An introduction to graphical models

Kevin P. Murphy

10 May 2001

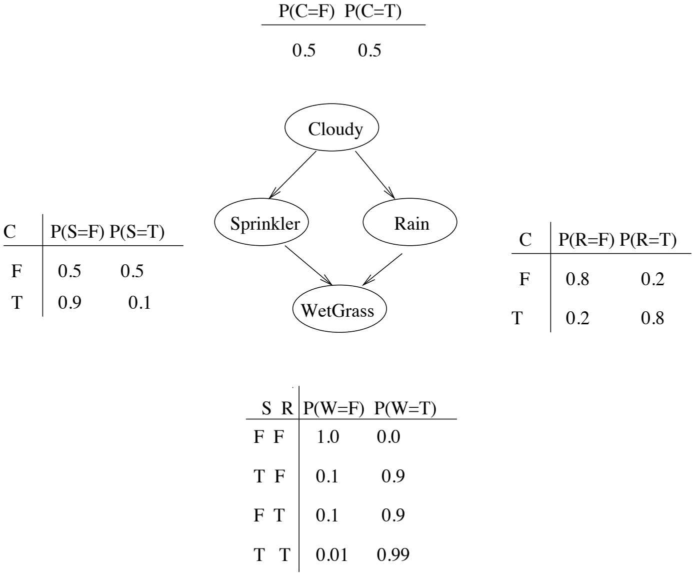
1 Introduction

The following quotation, from the Preface of [Jor99], provides a very concise introduction to graphical models.

> Graphical models are a marriage between probability theory and graph theory. They provide a natural tool for dealing with two problems that occur throughout applied mathematics and engineering – uncertainty and complexity – and in particular they are playing an increasingly important role in the design and analysis of machine learning algorithms. Fundamental to the idea of a graphical model is the notion of modularity – a complex system is built by combining simpler parts. Probability theory provides the glue whereby the parts are combined, ensuring that the system as a whole is consistent, and providing ways to interface models to data. The graph theoretic side of graphical models provides both an intuitively appealing interface by which humans can model highly-interacting sets of variables as well as a data structure that lends itself naturally to the design of efficient general-purpose algorithms.
>
> Many of the classical multivariate probabalistic systems studied in fields such as statistics, systems engineering, information theory, pattern recognition and statistical mechanics are special cases of the general graphical model formalism – examples include mixture models, factor analysis, hidden Markov models, Kalman filters and Ising models. The graphical model framework provides a way to view all of these systems as instances of a common underlying formalism. This view has many advantages – in particular, specialized techniques that have been developed in one field can be transferred between research communities and exploited more widely. Moreover, the graphical model formalism provides a natural framework for the design of new systems.

In this paper, we will flesh out this remark by discussing the following topics:

- Representation: how can a graphical model compactly represent a joint probability distribution?
- Inference: how can we efficiently infer the hidden states of a system, given partial and possibly noisy observations?
- Learning: how do we estimate the parameters and structure of the model?
- Decision theory: what happens when it is time to convert beliefs into actions?
- Applications: what has this machinery been used for?



P(C=F)	P(C=T)
0.5	0.5

C	P(S=F)	P(S=T)
F	0.5	0.5
T	0.9	0.1

C	P(R=F)	P(R=T)
F	0.8	0.2
T	0.2	0.8

S	R	P(W=F)	P(W=T)
F	F	1.0	0.0
T	F	0.1	0.9
F	T	0.1	0.9
T	T	0.01	0.99

Figure 1: A simple Bayesian network, adapted from [RN95].

2 Representation

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, as we will see below. 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), belief networks, generative models, causal models, etc. are more popular with the AI and machine learning communities. (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, as we will see below.) It is also possible to have a model with both directed and undirected arcs, which is called a chain graph.

2.1 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.[1] Hence directed cycles are disallowed.

Consider the example in Figure 1. Here, nodes represent binary random variables. We see that the event "grass is wet" (W=true) has two possible causes: either the water sprinker is on (S=true) or it is raining (R=true). The strength of this relationship is shown in the table below W; this is called W's conditional

[1] See the following books for a more detailed discussion of the use of Bayes nets for causal modelling: [Pea00, SGS00, GC99, Shi00].

