compact, as we see in Figure C.7.

The standard approach is to keep adding hidden nodes one at a time, performing structure learning at each step, until the score drops. There has been some recent work on more intelligent heuristics. For example,

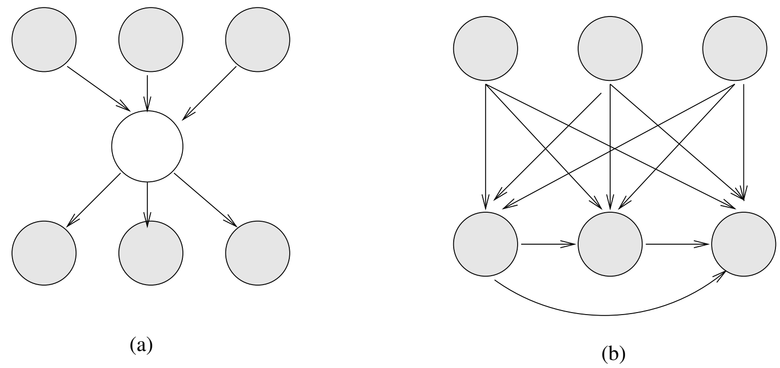


Figure C.7: (a) A BN with a hidden variable H. (b) The simplest network that can capture the same distribution without using a hidden variable (created using arc reversal and node elimination). If H is binary and the other nodes are trinary, and we assume full CPTs, the first network has 45 independent parameters, and the second has 708.

dense clique-like graphs (such as that in Figure C.7(b)) suggest that a hidden node should be added [ELFK00]. When learning DBNs, one can use violations of the Markov condition to suggest that a hidden node should be added [BFK99].

C.11 Derivation of the CLG parameter estimation formulas

In this section, we derive the equations in Table C.1, since I have not seen a published derivation at this level of generality before (although some of these details appear in [Bil99].) Recall that the expected complete-data log-likelihood is as follows:

$$L = -\tfrac{1}{2}\sum_m E\left[\sum_i q_m^i \log|\Sigma_i| + q_m^i (y_m - B_i x_m)'\Sigma_i^{-1}(y_m - B_i x_m) \mid D_m\right]$$

C.11.1 Estimating the regression matrix

To find the ML estimates, we take derivatives of L w.r.t. B_i and Σ_i, set to 0, and solve. Using the identity

$$\frac{\partial\left((Xa+b)'C(Xa+b)\right)}{\partial X} = (C+C')(Xa+b)a'$$

we have

$$\begin{aligned}\frac{\partial}{\partial B_i}L &= \tfrac{1}{2}\sum_l w_m^i E\left[2\Sigma_i^{-1}(y_l - B_i x_l)x_l'\right] \\ &= \sum_l w_m^i \Sigma_i^{-1}\langle y_m x_m'\rangle_i - \sum_l w_m^i \Sigma_i^{-1} B_i \langle x_m x_m'\rangle_i = 0\end{aligned}$$

Hence

$$B_i = \left(\sum_l w_m^i \langle y_m x_m'\rangle_i\right)\left(\sum_l w_m^i \langle x_m x_m'\rangle_i\right)^{-1}$$

In the special case that X is absent, so $B_i = \mu_i$ and $x_l = 1$, this formula becomes

$$\mu_i = \frac{\sum_l w_m^i \langle y_m\rangle_i}{\sum_l w_m^i}$$

C.11.2 Estimating a full covariance matrix

Using the identities

$$\frac{\partial(a'Xb)}{\partial X} = ab', \frac{\partial \ln|X|}{\partial X} = (X')^{-1} \text{ and } \ln|X| = -\ln|X^{-1}|$$

we have

$$\begin{aligned}
\frac{\partial}{\partial \Sigma_i^{-1}} L &= -\tfrac{1}{2}\sum_l w_m^i \langle (y_l - B_i x_l)(y_l - B_i x_l)' \rangle_i + \tfrac{1}{2}\sum_l w_m^i \Sigma_i \\
&= -\tfrac{1}{2}\sum_l w_m^i \left(\langle y_m y_m' \rangle_i - B_i \langle x_m y_m' \rangle_i - \langle y_m x_m' \rangle_i B_i' + B_i \langle x_m x_m' \rangle_i B_i' \right) + \tfrac{1}{2}\sum_l w_m^i \Sigma_i = 0
\end{aligned}$$

Using the new value of B_i and the fact that Σ_i is symmetric, we find

$$\begin{aligned}
B_i \left(\sum_l w_m^i \langle x_m x_m' \rangle_i \right) B_i' &= \left(\sum_l w_m^i \langle y_m x_m' \rangle_i \right) \left(\sum_l w_m^i \langle x_m x_m' \rangle_i \right)^{-1} \left(\sum_l w_m^i \langle x_m y_m' \rangle_i \right) \\
&= B_i \left(\sum_l w_m^i \langle x_m y_m' \rangle_i \right) = \left(\sum_l w_m^i \langle y_m x_m' \rangle_i \right)' B_i'
\end{aligned}$$

Hence

$$\Sigma_i = \frac{\sum_l w_m^i \langle y_m y_m' \rangle_i}{\sum_l w_m^i} - B_i \frac{\sum_l w_m^i \langle x_m y_m' \rangle_i}{\sum_l w_m^i} \tag{C.18}$$

If the covariance matrix is tied across states, we get

$$\frac{\partial}{\partial \Sigma^{-1}} L = -\tfrac{1}{2}\sum_l \sum_i w_m^i (\langle y_m y_m' \rangle_i - B_i \langle x_m y_m' \rangle_i) + \tfrac{1}{2}\Sigma \sum_l \sum_i w_m^i = 0$$

Since $\sum_l \sum_i w_m^i = N$ we have

$$\Sigma = \frac{\sum_i \left(\sum_l w_m^i \langle y_m y_m' \rangle_i \right)}{N} - \frac{\sum_i \left(\sum_l w_m^i B_i \langle x_m y_m' \rangle_i \right)}{N} \tag{C.19}$$

If X is absent, so $A_i = \mu_i$ and $x_l = 1$, Equation C.18 becomes (using the new estimate for μ_i)

$$\Sigma_i = \frac{\sum_l w_m^i \langle y_m y_m' \rangle_i}{\sum_l w_m^i} - \mu_i \mu_i' \tag{C.20}$$

In the tied case, we get

$$\Sigma = \frac{\sum_i \left(\sum_l w_m^i \langle y_m y_m' \rangle_i \right)}{N} - \frac{\sum_i \left(\sum_l w_m^i \mu_i \langle y_m \rangle_i' \right)}{N} = \frac{\sum_i \left(\sum_l w_m^i \langle y_m y_m' \rangle_i \right)}{N} - \frac{\sum_i (\sum_l w_m^i) \mu_i \mu_i'}{N} \tag{C.21}$$

C.11.3 Estimating a spherical covariance matrix

If we have the constraint that $\Sigma_i = \sigma_i^2 I$ is isotropic, the conditional density of Y becomes

$$p(y|x, Q=i) = c\sigma_i^{-d} \exp\left(-\tfrac{1}{2}\sigma_i^{-2} ||y - B_i x||^2 \right)$$

Hence

$$L = \sum_l \sum_i w_m^i \langle -d \log \sigma_i - \tfrac{1}{2}\sigma_i^{-2} ||y - B_i x||^2 \rangle_i$$

so

$$\frac{\partial}{\partial \sigma_i} L = -d \sum_l w_m^i \sigma_i^{-1} + \sigma_i^{-3} w_m^i \langle ||y_l - B_i x_l||^2 \rangle_i = 0$$

and

$$\sigma_i^2 = \frac{1/d}{\sum_l w_m^i} \left(\sum_l w_m^i \langle ||y_l - B_i x_l||^2 \rangle_i \right)$$

Now

$$||y_l - B_i x_l||^2 = (y_l - B_i x_l)'(y_l - B_i x_l) = y_l' y_l + x_l' B_i' B_i x_l - 2 y_l' B_i x_l$$

To compute the expected value of this distance, we use the fact that $x'Ay = \mathrm{Tr}(x'Ay) = \mathrm{Tr}(Ayx')$, so $E[x'Ay] = \mathrm{Tr}(AE[yx'])$. Hence

$$\sum_l w_m^i \langle (y_l - B_i x_l)'(y_l - B_i x_l) \rangle_i = \mathrm{Tr}(\sum_l w_m^i \langle y_m y_m' \rangle_i) + \mathrm{Tr}(\sum_l w_m^i B_i' B_i \langle x_m x_m' \rangle_i) - 2\mathrm{Tr}(\sum_l w_m^i B_i \langle x_m y_m' \rangle_i)$$

(Notice that, if Y is observed, this becomes

$$\sum_l w_m^i \langle (y_l - B_i x_l)'(y_l - B_i x_l) \rangle_i = \sum_l w_m^i y_l' y_l + \mathrm{Tr}(\sum_l w_m^i B_i' B_i \langle x_m x_m' \rangle_i) - 2 \sum_l y_l' w_m^i B_i \langle x_m \rangle_i$$

We will see later that, if $A_i = \mu_i$ and $X = 1$, we can eliminate all the trace operators.)

Now $\mathrm{Tr}(A) + \mathrm{Tr}(B) = \mathrm{Tr}(A + B)$, so

$$\sigma_i^2 = \frac{1/d}{\sum_l w_m^i} \mathrm{Tr} \left(\sum_l w_m^i \langle y_m y_m' \rangle_i + \sum_l w_m^i B_i' B_i \langle x_m x_m' \rangle_i - 2 \sum_l w_m^i B_i \langle x_m y_m' \rangle_i \right) \tag{C.22}$$

If σ_i^2 is tied, we get

$$\sigma^2 = \frac{1/d}{N} \mathrm{Tr} \left(\sum_i \sum_l w_m^i \langle y_m y_m' \rangle_i + \sum_i \sum_i w_m^i B_i' B_i \langle x_m x_m' \rangle_i - 2 \sum_i \sum_l w_m^i B_i \langle x_m y_m' \rangle_i \right) \tag{C.23}$$

In the special case that X is absent, so $B_i = \mu_i$ and $x_l = 1$, this formula becomes much simpler, since

$$\sum_l w_m^i \langle (y_l - B_i x_l)'(y_l - B_i x_l) \rangle_i = \sum_l w_m^i \left(\langle y_m' y_m \rangle_i + \mu_i' \mu_i - 2 \langle y_m \rangle_i' \mu_i \right)$$

so, using the new estimate for μ_i,

$$\sigma_i^2 = \frac{1}{d} \left(\frac{\sum_l w_m^i \langle y_m' y_m \rangle_i}{\sum_l w_m^i} - \mu_i' \mu_i \right) \tag{C.24}$$

For the tied case, we get

$$\sigma^2 = \frac{1}{Nd} \left(\sum_i \sum_l w_m^i \langle y_m' y_m \rangle_i + \sum_i (\sum_l w_m^i) \mu_i' \mu_i \right) \tag{C.25}$$

Appendix D

Notation and abbreviations

A summary of the most frequently used notation and abbreviations appears in the tables below. We also adopt the standard convention that random variables are denoted as capital letters, and instantiations of random variables (values) are denoted as lower-case letters. We do not explicitely distinguish between scalars and vectors. When working with linear algebra, we denote vectors (which may be random variables) as lower-case, to distinguish them from matrices, which are always upper case.

Symbol	Meaning
U_t	Control (intput) at time t
X_t	Hidden state variable at time t
Y_t	Observation (output) at time t
J	Size of input state space
K	Size of hidden state space
L	Size of output state space
T	Length of sequence
l	Lag
h	Prediction horizon
N_{train}	Num. training sequences
$y_{1:T}^m$	Training sequence m

Table D.1: Notation for general state-space models.

Symbol	Meaning
Z_t^i	i'th variable in timeslice t
N_h	Num. hidden variables per timeslice
N	Num. variables per timeslice
S	Num. hidden states, K^{N_h}

Table D.2: Notation for DBNs.

Symbol	Meaning
$\pi(j)$	Initial state probability: $P(X_1 = j)$
$A(i,j)$	Transition probability: $P(X_t = j \mid X_{t-1} = i)$
$B(i,j)$	Discrete observation probability: $P(Y_t = j \mid X_t = i)$
$O_t(i,i)$	Observation likelihood at time t $P(y_t \mid X_t = i)$
$\alpha_t(i)$	$P(X_t = i \mid y_{1:t})$
$\beta_t(i)$	$P(y_{t+1:T} \mid X_t = i$
$\gamma_t(i)$	$P(X_t = i \mid y_{1:T})$
$\xi_{t-1,t}(i,j)$	$P(X_{t-1} = i, X_t = j \mid y_{1:T})$

Table D.3: Notation for HMMs.

Symbol	Meaning
X_t^i	i'th sample of hidden variable X_t
$X_t^i(j)$	i'th sample of j'th component of hidden variable X_t
N_s	Num. samples
N_L	Num. landmarks

Table D.4: Notation for particle filtering.

$$P(X_t = x_t \mid X_{t-1} = x_{t-1}, U_t = u) = \mathcal{N}(x_t; Ax_{t-1} + Bu + \mu_X, Q)$$

$$P(Y_t = y \mid X_t = x, U_t = u) = \mathcal{N}(y; Cx + Du + \mu_Y, R)$$

$$\mathcal{N}(y; \mu, \Sigma) = \frac{1}{(2\pi)^{L/2}|\Sigma|^{\frac{1}{2}}} \exp\left(-\tfrac{1}{2}(y-\mu)'\Sigma^{-1}(y-\mu)\right)$$

Table D.5: Notation for a Kalman filter model.

Abbreviation	Meaning
BK	Boyen-Koller
CLG	Conditional linear Gaussian
CPD	Conditional probability distribution
DBN	Dynamic Bayesian network
EM	Expectation maximization
ESS	Expected sufficient statistics
FF	Factored frontier
Fgraph	factor graph
HMM	Hidden Markov model
HHMM	Hierarchical HMM
iid	independent, identically distributed
Jtree	Junction tree
KFM	Kalman filter model
LBP	Loopy belief propagation
MAP	Maximum a posteriori
MLE	Maximum likelihood estimate
PF	Particle filtering
RBPF	Rao-Blackwellised particle filtering
SLAM	Simultaneous localization and mapping

Table D.6: List of abbrevations.

Bibliography

[AC95] C. F. Aliferis and G. F. Cooper. A Structurally and Temporally Extended Bayesian Belief Netwok Model: Definitions, Properties and Modeling Techniques. In *UAI*, 1995.

[ACP87] S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. on Algebraic and Discrete Methods*, 8:277–284, 1987.

[AdFD00] C. Andrieu, N. de Freitas, and A. Doucet. Sequential Bayesian estimation and model selection for dynamic kernel machines. Technical report, Cambridge Univ., 2000.

[AK77] H. Akashi and H. Kumamoto. Random sampling approach to state estimation in switching environments. *Automatica*, 13:429–434, 1977.

[AM79] B. Anderson and J. Moore. *Optimal Filtering*. Prentice-Hall, 1979.

[AM00] S. M. Aji and R. J. McEliece. The generalized distributive law. *IEEE Trans. Info. Theory*, 46(2):325–343, March 2000.

[AMGC02] M. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A Tutorial on Particle Filters for Online Nonlinear/Non-Gaussian Bayesian Tracking. *IEEE Trans. on Signal Processing*, 50(2):174–189, February 2002.

[Ami01] E. Amir. Efficient approximation for triangulation of minimum treewidth. In *UAI*, 2001.

[Aok87] M. Aoki. *State space modeling of time series*. Springer, 1987.

[Arn85] S. A. Arnborg. Efficient algorithms for combinatorial problems on graphs with bounded decomposability - a survey. *Bit*, 25:2–23, 1985.

[Ast65] K. Astrom. Optimal control of Markov decision processes with incomplete state estimation. *J. Math. Anal. Applic.*, 10:174–205, 1965.

[Att00] H. Attias. A variational Bayesian framework for graphical models. In *NIPS-12*, 2000.

[Bar01] D. Barber. Tractable approximate belief propagation. In M. Opper and D. Saad, editors, *Advanced mean field methods*. MIT Press, 2001.

[BB72] U. Bertele and F. Brioschi. *Nonserial Dynamic Programming*. Academic Press, 1972.

[BB96] S. Bengio and Y. Bengio. An EM algorithm for asynchronous input/output HMMs. In *Proc. Intl. Conf. on Neural Info. Processing (ICONIP)*, 1996.

[BBT02] M. Bennewitz, W. Burgard, and S. Thrun. Learning motion patterns of persons for mobile service robots. In *Proc. Intl. Conf. on Robotics and Automation (ICRA)*, 2002.

[BC94] P. Baldi and Y. Chauvin. A smooth learning algorithm for hidden Markov models. *Neural Computation*, 6:305–316, 1994.

[BDG01] C. Boutilier, R. Dearden, and M. Goldszmidt. Stochastic dynamic programming with factored representations. *Artificial Intelligence*, 2001.

[Ben99] Y. Bengio. Markovian models for sequential data. *Neural Computing Surveys*, 2:129–162, 1999.

[Ber] A. Berger. Maxent and exponential models. http://www-2.cs.cmu.edu/ aberger/maxent.html.

[Ber85] J. Berger. *Statistical Decision Theory and Bayesian Analysis*. Springer-Verlag, 1985.

[BF95] Y. Bengio and P. Frasconi. Diffusion of context and credit information in markovian models. *J. of AI Research*, 3:249–270, 1995.

[BF96] Y. Bengio and P. Frasconi. Input/output HMMs for sequence processing. *IEEE Trans. on Neural Networks*, 7(5):1231–1249, 1996.

[BFG+93] R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. In *Intl. Conf. on Computed-Aided Design*, pages 188–191, 1993.

[BFGK96] C. Boutilier, N. Friedman, M. Goldszmidt, and D. Koller. Context-Specific Independence in Bayesian Networks. In *UAI*, 1996.

[BFK99] X. Boyen, N. Friedman, and D. Koller. Discovering the hidden structure of complex dynamic systems. In *UAI*, 1999.

[BG96] A. Becker and D. Geiger. A sufficiently fast algorithm for finding close to optimal junction trees. In *UAI*, 1996.

[BG01] C. Berzuini and W. Gilks. RESAMPLE-MOVE Filtering with Cross-Model Jumps. In A. Doucet, N. de Freitas, and N. Gordon, editors, *Sequential Monte Carlo Methods in Practice*. Springer, 2001.

[BGP97] E. Bienenstock, S. Geman, and D. Potter. Compositionality, MDL priors, and object recognition. In *NIPS*, 1997.

[BHK+93] M. P. Brown, R. Hughey, A. Krogh, I. S. Mian, K. Sjölander, and D. Haussler. Using dirichlet mixtures priors to derive hidden Markov models for protein families. In *Intl. Conf. on Intelligent Systems for Molecular Biology*, pages 47–55, 1993.

[Bil98] J. Bilmes. Data-driven extensions to HMM statistical dependencies. In *Intl. Conf. Speech Lang. Proc.*, 1998.

[Bil99] J. Bilmes. *Natural Statistical Models for Automatic Speech Recognition*. PhD thesis, CS Division, U.C. Berkeley, 1999.

[Bil00] J. Bilmes. Dynamic Bayesian multinets. In *UAI*, 2000.

[Bil01] J. A. Bilmes. Graphical models and automatic speech recognition. Technical Report UWEETR-2001-0005, Univ. Washington, Dept. of Elec. Eng., 2001.

[Bis95] C. M. Bishop. *Neural Networks for Pattern Recognition*. Clarendon Press, 1995.

[BJ01] F. Bach and M. I. Jordan. Kernel independent component analysis. Technical Report CSD-01-1166, Comp. Sci. Div., UC Berkeley, 2001.

[BJGGJ02] Z. Bar-Joseph, G. Gerber, D. Gifford, and T. Jaakkola. A new approach to analyzing gene expression time series data. In *6th Annual Intl. Conf. on Research in Computational Molecular Biology*, 2002.

[BK98a] X. Boyen and D. Koller. Approximate learning of dynamic models. In *NIPS-11*, 1998.

[BK98b] X. Boyen and D. Koller. Tractable inference for complex stochastic processes. In *UAI*, 1998.

[BK99] X. Boyen and D. Koller. Exploiting the architecture of dynamic systems. In *AAAI*, 1999.

[BKRK97] J. Binder, D. Koller, S. J. Russell, and K. Kanazawa. Adaptive probabilistic networks with hidden variables. *Machine Learning*, 29:213–244, 1997.