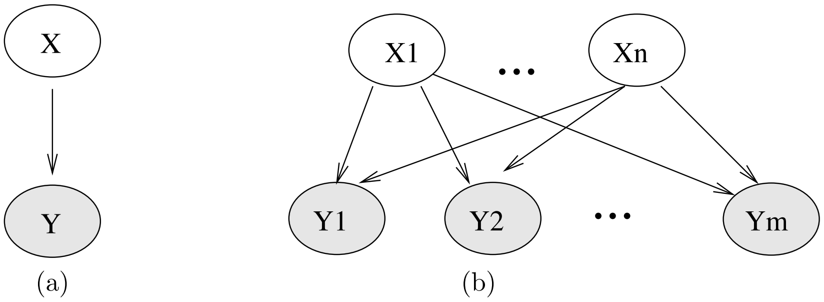


Figure 2: Factor analysis represented as a Bayesian network.

probability table (CPT). For example, we see that $P(W = true|S = true, R = false) = 0.9$ (second entry of second row), and hence, $P(W = false|S = true, R = false) = 1 - 0.9 = 0.1$, since each row must sum to one. Since the C node has no parents, its CPT specifies the prior probability that it is cloudy (in this case, 0.5).

The simplest statement of the conditional independence relationships encoded in a Bayesian network can be stated as follows: a node is independent of its ancestors given its parents, where the ancestor/parent relationship is with respect to some fixed topological ordering of the nodes. Let us see how we can use this fact to specify the joint distribution more compactly.

By the chain rule of probability, the joint probability of all the nodes in Figure 1 is

$$P(C, S, R, W) = P(C) \times P(S|C) \times P(R|C, S) \times P(W|C, S, R)$$

By using conditional independence relationships, we can rewrite this as

$$P(C, S, R, W) = P(C) \times P(S|C) \times P(R|C) \times P(W|S, R)$$

where we were allowed to simplify the third term because R is independent of S given its parent C (written $R \perp\!\!\!\perp S|C$), and the last term because $W \perp\!\!\!\perp C|S, R$.

We can see that the conditional independence relationships allow us to represent the joint more compactly. Here the savings are minimal, but in general, if we had n binary nodes, the full joint would require $O(2^n)$ parameters, but the factored form would only require $O(n2^k)$ parameters, where k is the maximum fan-in of a node.

We will now show how many popular statistical models can be represented as directed graphical models.

2.1.1 PCA, ICA, and all that

In the above example, the nodes represented discrete random variables. It is also possible that they represent continuous random variables. In this case, we must specify the conditional probability distribution (CPD) of a node given its parents in parametric form. A common example is a Gaussian. Consider the example in Figure 2(a): this represents the model $P(X, Y) = P(X)P(Y|X)$. Y is observed (hence it is shown shaded), but X is hidden. The goal will be to infer X given Y (i.e., to "invert the causal arrow"); we will discuss how to do this in Section 3. Another goal will be to estimate the parameters; we will discuss this in Section 4. For now, we just concentrate on the semantics of the model.

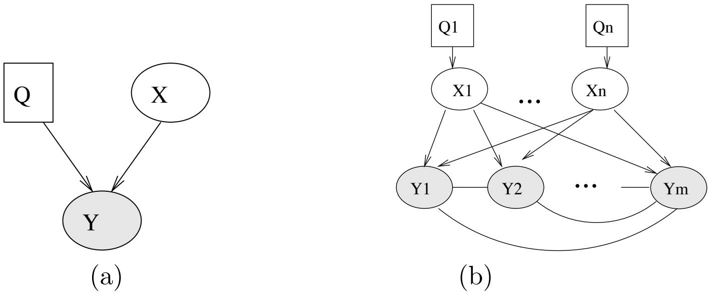


Figure 3: (a) Mixtures of factor analysis. (b) Independent factor analysis.

If we let the prior distribution on X be $P(X = x) = N(x; 0, I)$, and the conditional distribution on Y be $P(Y = y|X = x) = N(y; \mu + Wx, \Psi)$, where Ψ is a diagonal matrix, then this model is called factor analysis. (The notation $N(x; \mu, \Sigma)$ means the value of a Gaussian pdf with mean μ and covariance Σ evaluated at the point (vector) x.) We usually assume that $n \ll m$, where $X \in \mathbb{R}^n$ and $Y \in \mathbb{R}^m$, so the model is trying to explain a high dimensional vector as a linear combination of low dimensional features.

A common simplification is to assume isotropic noise, i.e., $\Psi = \sigma I$, for some scalar σ. In this case, the components of the vectors X and Y are uncorrelated, which we can explicitely represent using the model in Figure 2(b). It turns out that the maximum likelihood estimate of the factor loading matrix W in this case is given by the first m principle eigenvectors of the sample covariance matrix, with scalings determined by the eigenvalues and sigma. Classical PCA can be obtained by taking the $\sigma \to 0$ limit [TB99, RG99].