[BKS97] E. Bauer, D. Koller, and Y. Singer. Batch and on-line parameter estimation in Bayesian networks. In *UAI*, 1997.

[BLN01] O. Bangso, H. Langseth, and T. Nielsen. Structural learning in object oriented domains. In *Proc. 14th Intl Florida Articial Intelligence Research Society Conference (FLAIRS-2001)*, 2001.

[BMI99] C. Bielza, P. Mueller, and D. Ríos Insua. Decision analysis by augmented probability simulation. *Management Science*, 45(7):995–1007, 1999.

[BMR97a] J. Binder, K. Murphy, and S. Russell. Space-efficient inference in dynamic probabilistic networks. In *IJCAI*, 1997.

[BMR97b] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint satisfaction and optimization. *J. of the ACM*, 44(2):201–236, 1997.

[BPSW70] L. E. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occuring in the statistical analysis of probabalistic functions in markov chains. *The Annals of Mathematical Statistics*, 41:164–171, 1970.

[Bra96] M. Brand. Coupled hidden Markov models for modeling interacting processes. Technical Report 405, MIT Lab for Perceptual Computing, 1996.

[Bra99a] M. Brand. Structure learning in conditional probability models via an entropic prior and parameter extinction. *Neural Computation*, 11:1155–1182, 1999.

[Bra99b] T. Brants. Cascaded markov models. In *Proc. 9th Conf. of European Chapter of ACL*, 1999.

[BRP01] C. Boutilier, R. Reiter, and B. Price. Symbolic Dynamic Programming for First-Order MDPs. In *IJCAI*, 2001.

[BS90] Y. Bar-Shalom, editor. *Multitarget-multisensor tracking : advanced applications*. Artech House, 1990.

[BS94] J. Bernardo and A. Smith. *Bayesian Theory*. John Wiley, 1994.

[BSF88] Y. Bar-Shalom and T. Fortmann. *Tracking and data association*. Academic Press, 1988.

[BSL93] Y. Bar-Shalom and X. Li. *Estimation and Tracking: Principles, Techniques and Software*. Artech House, 1993.

[Bun89] W. Buntine. Decision tree induction systems: a Bayesian analysis. *Uncertainty in AI*, 3:109–127, 1989.

[Bun91] W. Buntine. Theory refinemement on Bayesian networks. In *UAI*, 1991.

[Bun94] W. L. Buntine. Operations for learning with graphical models. *J. of AI Research*, pages 159–225, 1994.

[BVW00a] H. Bui, S. Venkatesh, and G. West. On the recognition of abstract Markov policies. In *AAAI*, 2000.

[BVW00b] H. Bui, S. Venkatesh, and G. West. Policy Recognition in the Abstract Hidden Markov Model. Technical Report 4/2000, School of Computing Science, Curtin Univ. of Technology, Perth, 2000.

[BVW01] H. Bui, S. Venkatesh, and G. West. Tracking and surveillance in wide-area spatial environments using the Abstract Hidden Markov Model. *Intl. J. of Pattern Rec. and AI*, 2001.

[BW00] O. Bangso and P. Wuillemin. Top-down construction and repetitive structures representation in Bayesian networks. In *FLAIRS*, 2000.

[BZ02] J. Bilmes and G. Zweig. The graphical models toolkit: An open source software system for speech and time-series processing. In *Intl. Conf. on Acoustics, Speech and Signal Proc.*, 2002.

[CB97] A. Y. W. Cheuk and C. Boutilier. Structured arc reversal and simulation of dynamic probabilistic networks. In *UAI*, 1997.

[CD00] J. Cheng and M. Druzdzel. AIS-BN: An adaptive importance sampling algorithm for evidential reasoning in large Bayesian networks. *J. of AI Research*, 13:155–188, 2000.

[CDLS99] R. G. Cowell, A. P. Dawid, S. L. Lauritzen, and D. J. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Springer, 1999.

[CF95] K. C. Chang and R. M. Fung. Symbolic probabilistic inference with both discrete and continuous variables. *IEEE Trans. on Systems, Man, and Cybernetics*, 25(6):910–917, 1995.

[CG96] S. Chen and J. Goodman. An empirical study of smoothing techniques for language modeling. In *Proc. 34th ACL*, pages 310–318, 1996.

[CGH97] E. Castillo, J. M. Gutierrez, and A. S. Hadi. *Expert systems and probabilistic network models*. Springer, 1997.

[CGK+02] J. Cheng, R. Greiner, J. Kelly, D. Bell, and W. Liu. Learning bayesian networks from data: An information-theory based approach. *Artificial Intelligence Journal*, 2002. To appear.

[CH92] G. Cooper and E. Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9:309–347, 1992.

[CH96] I.J. Cox and S.L. Hingorani. An Efficient Implementation of Reid's Multiple Hypothesis Tracking Algorithm and its Evaluation for the Purpose of Visual Tracking. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 18(2):138–150, 1996.

[CH97] D. Chickering and D. Heckerman. Efficient approximations for the marginal likelihood of incomplete data given a Bayesian network. *Machine Learning*, 29:181–212, 1997.

[Chi95] S. Chib. Marginal likelihood from the Gibbs output. *JASA*, 90:1313–1321, 1995.

[Chi02] David Maxwell Chickering. Learning equivalence classes of Bayesian-network structures. *Journal of Machine Learning Research*, 2:445–498, February 2002.

[CK96] C. Carter and R. Kohn. Markov chain Monte Carlo in conditionally Gaussian state space models. *Biometrika*, 83:589–601, 1996.

[CL68] C. K. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Trans. on Info. Theory*, 14:462–67, 1968.

[CL00] R. Chen and S. Liu. Mixture Kalman filters. *J. Royal Stat. Soc. B*, 2000.

[CLR90] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *An Introduction to Algorithms*. MIT Press, 1990.

[Coo97] G. F. Cooper. A simple constraint-based algorithm for efficiently mining observational databases for causal relationships. *Data Mining and Knowledge Discovery*, 1:203–224, 1997.

[Coz00] F. Cozman. Generalizing variable-elimination in Bayesian Networks. In *Workshop on Prob. Reasoning in Bayesian Networks at SBIA/Iberamia*, pages 21–26, 2000.

[CR96] G Casella and C P Robert. Rao-Blackwellisation of sampling schemes. *Biometrika*, 83(1):81–94, 1996.

[CT91] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. John Wiley, 1991.

[CT02] K. C. Chang and Z. Tian. Efficient Inference for Mixed Bayesian Networks. In *Proc. 5th ISIF/IEEE Intl. Conf. on Info. Fusion (Fusion '02)*, 2002.

[CTS78] C. Cannings, E. A. Thompson, and M. H. Skolnick. Probability functions in complex pedigrees. *Advances in Applied Probability*, 10:26–61, 1978.

[Cur97] R. M. Curds. *Propagation Techniques in Probabilistic Expert Systems*. PhD thesis, Dept. Statistical Science, Univ. College London, 1997.

[CY99] G. Cooper and C. Yoo. Causal discovery from a mixture of experimental and observational data. In *UAI*, 1999.

[DA99] A. Doucet and C. Andrieu. Iterative Algorithms for State Estimation of Jump Markov Linear Systems. Technical report, Cambridge Univ. Engineering Dept., 1999.

[Dah00] R. Dahlhaus. Graphical interaction models for multivariate time series. *Metrika*, 51:157–172, 2000.

[Dar95] A. Darwiche. Conditioning algorithms for exact and approximate inference in causal networks. In *UAI*, 1995.

[Dar00] A. Darwiche. Recursive conditioning: Any space conditioning algorithm with treewidth bounded complexity. *Artificial Intelligence Journal*, 2000.

[Dar01] A. Darwiche. Constant Space Reasoning in Dynamic Bayesian Networks. *Intl. J. of Approximate Reasoning*, 26:161–178, 2001.

[Daw92] A. P. Dawid. Applications of a general propagation algorithm for probabilistic expert systems. *Statistics and Computing*, 2:25–36, 1992.

[DdFG01] A. Doucet, N. de Freitas, and N. J. Gordon. *Sequential Monte Carlo Methods in Practice*. Springer Verlag, 2001.

[DdFMR00] A. Doucet, N. de Freitas, K. Murphy, and S. Russell. Rao-blackwellised particle filtering for dynamic Bayesian networks. In *UAI*, 2000.

[DDN01] R. Deventer, J. Denzler, and H. Niemann. Bayesian Control of Dynamic Systems. Technical report, Lehrstuhl für Mustererkennung, Institut für Informatik, Uni. Nürnberg, 2001.

[DE00] R. Dahlhaus and M. Eichler. Causality and graphical models for time series. In P. Green, N. Hjort, and S. Richardson, editors, *Highly structured stochastic systems*. Oxford University Press, 2000.

[Dec98] R. Dechter. Bucket elimination: a unifying framework for probabilistic inference. In M. Jordan, editor, *Learning in Graphical Models*. MIT Press, 1998.

[DeG70] M. DeGroot. *Optimal Statistical Decisions*. McGraw-Hill, 1970.

[DEKM98] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, Cambridge, 1998.

[DF99] P. Dellaportas and J. Forster. Markov chain Monte Carlo model determination for hierarchical and graphical log-linear models. *Biometrika*, 1999. To appear.

[DG95] P. Dagum and A. Galper. Time-series prediction using belief network models. *Intl. J. of Human-Computer Studies*, 42:617–632, 1995.

[DGJ01] A. Deshpande, M. N. Garofalakis, and M. I. Jordan. Efficient stepwise selection in decomposable models. In *UAI*, 2001.

[DGK99] A. Doucet, N. Gordon, and V. Krishnamurthy. Particle Filters for State Estimation of Jump Markov Linear Systems. Technical report, Cambridge Univ. Engineering Dept., 1999.

[Die93] F. J. Diez. Parameter adjustment in Bayes networks. The generalized noisy-or gate. In *UAI*, 1993.

[DK89] T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Artificial Intelligence*, 93(1–2):1–27, 1989.

[DK00] J. Durbin and S. J. Koopman. Time series analysis of non-Gaussian observations based on state space models from both classical and Bayesian perspectives (with discussion). *J. Royal Stat. Soc. B*, 2000. To appear.

[DK01] J. Durbin and S. J. Koopman. *Time Series Analysis by State Space Methods*. Oxford University Press, 2001.

[DK02] J. Durbin and S.J. Koopman. A simple and efficient smoother for state space time series analysis. *Biometrika*, 2002. To appear.

[DL93] P. Dagum and M. Luby. Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial Intelligence*, 60:141–153, 1993.

[DLR77] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *J. of the Royal Statistical Society, Series B*, 34:1–38, 1977.

[DM95] E. Driver and D. Morrel. Implementation of continuous Bayesian networks usings sums of weighted Gaussians. In *UAI*, pages 134–140, 1995.

[DNCDW01] M. G. Dissanayake, P. Newman, S. Clark, and H. F. Durrant-Whyte. A Solution to the Simultaneous Localization and Map Building (SLAM) Problem. *IEEE Trans. Robotics and Automation*, 17(3):229–241, 2001.

[Dou98] A Doucet. On sequential simulation-based methods for Bayesian filtering. Technical report CUED/F-INFENG/TR 310, Department of Engineering, Cambridge University, 1998.

[DP97] A. Darwiche and G. Provan. Query DAGs: A practical paradigm for implementing belief-network inference. *J. of AI Research*, 6:147–176, 1997.

[DRO93] V. Digalakis, J. R. Rohlicek, and M. Ostendorf. ML estimation of a stochastic linear systems with the EM algorithm and its application to speech recognition. *IEEE Trans. on Speech and Audio Proc.*, 1(4):421–442, 1993.

[Duf02] M. Duff. *Optimal Learning: Computational procedures for Bayes-adaptive Markov decision processes*. PhD thesis, U. Mass. Dept. Comp. Sci., 2002.

[DW91] T. Dean and M. Wellman. *Planning and Control*. Morgan Kaufmann, 1991.

[Edw00] D. Edwards. *Introduction to graphical modelling*. Springer, 2000. 2nd edition.

[EH01] T. El-Hay. Efficient methods for exact and approximate inference in graphical models. Master's thesis, Hebrew Univ., Dept. Comp. Sci., 2001.

[Eic01] M. Eichler. Markov properties for graphical time series models. Technical report, Dept. Statistics, Univ. Heidelberg, 2001.

[EL01] D. Edwards and S. L. Lauritzen. The TM algorithm for Maximizing a Conditional Likelihood Function. *Biometrika*, 88:961–972, 2001.

[ELFK00] G. Elidan, N. Lotner, N. Friedman, and D. Koller. Discovering hidden variables: A structure-based approach. In *NIPS*, 2000.

[ENFS02] G. Elidan, M. Ninion, N. Friedman, and D. Schuurmans. Data perturbation for escaping local maxima in learning. In *AAAI*, 2002.

[FBT98] D. Fox, W. Burgard, and S. Thrun. Active Markov localization for mobile robots. *Robotics and Autonomous Systems*, 1998.

[FC89] R. Fung and K. Chang. Weighting and integrating evidence for stochastic simulation in Bayesian networks. In *UAI*, 1989.

[FG96] N. Friedman and M. Goldszmidt. Learning Bayesian networks with local structure. In *UAI*, 1996.

[FG97] N. Friedman and M. Goldszmidt. Sequential update of Bayesian network structure. In *UAI*, 1997.

[FGKP99] N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *IJCAI*, 1999.

[FGL00] N. Friedman, D. Geiger, and N. Lotner. Likelihood computations using value abstraction. In *UAI*, 2000.

[FHKR95] J. Forbes, T. Huang, K. Kanazawa, and S. Russell. The BATmobile: Towards a Bayesian automated taxi. In *IJCAI*, 1995.

[FK00] N. Friedman and D. Koller. Being Bayesian about network structure. In *UAI*, 2000.

[FKP98] N. Friedman, D. Koller, and A. Pfeffer. Structured representation of complex stochastic systems. In *AAAI*, 1998.

[FMR98] N. Friedman, K. Murphy, and S. Russell. Learning the structure of dynamic probabilistic networks. In *UAI*, 1998.

[FN00] N. Friedman and I. Nachman. Gaussian Process Networks. In *UAI*, 2000.

[FNP99] N. Friedman, I. Nachman, and D. Peer. Learning Bayesian network structure from massive datasets: The ”sparse candidate” algorithm. In *UAI*, 1999.

[FP69] D. C. Fraser and J. E. Potter. The optimum linear smoother as a combination of two optimum linear filters. *IEEE Trans. on Automatical Control*, pages 387–390, August 1969.

[FPC00] W. T. Freeman, E. C. Pasztor, and O. T. Carmichael. Learning low-level vision. *Intl. J. Computer Vision*, 2000.

[Fri97] N. Friedman. Learning Bayesian networks in the presence of missing values and hidden variables. In *UAI*, 1997.

[Fri98] N. Friedman. The Bayesian structural EM algorithm. In *UAI*, 1998.

[FS02] B. Fischer and J. Schumann. Generating data analysis programs from statistical models. *J. Functional Programming*, 2002.

[FST98] S. Fine, Y. Singer, and N. Tishby. The hierarchical Hidden Markov Model: Analysis and applications. *Machine Learning*, 32:41, 1998.

[FTBD01] D. Fox, S. Thrun, W. Burgard, and F. Dellaert. Particle filters for mobile robot localization. In A. Doucet, N. de Freitas, and N. Gordon, editors, *Sequential Monte Carlo Methods in Practice*. Springer, 2001.

[FW00] W. Freeman and Y. Weiss. On the fixed points of the max-product algorithm. *IEEE Trans. on Info. Theory*, 2000. To appear.

[Gal99] M. J. F. Gales. Semi-tied covariance matrices for hidden Markov models. *IEEE Trans. on Speech and Audio Processing*, 7(3):272–281, 1999.

[GB00] Z. Ghahramani and M. Beal. Propagation algorithms for variational Bayesian learning. In *NIPS-13*, 2000.

[GC92] R. P. Goldman and E. Charniak. Probabilistic text understanding. *Statistics and Computing*, 2(2):105–114, 1992.

[GC01] P. Giudici and R. Castelo. Improving Markov chain Monte Carlo model search for data mining. *Machine Learning*, 2001. To appear.

[GDW00] S. Godsill, A. Doucet, and M. West. Methodology for Monte Carlo Smoothing with Application to Time-Varying Autoregressions. In *Proc. Intl. Symp. on Frontiers of Time Series Modelling*, 2000.

[GG84] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 6(6), 1984.

[GGT00] P. Giudici, P. Green, and C. Tarantola. Efficient model determination for discrete graphical models. *Biometrika*, 2000. To appear.

[GH94] D. Geiger and D. Heckerman. Learning Gaussian networks. In *UAI*, volume 10, pages 235–243, 1994.

[GH96a] Z. Ghahramani and G. Hinton. The EM algorithm for mixtures of factor analyzers. Technical report, Dept. of Comp. Sci., Uni. Toronto, 1996.

[GH96b] Z. Ghahramani and G. Hinton. Parameter estimation for linear dynamical systems. Technical Report CRG-TR-96-2, Dept. Comp. Sci., Univ. Toronto, 1996.

[GH97] D. Geiger and D. Heckerman. A characterization of Dirchlet distributions through local and global independence. *Annals of Statistics*, 25:1344–1368, 1997.

[GH98] Z. Ghahramani and G. Hinton. Variational learning for switching state-space models. *Neural Computation*, 12(4):963–996, 1998.

[GHM96] D. Geiger, D. Heckerman, and C. Meek. Asymptotic model selection for directed networks with hidden variables. In *UAI*, 1996.

[GJ97] Z. Ghahramani and M. Jordan. Factorial hidden Markov models. *Machine Learning*, 29:245–273, 1997.

[GK95] S. Glesner and D. Koller. Constructing flexible dynamic belief networks from first-order probalistic knowledge bases. In *Symbolic and Quantitative Approaches to Reasoning and Uncertainty*, pages 217–226, 1995.

[GLS99] G. Gottlob, N. Leone, and F. Scarello. A comparison of structural CSP decomposition methods. In *IJCAI*, 1999.

[GMW81] P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization*. Academic Press, 1981.

[Goo99] J. Goodman. Semiring parsing. *Computational Linguistics*, 25(4):573–605, December 1999.

[Goo01] J. Goodman. Reduction of maximum entropy models to hidden markov models. Technical report, Microsoft Research, 2001.

[Gor93] N. Gordon. Novel approach to nonlinear/non-Gaussian Bayesian state estimation. *IEE Proceedings (F)*, 140(2):107–113, 1993.

[GP01] S. Gillispie and M. Perlman. Enumerating Markov Equivalence Classes of Acyclic Digraph Models. In *UAI*, 2001.

[GR98] Z. Ghahramani and S. Roweis. Learning Nonlinear Stochastic Dynamics using the Generalized EM Algorithm. In *NIPS*, 1998.

[Gre93] U. Grenander. *General Pattern Theory*. Oxford University Press, 1993.

[Gre98] P. Green. Reversible jump Markov chain Monte Carlo computation and Bayesian model determination. *Biometrika*, 82:711–732, 1998.

[GRG96] A. Gelman, G. Roberts, and W. Gilks. Efficient Metropolis jumping rules. In J. Bernardo, J. Berger, A. Dawid, and A. Smith, editors, *Bayesian Statistics 5*. Oxford, 1996.

[GRS96] W. Gilks, S. Richardson, and D. Spiegelhalter. *Markov Chain Monte Carlo in Practice*. Chapman and Hall, 1996.

[Gui02] L. Guibas. Sensing, tracking, and reasoning with relations. *IEEE Signal Processing Magazine*, March 2002.

[Hal96] F. L. Hall. Traffic stream characteristics. In N.H. Gartner, C.J. Messer, and A.K. Rathi, editors, *Traffic Flow Theory*. US Federal Highway Administration, 1996.

[Ham90] J. Hamilton. Analysis of time series subject to changes in regime. *J. Econometrics*, 45:39–70, 1990.

[Ham94] J. Hamilton. *Time Series Analysis*. Wiley, 1994.

[Har89] A. C. Harvey. *Forecasting, Structural Time Series Models, and the Kalman Filter*. Cambridge Univerity Press, 1989.

[HB94] D. Heckerman and J.S. Breese. Causal Independence for Probability Assessment and Inference Using Bayesian Networks. *IEEE Trans. on Systems, Man and Cybernetics*, 26(6):826–831, 1994.