We can lift the restrictive assumption of linearity by modelling Y as a *mixture* of linear subspaces [TB99]. This model is is shown in Figure 3(a). Q is a discrete latent (hidden) variable, that indicates which of the subspaces to use to generate Y.[2] Mathematically, this can be written as

$$\begin{aligned} P(Q = i) &= \pi_i \\ P(X = x) &= N(x; 0, I) \\ P(Y = y|X = x, Q = i) &= N(y; \mu_i + W_i x, \Psi_i) \end{aligned}$$

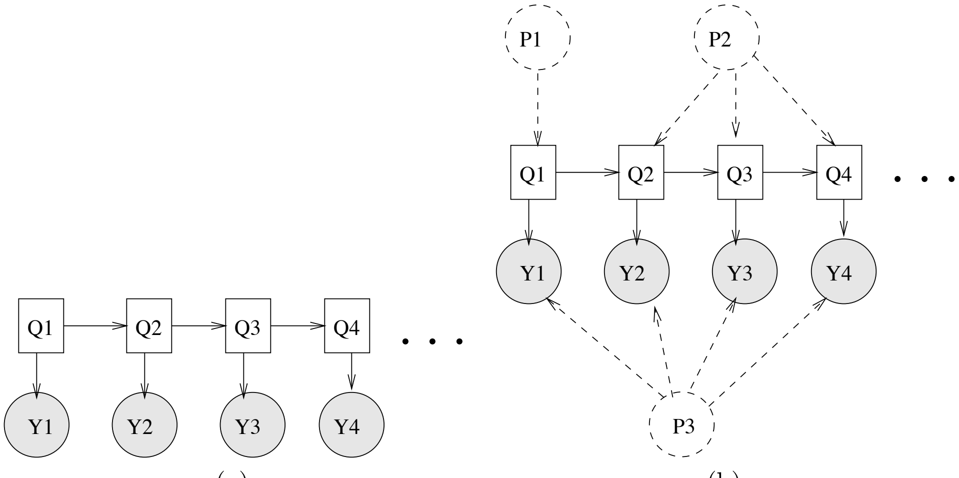
Similarly, we can lift the restrictive assumption of Gaussian noise by modelling the prior on X as a mixture of Gaussians. One way to do this, known as independent factor analysis (IFA) [Att98], is shown in Figure 3(b). (Technically, this is a chain graph: the undirected arcs between the components of Y indicate that they are correlated, i.e., that Ψ is not diagonal). This model is closely related to independent components analysis (ICA).

2.1.2 HMMs, Kalman filters, and all that

Consider the Bayesian network in Figure 4(a), which represents a hidden Markov model (HMM). This encodes the joint distribution

$$P(Q, Y) = P(Q_1)P(Y_1|Q_1)\prod_{t=2}^{4} P(Q_t|Q_{t-1})P(Y_t|Q_t)$$

[2] In this paper, we adopt the (non-standard) convention that square nodes represent discrete random variables, and circles represent continuous random variables. We also use the (standard) convention that shaded nodes are observed, and clear nodes are (usually) hidden.



(a) (b)

Figure 4: (a) A hidden Markov model (HMM) unrolled for 4 time slices. (b) An HMM where the parameters are explicitely shown (dotted circles). P1 represents π, P2 represents the transition matrix, and P3 represents the parameters for the observation model.

For a sequence of length T, we simply "unroll" the model for T time steps. In general, such a dynamic Bayesian network (DBN) can be specified by just drawing two time slices (this is sometimes called a 2TBN) — the structure (and parameters) are assumed to repeat.

The Markov property states that the future is independent of the past given the present, i.e., $Q_{t+1} \perp\!\!\!\perp Q_{t-1} | Q_t$. We can parameterize this Markov chain using a transition matrix, $M_{ij} = P(Q_{t+1} = j | Q_t = i)$, and a prior distribution, $\pi_i = P(Q_1 = i)$.

We have assumed that this is a homogeneous Markov chain, i.e., the parameters do not vary with time. This assumption can be made explicit by representing the parameters as nodes (see Figure 4(b)). If we think of these parameters as random variables (as in the Bayesian approach), parameter estimation becomes equivalent to inference. If we think of the parameters as fixed, but unknown, quantities, parameter estimation requires a separate learning procedure, to be discussed in Section 4. In the latter case, we typically do not represent the parameters in the graph; shared parameters (as in this example) are implemented by specifying that the corresponding CPDs are "tied".

An HMM is a *hidden* Markov model because we don't see the states of the Markov chain, Q_t, but just a function of them, namely Y_t. For example, if Y_t is a vector, we might define $P(Y_t = y | Q_t = i) = N(y; \mu_i, \Sigma_i)$. A richer model, widely used in speech recognition, is to model the output (conditioned on the hidden state) as a mixture of Gaussians. This is shown in Figure 5(d).