[HCM+00] D. Heckerman, D. Chickering, C. Meek, R. Rounthwaite, and C. Kadie. Dependency networks for density estimation, collaborative filtering, and data visualization. Technical Report MSR-TR-00-16, Microsoft Research, 2000.

[HD96] C. Huang and A. Darwiche. Inference in belief networks: A procedural guide. *Intl. J. Approx. Reasoning*, 15(3):225–263, 1996.

[HD02] G. Hulten and P. Domingos. Learning from infinite data in finite time. In *NIPS-14*, 2002.

[Hec89] D. Heckerman. A tractable inference algorithm for diagnosing multiple diseases. In *UAI*, 1989.

[Hec93] D. Heckerman. Causal independence for knowledge acquistion and inference. In *UAI*, 1993.

[Hec95] D. Heckerman. A Bayesian approach to learning causal networks. In *UAI*, 1995.

[Hec98] D. Heckerman. A tutorial on learning with Bayesian networks. In M. Jordan, editor, *Learning in Graphical Models*. MIT Press, 1998.

[HGC95] D. Heckerman, D. Geiger, and M. Chickering. Learning Bayesian networks: the combination of knowledge and statistical data. *Machine Learning*, 1995.

[HIM$^+$00] M. Hu, C. Ingram, M.Sirski, C. Pal, S. Swamy, and C. Patten. A Hierarchical HMM Implementation for Vertebrate Gene Splice Site Prediction. Technical report, Dept. Computer Science, Univ. Waterloo, 2000.

[HMRV99] J. Hoeting, D. Madigan, A. Raftery, and C. Volinsky. Bayesian model averaging: A tutorial. *Statistical Science*, 4(4), 1999.

[Hoe01] J. Hoey. Hierarchical unsupervised learning of facial expression categories. In *ICCV Workshop on Detection and Recognition of Events in Video*, 2001.

[HSC88] E. Horvitz, H. Suermondt, and G. Cooper. Bounded cutset conditioning: An incremental-refinement approach to belief under uncertain resources. Technical Report KSL-88-36, Stanford Univ., 1988.

[HT01] K. Humphreys and M. Titterington. Some examples of recursive variational approximations for Bayesian inference. In M. Opper and D. Saad, editors, *Advanced mean field methods*. MIT Press, 2001.

[HTF01] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, 2001.

[HZ02] T. Heskes and O. Zoeter. Expectation propagation for approximate inference in dynamic Bayesian networks. In *UAI*, 2002.

[IB96] M. Isard and A. Blake. Contour tracking by stochastic propagation of conditional density. In *Proc. European Conf. on Computer Vision*, volume 1, pages 343–356, 1996.

[IB98] M. Isard and A. Blake. A smoothing filter for condensation. In *Proc. European Conf. on Computer Vision*, volume 1, pages 767–781, 1998.

[IB00] Y. Ivanov and A. Bobick. Recognition of visual activities and interactions by stochastic parsing. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 22(8):852–872, 2000.

[ITK00] T. Ideker, V. Thorsson, and R. Karp. Discovery of regulatory interactions through perturbation: inference and experimental design. In *Proc. of the Pacific Symp. on Biocomputing*, 2000.

[ITR$^+$01] T. Ideker, V. Thorsson, J. Ranish, R. Christmas, J. Buhler, R. Bumgarner, R. Aebersold, and L. Hood. Integrated genomic and proteomic analysis of a systematically perturned metabolic network. *Science*, 2001. Submitted.

[JA90] F. Jensen and S. K. Andersen. Approximations in Bayesian belief universes for knowledge-based systems. In *UAI*, 1990.

[Jaa01] T. Jaakkola. Tutorial on variational approximation methods. In M. Opper and D. Saad, editors, *Advanced mean field methods*. MIT Press, 2001.

[Jeb01] T. Jebara. *Discriminative, Generative and Imitative Learning*. PhD thesis, Media Lab, MIT, 2001.

[Jel97] F. Jelinek. *Statistical methods for speech recognition*. MIT Press, 1997.

[Jen01] F. V. Jensen. *Bayesian Networks and Decision Graphs*. Springer-Verlag, 2001.

[JGJS98] M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul. An introduction to variational methods for graphical models. In M. Jordan, editor, *Learning in Graphical Models*. MIT Press, 1998.

[JGS96] M. I. Jordan, Z. Ghahramani, and L. K. Saul. Hidden Markov decision trees. In *NIPS*, 1996.

[JH99] T. S. Jaakkola and D. Haussler. Exploiting generative models in discriminative classifiers. In *NIPS-10*, 1999.

[JJ93] M. Jamshidian and R. I. Jennrich. Conjugate gradient acceleration of the EM algorithm. *JASA*, 88(421):221–228, 1993.

[JJ94a] F. V. Jensen and F. Jensen. Optimal junction trees. In *UAI*, 1994.

[JJ94b] M. I. Jordan and R. A. Jacobs. Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, 6:181–214, 1994.

[JJ96] T. Jaakkola and M. Jordan. Computing upper and lower bounds on likelihoods in intractable networks. In *UAI*, 1996.

[JJ00] T. S. Jaakkola and M. I. Jordan. Bayesian parameter estimation via variational methods. *Statistics and Computing*, 10:25–37, 2000.

[JJD94] F. V. Jensen, F. Jensen, and S. L. Dittmer. From influence diagrams to junction trees. In *UAI*, 1994.

[JKK95] C. S. Jensen, A. Kong, and U. Kjaerulff. Blocking-gibbs sampling in very large probabilistic expert systems. *Intl. J. Human-Computer Studies*, pages 647–666, 1995.

[JKOP89] F. V. Jensen, U. Kjaerulff, K. G. Olesen, and J. Pedersen. An expert system for control of waste water treatment — a pilot project. Technical report, Univ. Aalborg, Judex Datasystemer, 1989. In Danish.

[JLM92] F. Jelinek, J. D. Lafferty, and R. L. Mercer. Basic methods of probabilistic context-free grammars. *Computational Linguistics*, To appear, 1992.

[JLO90] F. Jensen, S. L. Lauritzen, and K. G. Olesen. Bayesian updating in causal probabilistic networks by local computations. *Computational Statistics Quarterly*, 4:269–282, 1990.

[JM00] D. Jurafsky and J. H. Martin. *Speech and language processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice-Hall, 2000.

[Jor95] M. I. Jordan. Why the logistic function? A tutorial discussion on probabilities and neural networks. Technical Report 9503, MIT Computational Cognitive Science Report, August 1995.

[Jor99] M. I. Jordan, editor. *Learning in Graphical Models*. MIT Press, 1999.

[Jor02] M. I. Jordan. An introduction to probabilistic graphical models, 2002. In preparation.

[JP95] R. Jirousek and S. Preucil. On the effective implementation of the iterative proportional fitting procedure. *Computational Statistics & Data Analysis*, 19:177–189, 1995.

[JX95] M. I. Jordan and L. Xu. Convergence results for the EM approach to mixtures of experts architectures. *Neural Networks*, 8:1409–1431, 1995.

[KA94] S. Kuo and O. Agazzi. Keyword spotting in poorly printed documents using pseudo 2-D Hidden Markov Models. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 16:842–848, 1994.

[KA96] N. Kumar and A. G. Andreou. A generalization of linear discriminant analysis in maximum likelihood framework. In *Proc. of the Joint Statistical Meeting, Statistical Computing section*, 1996.

[KC01] T. Kocak and R. Castelo. Improved Learning of Bayesian Networks. In *UAI*, 2001.

[KCB00] J. Kwon, B. Coifman, and P. Bickel. Day-to-day travel time trends and travel time prediction from loop detector data. *Transportation Research Record*, 1554, 2000.

[KDLC01] K. Kask, R. Dechter, J. Larrosa, and F. Cozman. Bucket-elimination for automated reasoning. Technical Report R92, UC Irvine ICS, 2001.

[KFL01] F. Kschischang, B. Frey, and H-A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Trans Info. Theory*, February 2001.

[Kim94] C-J. Kim. Dynamic linear models with Markov-switching. *J. of Econometrics*, 60:1–22, 1994.

[Kim01] K-E. Kim. *Representations and Algorithms for Large Stochastic Planning Problems*. PhD thesis, Brown U., Dept. Comp. Sci., 2001.

[Kja90] U. Kjaerulff. Triangulation of graphs – algorithms giving small total state space. Technical Report R-90-09, Dept. of Math. and Comp. Sci., Aalborg Univ., Denmark, 1990.

[Kja92] U. Kjaerulff. Optimal decomposition of probabilistic networks by simulated annealing. In *Statistics and Computing*, volume 2, pages 7–17, 1992.

[Kja94] U. Kjaerulff. Reduction of computational complexity in bayesian networks through removal of weak dependencies. In *UAI*, 1994.

[Kja95] U. Kjaerulff. dHugin: A computational system for dynamic time-sliced Bayesian networks. *Intl. J. of Forecasting*, 11:89–111, 1995.

[KJV94] S. Kirkpatrick, C. Gelatt Jtr., and M. Vecchi. Optimization by simulated annealing. *Science*, 220:76–86, 1994.

[KKR95] K. Kanazawa, D. Koller, and S. Russell. Stochastic simulation algorithms for dynamic probabilistic networks. In *UAI*, 1995.

[KL01] D. Koller and U. Lerner. Sampling in Factored Dynamic Systems. In A. Doucet, N. de Freitas, and N. Gordon, editors, *Sequential Monte Carlo Methods in Practice*. Springer, 2001.

[KLD01] K. Kask, J. Larrosa, and R. Dechter. Up and down mini-buckets: A scheme for approximating combinatorial optimization tasks. Technical Report R91, UC Irvine ICS, 2001.

[KM00] J. Kwon and K. Murphy. Modeling freeway traffic with coupled HMMs. Technical report, Univ. California, Berkeley, 2000.

[KMN99] M. Kearns, Y. Mansour, and A. Ng. Approximate planning in large POMDPs via reusable trajectories. In *NIPS-12*, 1999.

[KMS+98] P. Kontkanen, P. Mullymäki, T. Silander, H. Tirri, and P. Grünwald. A comparison of non-informative priors for Bayesian networks. In *The Yearbook of the Finnish Statistical Society 1997*, pages 53–62. ?, 1998.

[KN98] C-J. Kim and C. Nelson. *State-Space Models with Regime-Switching: Classical and Gibbs-Sampling Approaches with Applications*. MIT Press, 1998.