Some popular variations on the basic HMM theme are illustrated in Figure 5. (In the input-output model, the CPD $P(Q|U)$ could be a softmax function, or a neural network.) If we have software to handle inference and learning in general Bayesian networks, all of these models becomes trivial to implement.

A linear dynamical system (LDS) has the same topology as the Bayesian network in Figure 5(a), except that the hidden nodes have linear-Gaussian CPDs. Replacing Q_t with X_t, the model becomes

$$\begin{aligned} P(X_1 = x) &= N(x; x_0, V_0) \\ P(X_{t+1} = x_{t+1} | U_t = u, X_t = x) &= N(x_{t+1}; Ax + Bu, Q) \\ P(Y_t = y | X_t = x, U_t = u) &= N(y; Cx + Du, R) \end{aligned}$$

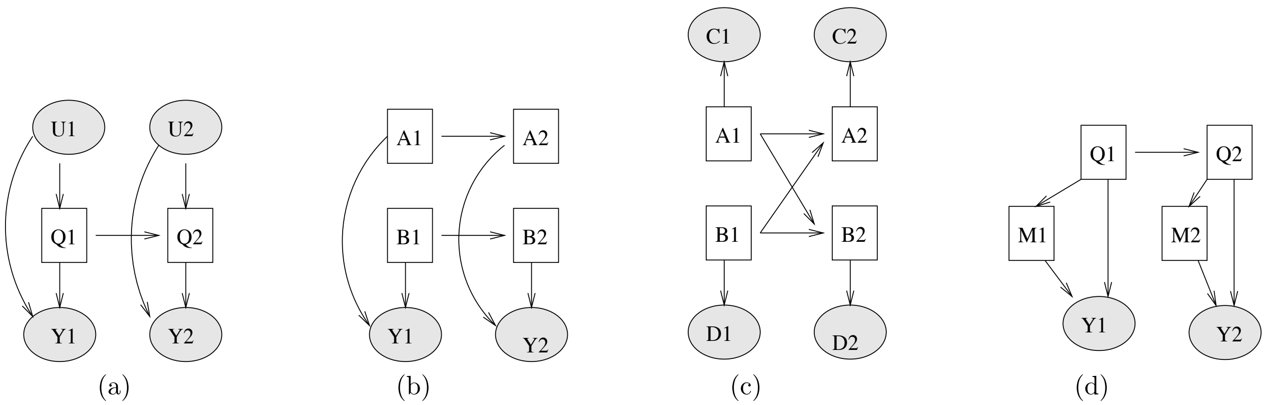


Figure 5: (a) An input-output HMM. (b) A factorial HMM. (c) A coupled HMM. (d) An HMM with mixture of Gaussians output.

The Kalman filter is an algorithm for computing $P(X_t|y_{1:t}, u_{1:t})$ online. The Rauch-Tung-Striebel (RTS) smoother is an algorithm for computing $P(X_t|y_{1:T}, u_{1:T})$ offline. Both of these algorithms turn out to be special cases of the general inference algorithms we will discuss in Section 3.2.

2.1.3 Summary

Figure 6 is a good summary of the relationship between various popular statistical models.

2.2 Undirected graphical models

The joint distribution of a Markov Random Field (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 in the graph, $\psi_C(x_C)$ is a potential function (a positive, but otherwise arbitrary, real-valued function) on the clique x_C, and Z is the normalization factor

$$Z = \sum_x \prod_{C \in \mathcal{C}} \psi_C(x_C).$$

Consider the example in Figure 7. 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 the Ising model, the graph is a grid, so the cliques are edges, and the X_i nodes are binary. The potentials $\Psi(x_i, x_j)$ for neighboring nodes i, j penalize differences: $\Psi(x_i, x_j) = e^{-J(x_i,x_j)}$, where $J(x_i, x_j) = 1$ if $x_i = x_j$ and $J(x_i, x_j) = -1$ otherwise.