[KP97] D. Koller and A. Pfeffer. Object-Oriented Bayesian Networks. In *UAI*, 1997.

[KW97] J. Kivinen and M. Warmuth. Additive versus exponentiated gradient updates for linear prediction. *Info. and Computation*, 132(1):1–64, 1997.

[Lau92] S. L. Lauritzen. Propagation of probabilities, means and variances in mixed graphical association models. *JASA*, 87(420):1098–1108, December 1992.

[Lau95] S. L. Lauritzen. The EM algorithm for graphical association models with missing data. *Computational Statistics and Data Analysis*, 19:191–201, 1995.

[Lau96] S. Lauritzen. *Graphical Models*. OUP, 1996.

[LB01] H. Langseth and O. Bangsø. Parameter learning in object oriented Bayesian networks. *Annals of Mathematics and Artificial Intelligence*, 2001. Forthcomming.

[LBN96] B. Levy, A. Benveniste, and R. Nikoukhah. High-level primitives for recursive maximum likelihood estimation. *IEEE Trans. Automatic Control*, 41(8):1125–1145, 1996.

[LC95] J. S. Liu and R. Chen. Blind deconvolution via sequential imputations. *JASA*, 90:567–576, 1995.

[LC98] J. Liu and R. Chen. Sequential Monte Carlo methods for dynamic systems. *JASA*, 93:1032–1044, 1998.

[LD94] Z. Li and B. D'Ambrosio. Efficient inference in Bayes networks as a combinatorial optimization problem. *Intl. J. Approximate Reasoning*, 11(1):55–81, 1994.

[LJ97] S. L. Lauritzen and F. V. Jensen. Local computation with valuations from a commutative semigroup. *Annals of Mathematics and Artificial Intelligence*, 21(1):51–69, 1997.

[LJ01] S. Lauritzen and F. Jensen. Stable local computation with conditional Gaussian distributions. *Statistics and Computing*, 11:191–203, 2001.

[Lju87] L. Ljung. *System Identificiation: Theory for the User*. Prentice Hall, 1987.

[LM97] K. Laskey and S. M. Mahoney. Network fragments: Representing knowledge for constructing probabilistic models. In *UAI*, 1997.

[LMP01] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Intl. Conf. on Machine Learning*, 2001.

[LN01] S. Lauritzen and D. Nilsson. Representing and solving decision problems with limited information. *Management Science*, 47:1238–1251, 2001.

[LP01] U. Lerner and R. Parr. Inference in hybrid networks: Theoretical limits and practical algorithms. In *UAI*, 2001.

[LS83] L. Ljung and T. Söderström. *Theory and Practice of Recursive Identification*. MIT Press, 1983.

[LS98] V. Lepar and P. P. Shenoy. A Comparison of Lauritzen-Spiegelhalter, Hugin and Shenoy-Shafer Architectures for Computing Marginals of Probability Distributions. In G. Cooper and S. Moral, editors, *UAI*, pages 328–337. Morgan Kaufmann, 1998.

[LSK01] U. Lerner, E. Segal, and D. Koller. Exact inference in networks with discrete children of continuous parents. In *UAI*, 2001.

[LSS01] M. Littman, R. Sutton, and S. Singh. Predictive representations of state. In *NIPS*, 2001.

[LW89] S. L. Lauritzen and N. Wermuth. Graphical models for associations between variables, some of which are qualitative and some quantitative. *Annals of Statistics*, 17:31–57, 1989.

[LW01] J. Liu and M. West. Combined Parameter and State Estimation in Simulation-Based Filtering. In A. Doucet, N. de Freitas, and N. Gordon, editors, *Sequential Monte Carlo Methods in Practice*. Springer, 2001.

[Mac95] D. MacKay. Probable networks and plausible predictions — a review of practical Bayesian methods for supervised neural networks. *Network*, 1995.

[Mac98] D. MacKay. Introduction to Monte Carlo methods. In M. Jordan, editor, *Learning in graphical models*. MIT Press, 1998.

[Mar01] I. C. Marschner. On stochastic version of the EM algorithm. *Biometrika*, 88:281–286, 2001.

[May90] A. D. May. *Traffic Flow Fundamentals*. Prentice Hall, 1990.

[MB99] P. J. Mosterman and G. Biswas. Diagnosis of continuous valued systems in transient operating regions. *IEEE Trans. on Systems, Man, and Cybernetics, Part A*, 29(6):554–565, 1999.

[McC95] A. McCallum. *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, Univ. Rochester, 1995.

[MCH02] Christopher Meek, David Maxwell Chickering, and David Heckerman. Autoregressive tree models for time-series analysis. In *Proceedings of the Second International SIAM Conference on Data Mining*, pages 229–244, Arlington, VA, April 2002. SIAM.

[MD99] A. Madsen and B. D'Ambrosio. A factorized representation of independence of causal influence and lazy propagation. In *AAAI*, 1999.

[ME85] H. Moravec and A. Elfes. High resolution maps from wide angle sonar. In *ICRA*, 1985.

[Mee01] Christopher Meek. Finding a path is harder than finding a tree. *Journal of Artificial Intelligence Research*, 15:383–389, 2001. http://www.jair.org/abstracts/meek01a.html.

[MF92] F. Martinirie and P. Forster. Data Association and Tracking Using Hidden Markov Models and Dynamic Programming. In *Intl. Conf. on Acoustics, Speech and Signal Proc.*, 1992.

[MFP00] A. McCallum, D. Freitag, and F. Pereira. Maximum Entropy Markov Models for Information Extraction and Segmentation. In *Intl. Conf. on Machine Learning*, 2000.

[MH97] C. Meek and D. Heckerman. Structure and parameter learning for causal independence and causal interaction models. In *UAI*, pages 366–375, 1997.

[Min99] T. Minka. From Hidden Markov Models to Linear Dynamical Systems. Technical report, MIT, 1999.

[Min00a] T. Minka. Bayesian linear regression. Technical report, MIT, 2000.

[Min00b] T. Minka. Estimating a Dirichlet distribution. Technical report, MIT, 2000.

[Min00c] T. Minka. Inferring a Gaussian distribution. Technical report, MIT, 2000.

[Min01] T. Minka. *A family of algorithms for approximate Bayesian inference*. PhD thesis, MIT, 2001.

[MJ97] M. Meila and M. Jordan. Triangulation by continuous embedding. Technical Report 1605, MIT AI Lab, 1997.

[MJ99] A. Madsen and F. Jensen. Lazy propagation: a junction tree inference algorithm based on lazy evaluation. *Artificial Intelligence*, 113:203–245, 1999.

[MJ00] M. Meila and M. I. Jordan. Learning with mixtures of trees. *J. of Machine Learning Research*, 1:1–48, 2000.

[MK97] G. J. McLachlan and T. Krishnan. *The EM Algorithm and Extensions*. Wiley, 1997.

[ML02] T. Minka and J. Lafferty. Expectation propagation for the generative aspect model. In *UAI*, 2002.

[MM93] O. Maron and A. W. Moore. Hoeffding races: Accelerating model selection search for classification and function approximation. In *NIPS-6*, 1993.

[MMC98] R. J. McEliece, D. J. C. MacKay, and J. F. Cheng. Turbo decoding as an instance of Pearl's 'belief propagation' algorithm. *IEEE J. on Selectred Areas in Comm.*, 16(2):140–152, 1998.

[MN83] McCullagh and Nelder. *Generalized Linear Models*. Chapman and Hall, 1983.

[Moh96] M. Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 20(1):1–33, 1996.

[Moo73] J. Moore. Discrete-time fixed-lag smoothing algorithms. *Automatica*, 9:163–173, 1973.

[MP95] A. Manna and A. Pnueli. *Temporal verification of Reactive Systems*. Springer, 1995.

[MP01] K. Murphy and M. Paskin. Linear time inference in hierarchical HMMs. In *NIPS*, 2001.

[MR94] D. Madigan and A. Raftery. Model selection and accounting for model uncertainty in graphical models using Occam's window. *JASA*, 89:1535–1546, 1994.

[MR01] K. Murphy and S. Russell. Rao-Blackwellised Particle Filtering for Dynamic Bayesian Networks. In A. Doucet, N. de Freitas, and N. Gordon, editors, *Sequential Monte Carlo Methods in Practice*. Springer, 2001.

[MT01] D. Margaritis and S. Thrun. A Bayesian Multiresolution Independence Test for Continuous Variables. In *UAI*, 2001.

[MTH01] C. Meek, B. Thiesson, and D. Heckerman. A learning-curve approach to clustering. Technical Report MSR-TR-01-34, Microsoft Research, 2001.

[MTKW02] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem. In *AAAI*, 2002.

[Mur98] K. P. Murphy. Switching Kalman filters. Technical report, DEC/Compaq Cambridge Research Labs, 1998.

[Mur99] K. P. Murphy. A variational approximation for Bayesian networks with discrete and continuous latent variables. In *UAI*, 1999.

[Mur00] K. P. Murphy. Bayesian map learning in dynamic environments. In *NIPS-12*, pages 1015–1021, 2000.

[Mur01a] K. Murphy. Active learning of causal Bayes net structure. Technical report, Comp. Sci. Div., UC Berkeley, 2001.

[Mur01b] K. Murphy. The Bayes Net Toolbox for Matlab. In *Computing Science and Statistics: Proceedings of the Interface*, volume 33, 2001.

[MvD97] X. L. Meng and D. van Dyk. The EM algorithm — an old folk song sung to a fast new tune (with Discussion). *J. Royal Stat. Soc. B*, 59:511–567, 1997.

[MW01] K. Murphy and Y. Weiss. The Factored Frontier Algorithm for Approximate Inference in DBNs. In *UAI*, 2001.

[MWJ99] K. Murphy, Y. Weiss, and M. Jordan. Loopy belief propagation for approximate inference: an empirical study. In *UAI*, 1999.

[MY95] D. Madigan and J. York. Bayesian graphical models for discrete data. *Intl. Statistical Review*, 63:215–232, 1995.

[MZ97] I. MacDonald and W. Zucchini. *Hiden Markov and Other Models for Discrete Valued Time Series*. Chapman Hall, 1997.

[NB94] A. E. Nicholson and J. M. Brady. Dynamic belief networks for discrete monitoring. *IEEE Systems, Man and Cybernetics*, 24(11):1593–1610, 1994.

[ND01] S. Narayanan and D.Jurafsky. A Bayesian Model Predicts Parse Preferences and Reading Times in Sentence Comprehension. In *NIPS*, 2001.

[Nea92] R. Neal. Connectionist learning of belief networks. *Artificial Intelligence*, 56:71–113, 1992.

[Nea93] R. Neal. Probabilistic Inference Using Markov Chain Monte Carlo Methods. Technical report, Univ. Toronto, 1993.

[NG01] D. Nilsson and J. Goldberger. Sequentially finding the N-Best List in Hidden Markov Models. In *IJCAI*, pages 1280–1285, 2001.

[NH97] Liem Ngo and Peter Haddawy. Answering queries from context-sensitive probabilistic knowledge bases. *Theoretical Computer Science*, 171(1–2):147–177, 1997.

[NH98] R. M. Neal and G. E. Hinton. A new view of the EM algorithm that justifies incremental and other variants. In M. Jordan, editor, *Learning in Graphical Models*. MIT Press, 1998.

[NI00] A. Nefian and M. Hayes III. Maximum likelihood training of the embedded HMM for face detection and recognition. In *IEEE Intl. Conf. on Image Processing*, 2000.

[Nik98] D. Nikovski. Learning stationary temporal probabilistic networks. In *Conf. on Automated Learning and Discovery*, 1998.

[Nil98] D. Nilsson. An efficient algorithm for finding the M most probable configurations in a probabilistic expert system. *Statistics and Computing*, 8:159–173, 1998.

[NJ01] A. Y. Ng and M. Jordan. Convergence rates of the Voting Gibbs classifier, with application to Bayesian feature selection. In *Intl. Conf. on Machine Learning*, 2001.

[NJ02] A. Y. Ng and M. I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *NIPS-14*, 2002.

[NLP$^+$02] A. Nefian, L. Liang, X. Pi, L. Xiaoxiang, C. Mao, and K. Murphy. A Coupled HMM for Audio-Visual Speech Recognition. In *Intl. Conf. on Acoustics, Speech and Signal Proc.*, 2002.

[NPP02] B. Ng, L. Peshkin, and A. Pfeffer. Factored particles for scalable monitoring. In *UAI*, 2002.

[NWJK00] T. Nielsen, P. Wuillemin, F. Jensen, and U. Kjaerulff. Using ROBDDs for inference in Bayesian networks with troubleshooting as an example. In *UAI*, 2000.

[NY00] H. J. Nock and S. J. Young. Loosely coupled HMMs for ASR. In *Intl. Conf. on Acoustics, Speech and Signal Proc.*, 2000.

[ODK96] M. Ostendorf, V. Digalakis, and O. Kimball. From HMM's to segment models: a unied view of stochastic modeling for speech recognition. *IEEE Trans. on Speech and Audio Processing*, 4(5):360–378, 1996.

[OKJ$^+$89] K. G. Olesen, U. Kjaerulff, F. Jensen, B. Falck, S. Andreassen, and S. K. Andersen. A munin network for the median nerve – a case study on loops. *Applied AI*, 3:384–403, 1989.

[Ole93] K. G. Olesen. Causal probabilistic networks with both discrete and continuous variables. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 3(15), 1993.

[OM96] P. Van Overschee and B. De Moor. *Subspace Identification for Linear Systems: Theory, Implementation, Applications*. Kluwer Academic Publishers, 1996.

[OM99] K. Olesen and A. Madsen. Maximal prime subgraph decomposition of bayesian networks. Technical Report R-99-5006, Dept. Comp. Sci., Aalborg Univ., 1999.

[Pea88] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[Pea00] J. Pearl. *Causality: Models, Reasoning and Inference*. Cambridge Univ. Press, 2000.

[Pfe00] A. Pfeffer. *Probabilistic Reasoning for Complex Systems*. PhD thesis, Dept. Comp. Sci., Stanford Univ., 2000.

[Pfe01] A. Pfeffer. Sufficiency, separability and temporal probabilistic models. In *UAI*, 2001.

[Pfl98] K. Pfleger. Categorical Boltzmann Machines. Technical Report KSL-98-05, Knowledge systems lab, Computer Science Department, Stanford University, 1998.

[PPL97] S. Della Pietra, V. Della Pietra, and J. Lafferty. Inducing features of random fields. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 19(4), 1997.

[PR97a] R. Parr and S. Russell. Reinforcement learning with hierarchies of machines. In *NIPS*, 1997.

[PR97b] F. Pereira and M. Riley. Speech recognition by composition of weighted finite automata. In *Finite-State Language Processing*, pages 431–453. MIT Press, 1997.

[PR01] H. Pasula and S. Russell. Approximate inference for first-order probabilistic languages. In *IJCAI*, 2001.

[PRCM99] V. Pavlovic, J. Rehg, T-J. Cham, and K. Murphy. A Dynamic Bayesian Network Approach to Figure Tracking Using Learned Dynamic Models. In *IEEE Conf. on Computer Vision and Pattern Recognition*, 1999.

[PRM00] V. Pavlovic, J. M. Rehg, and J. MacCormick. Learning switching linear models of human motion. In *NIPS-13*, 2000.

[PROR99] H. Pasula, S. Russell, M. Ostland, and Y. Ritov. Tracking many objects with many sensors. In *IJCAI*, 1999.

[PS91] M. Peot and R. Shachter. Fusion and propogation with multiple observations in belief networks. *Artificial Intelligence*, 48:299–318, 1991.

[PV91] J. Pearl and T. Verma. A theory of inferred causation. In *Knowledge Representation*, pages 441–452, 1991.

[PVTF88] W. Press, W. Vetterling, S. Teukolosky, and B. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, second edition, 1988.

[PW96] D. Pynadath and M. Wellman. Generalized Queries on Probabilistic Context-Free Grammars. In *UAI*, 1996.

[Rab89] L. R. Rabiner. A tutorial on Hidden Markov Models and selected applications in speech recognition. *Proc. of the IEEE*, 77(2):257–286, 1989.

[Raf96] A. Raftery. Hypothesis testing and model selection via posterior simulation. In *Markov Chain Monte Carlo in Practice*. Chapman and Hall, 1996.

[RBD00] M. Richardson, J. Bilmes, and C. Diorio. Hidden-articulatory Markov models for speech recognition. In *Intl. Conf. on Acoustics, Speech and Signal Proc.*, 2000.

[RBKK95] S. Russell, J. Binder, D. Koller, and K. Kanazawa. Local learning in probabilistic networks with hidden variables. In *IJCAI*, 1995.

[RD98] I. Rish and R. Dechter. On the impact of causal independence. Technical report, Dept. Information and Computer Science, UCI, 1998.

[RG99] S. Roweis and Z. Ghahramani. A Unifying Review of Linear Gaussian Models. *Neural Computation*, 11(2), 1999.

[RG01] D. Rusakov and D. Geiger. On the parameter priors for discrete DAG models. In *AI/Stats*, 2001.

[RN95] S. Russell and P. Norvig. *Instructor's Solution Manual to Accompany Artificial Intelligence: A Modern Approach*. Prentice Hall, Englewood Cliffs, NJ, 1995.

[RN02] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2002. 2nd edition, in preparation.

[Rob73] R. W. Robinson. Counting labeled acyclic digraphs. In F. Harary, editor, *New Directions in the Theory of Graphs*, pages 239–273. Academic Press, 1973.

[Ros98] K. Rose. Deterministic annealing for clustering, compression, classification, regression, and related optimization problems. *Proc. IEEE*, 80:2210–2239, November 1998.

[Row99] S. Roweis. Constrained hidden Markov models. In *NIPS-12*, 1999.

[RR01] A. Rao and K. Rose. Deterministically Annealed Design of Hidden Markov Model Speech Recognizers. *IEEE Trans. on Speech and Audio Proc.*, 9(2):111–126, February 2001.

[RS91] Neil Robertson and Paul D. Seymour. Graph minors – X: Obstructions to tree-decompositions. *J. Comb. Theory Series B*, 52:153–190, 1991.

[RS98] M. Ramoni and P. Sebastiani. Parameter Estimation in Bayesina networks from incomplete databases. *Intelligent Data Analysis Journal*, 2(1), 1998.

[RST96] Dana Ron, Yoram Singer, and Naftali Tishby. The power of amnesia: Learning probabilistic automata with variable memory length. *Machine Learning*, 25, 1996.

[RTL76] D. Rose, R. Tarjan, and G. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM J. on Computing*, 5:266–283, 1976.

[RW84] R. A. Redner and H. F. Walker. Mixture densities, maximum likelihood, and the EM algorithm. *SIAM Review*, 26:195–239, 1984.

[SAS94] R. Shachter, S. Andersen, and P. Szolovits. Global conditioning for probabilistic inference in belief networks. In *UAI*, 1994.

[SDLC93] David J. A. Spiegelhalter, Philip Dawid, Steffen L. Lauritzen, and Robert G. Cowell. Bayesian analysis in expert systems. *Statistical Science*, 8(3):219–283, 1993.

[SDW01] S. Soatto, G. Doretto, and Y.N. Wu. Dynamic textures. In *IEEE Conf. on Computer Vision and Pattern Recognition*, volume 2, pages 439–446, 2001.

[SGS00] P. Spirtes, C. Glymour, and R. Scheines. *Causation, Prediction, and Search*. MIT Press, 2000. 2nd edition.

[Sha86] R. Shachter. Evaluating influence diagrams. *Operations Research*, 33(6):871–882, 1986.

[Sha88] R. Shachter. Probabilistic inference and influence diagrams. *Oper. Res.*, 36(4):589–604, 1988.

[Sha98] R. Shachter. Bayes-ball: The rational pastime (for determining irrelevance and requisite information in belief networks and influence diagrams). In *UAI*, 1998.

[She92] P. P. Shenoy. Valuation-based systems for Bayesian Decision Analysis. *Operations Research*, 40:463–484, 1992.

[She97] P. Shenoy. Binary join trees for computing marginals in the shenoy-shafer architecture. *Intl. J. of Approximate Reasoning*, 17(2–3):239–263, 1997.

[SHJ97] P. Smyth, D. Heckerman, and M. I. Jordan. Probabilistic independence networks for hidden Markov probability models. *Neural Computation*, 9(2):227–269, 1997.

[SJ95] L. Saul and M. Jordan. Boltzmann chains and hidden Markov models. In *NIPS-7*, 1995.

[SJ99] L. Saul and M. Jordan. Mixed memory markov models: Decomposing complex stochastic processes as mixture of simpler ones. *Machine Learning*, 37(1):75–87, 1999.

[SJ02] H. Steck and T. Jaakkola. Unsupervised active learning in large domains. In *UAI*, 2002.

[SL90] D. J. Spiegelhalter and S. L. Lauritzen. Sequential updating of conditional probabilities on directed graphical structures. *Networks*, 20, 1990.

[SM80] A. Smith and U. Makov. Bayesian detection and estimation of jumps in linear systems. In O. Jacobs, M. Davis, M. Dempster, C. Harris, and P. Parks, editors, *Analysis and optimization of stochastic systems*. 1980.

[Smy96] P. Smyth. Clustering sequences with hidden Markov models. In *NIPS*, 1996.

[SNR00] J. Satagopan, M. Newton, and A. Raftery. Easy Estimation of Normalizing Constants and Bayes Factors from Posterior Simulation: Stabilizing the Harmonic Mean Estimator. Technical report, U. Washington, 2000.

[SP89] R. D. Shachter and M. A. Peot. Simulation approaches to general probabilistic inference on belief networks. In *UAI*, volume 5, 1989.

[SPS99] R.S. Sutton, D. Precup, and S. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211, 1999.

[Sri93] S. Srinivas. A generalization of the noisy-or model. In *UAI*, 1993.

[SS88] G. R. Shafer and P. P. Shenoy. Local computation in hypertrees. Technical Report 201, School of Business, Uni. Kansas, 1988.

[SS90a] G. R. Shafer and P. P. Shenoy. Probability propagation. *Annals of Mathematics and AI*, 2:327–352, 1990.

[SS90b] P. P. Shenoy and G. R. Shafer. Axioms for probability and belief-function propagation. In *UAI*, 1990.

[SS98] T. Schmidt and P. P. Shenoy. Some Improvements to the Shenoy-Shafer and Hugin Architectures for Computing Marginals. *Artificial Intelligence*, 102(2):323–333, 1998.

[SSC88] R. Smith, M. Self, and P. Cheeseman. Estimating uncertain spatial relationships in robotics. In Lemmer and Kanal, editors, *Uncertainty in Artificial Intelligence*, volume 2, pages 435–461. Elsevier, 1988.

[SSG99] R. Settimi, J. Smith, and A. Gargoum. Approximate learning in complex dynamuc Bayesian networks. In *UAI*, 1999.

[Ste00] B. Steinsky. Enumeration of labelled chain graphs and labelled essential directed acyclic graphs. Technical report, Univ. Salzburg, Austria, 2000.

[Sto94] A. Stolcke. *Bayesian Learning of Probabilistic Language Models*. PhD thesis, ICSI, UC Berkeley, 1994.

[SW96] Y. Singer and M. Warmuth. Training Algorithms for Hidden Markov Models Using Entropy Based Distance Function. In *NIPS-9*, 1996.

[SY99] Eugene Santos, Jr. and Joel D. Young. Probabilistic temporal networks: A unified framework for reasoning with time and uncertainty. *Intl. J. of Approximate Reasoning*, 20(3):191–216, 1999.

[TBF98] S. Thrun, W. Burgard, and D. Fox. A probabilistic approach to concurrent mapping and localization for mobile robots. *Machine Learning*, 31:29–53, 1998.

[TDW02] M. Takikawa, B. D'Ambrosio, and E. Wright. Real-time infernece with large-scale temporal bayes nets. In *UAI*, 2002. Submitted.

[Thi95] B. Thiesson. Accelerated quantification of bayesian networks with incomplete data. In *Proc. of the Int'l Conf. on Knowledge Discovery and Data Mining*, 1995.

[Thr98] S. Thrun. Bayesian landmark learning for mobile robot localization. *Machine Learning*, 33(1), 1998.

[Tip98] M. Tipping. Probabilistic visualization of high-dimensional binary data. In *NIPS*, 1998.

[TJ02] S. C. Tatikonda and M. I. Jordan. Loopy Belief Propagation and Gibbs Measures. In *UAI*, 2002.

[TK00] Simon Tong and Daphne Koller. Active learning for parameter estimation in bayesian networks. In *NIPS*, pages 647–653, 2000.

[TK01] S. Tong and D. Koller. Active learning for structure in Bayesian networks. In *IJCAI*, 2001.

[TLV01] S. Thrun, J. Langford, and V. Verma. Risk sensitive particle filters. In *NIPS*, 2001.

[TMH01] B. Thiesson, C. Meek, and D. Heckerman. Accelerating EM for Large Databases. Technical Report MSR-TR-99-31, Microsoft Research, 2001.

[TSM85] D. M. Titterington, A. F. M. Smith, and U. E. Makov. *Statistical analysis of finite mixture distributions*. Wiley, 1985.

[TW87] M. Tanner and W. Wong. The calculation of posterior distributions by data augmentation. *JASA*, 82(398):528–540, 1987.

[TY84] R. E. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM J. on Computing*, 13:566–579, 1984.

[UN98] N. Ueda and R. Nakano. Deterministic annealing EM algorithm. *Neural Networks*, 11:271–282, 1998.

[vdMDdFW00] R. van der Merwe, A. Doucet, N. de Freitas, and E. Wan. The unscented particle filter. In *NIPS-13*, 2000.

[VP90] T. Verma and J. Pearl. Equivalence and synthesis of causal models. In *UAI*, 1990.

[WBG92] M. P. Wellman, J. S. Breese, and R. P. Goldman. From knowledge bases to decision models. *The Knowledge Engineering Review*, 7(1):35–53, 1992.

[WdM01] E. A. Wan and R. Van der Merwe. The Unscented Kalman Filter. In S. Haykin, editor, *Kalman Filtering and Neural Networks*. Wiley, 2001.

[Wei00] Y. Weiss. Correctness of local probability propagation in graphical models with loops. *Neural Computation*, 12:1–41, 2000.

[Wei01] Y. Weiss. Comparing the mean field method and belief propagation for approximate inference in MRFs. In Saad and Opper, editors, *Advanced Mean Field Methods*. MIT Press, 2001.

[Wel90] M. P. Wellman. Fundamental concepts of qualitative probabilistic networks. *Artificial Intelligence*, 44(3):257–303, 1990.

[WF99] Y. Weiss and W. T. Freeman. Correctness of belief propagation in Gaussian graphical models of arbitrary topology. In *NIPS-12*, 1999.

[WH97] Mike West and Jeff Harrison. *Bayesian forecasting and dynamic models*. Springer, 1997.

[Wie00] W. Wiegerinck. Variational approximations between mean field theory and the junction tree algorithm. In *UAI*, pages 626–633, 2000.

[WJW01] M. Wainwright, T. Jaakkola, and A. Willsky. Tree-based reparameterization for approximate estimation on loopy graphs. In *NIPS-14*, 2001.

[WN01] E. A. Wan and A. T. Nelson. Dual EKF Methods. In S. Haykin, editor, *Kalman Filtering and Neural Networks*. Wiley, 2001.

[WT01] M. Welling and Y-W. Teh. Belief optimization for binary networks: a stable alternative to loopy belief propagation. In *UAI*, 2001.

[WT02] M. Welling and Y. W. Teh. Inference in Boltzmann Machines, Mean Field, TAP and Bethe Approximations. Technical report, U. Toronto, Dept. Comp. Sci., 2002.

[XJ96] L. Xu and M. I. Jordan. On convergence properties of the EM algorithm for Gaussian mixtures. *Neural Computation*, 8:129–151, 1996.

[XJKR02] E. P. Xing, M. I. Jordan, R. M. Karp, and S. J. Russell. A Hierarchical Bayesian Markovian Model for Motifs in Biopolymer Sequences. In *NIPS*, 2002. Submitted.

[XPB93] Y. Xiang, D. Poole, and M. P. Beddoes. Multiply sectioned Bayesian networks and junction forests for large knowledge-based systems. *Computational Intelligence*, 9(2):171–220, 1993.

[Yan81] M. Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM J. Alg. Discrete Methods*, 2:77–79, 1981.

[YFW00] J. Yedidia, W. T. Freeman, and Y. Weiss. Generalized belief propagation. In *NIPS-13*, 2000.

[YFW01] J. Yedidia, W. T. Freeman, and Y. Weiss. Understanding belief propagation and its generalizations. In *IJCAI*, 2001.

[YMHL95] J. York, D. Madigan, I. Heuch, and R. Lie. Birth defects registered by double sampling: a Bayesian approach incorporating covariates and model uncertainty. *Applied Statistics*, 44(2):227–242, 1995.

[Yui00] A. Yuille. A double loop algorithm to minimize the Bethe and Kikuchi free energies. In *NIPS*, 2000.

[Zha96] N. L. Zhang. Irrelevance and parameter learning in Bayesian networks. *Artificial Intelligence Journal*, 88:359–373, 1996.

[Zha98a] N. Zhang. Probabilistic Inference in Influence Diagrams. *Computational Intelligence*, 14(4):475–497, 1998.

[Zha98b] Nevin Zhang. Inference in Bayesian Networks: The Role of Context-Specific Independence. Technical Report HKUST-CS98-09, Dept. Comp. Sci., Hong Kong Univ., 1998.

[ZP96] N. Zhang and D. Poole. Exploiting causal independence in Bayesian network inference. *J. of AI Research*, pages 301–328, 1996.

[ZP99] N. L. Zhang and D. Poole. On the role of context-specific independence in probabilistic reasoning. In *IJCAI*, pages 1288–1293, 1999.

[ZP00] G. Zweig and M. Padmanabhan. Exact alpha-beta computation in logarithmic space with application to map word graph construction. In *Proc. Intl. Conf. Spoken Lang. Proc.*, 2000.

[ZR97] G. Zweig and S. Russell. Compositional modelling with DPNs. UC Berkeley CS Dept., 1997.

[Zwe96] G. Zweig. A forward-backward algorithm for inference in Bayesian networks and an empirical comparison with HMMs. Master's thesis, Dept. Comp. Sci., U.C. Berkeley, 1996.

[Zwe98] G. Zweig. *Speech Recognition with Dynamic Bayesian Networks*. PhD thesis, U.C. Berkeley, Dept. Comp. Sci., 1998.

[ZY97] N. L. Zhang and Li Yan. Independence of causal influence and clique tree propagation. *Intl. J. of Approximate Reasoning*, 19:335–349, 1997.