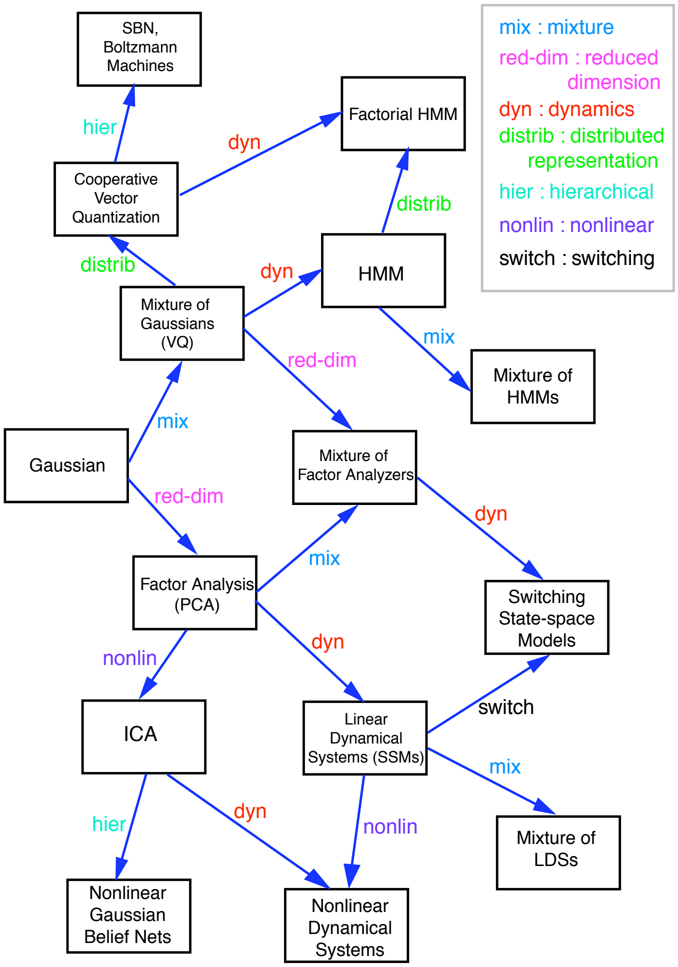


Figure 6: A generative model for generative models. This figure is from S. Roweis and Z. Ghahramani.

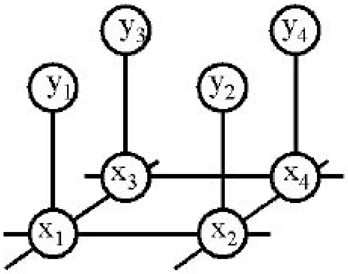


Figure 7: A Markov Random Field for low level vision. X_i is the hidden state of the world at grid position i, Y_i is the corresponding observed state. This figure is from [FP00].

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 7. 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".

3 Inference

The main goal of inference is to estimate the values of hidden nodes, given the values of the observed nodes. If we observe the "leaves" of a generative model, and try to infer the values of the hidden causes, this is called diagnosis, or bottom-up reasoning; if we observe the "roots" of a generative model, and try to predict the effects, this is called prediction, or top-down reasoning. Bayes nets can be used for both of these tasks (and others).

For example, consider the water sprinkler network in Figure 1, and suppose we observe the fact that the grass is wet, which we denote by $W = 1$ (1 represents true, 0 represents false). There are two possible causes for this: either it is raining, or the sprinkler is on. Which is more likely? We can use Bayes' rule to compute the posterior probability of each explanation. Bayes' rule states that

$$P(X|y) = \frac{P(y|X)P(X)}{P(y)}$$

where X are the hidden nodes and y is the observed evidence. (We follow standard practice and represent random *variables* by upper case letters, and sampled *values* by lower case letters.) In words, this formula becomes

$$\text{posterior} = \frac{\text{conditional likelihood } \times \text{prior}}{\text{likelihood}}$$

In the current example, we have

$$P(S = 1|W = 1) = \frac{P(S = 1, W = 1)}{P(W = 1)} = \frac{\sum_{c,r} P(C = c, S = 1, R = r, W = 1)}{P(W = 1)} = \frac{0.2781}{0.6471} = 0.430$$

and

$$P(R = 1|W = 1) = \frac{P(R = 1, W = 1)}{P(W = 1)} = \frac{\sum_{c,s} P(C = c, S = s, R = 1, W = 1)}{P(W = 1)} = \frac{0.4581}{0.6471} = 0.708$$

where

$$P(W = 1) = \sum_{c,s,r} P(C = c, S = s, R = r, W = 1) = 0.6471$$

is a normalizing constant, equal to the probability (likelihood) of the data. So we see that it is more likely that the grass is wet because it is raining than because of the sprinkler: the likelihood ratio is 0.708/0.430 = 1.647. (Note that this computation has considered both scenarios, in which it is cloudy and not cloudy.)

In general, computing posterior esimates using Bayes' rule is computationally intractable. One way to see this is just to consider the normalizing constant, Z: in general, this involves a sum over an exponential number of terms. (For continuous random variables, the sum becomes an integral, which, except for certain notable cases like Gaussians, is not analytically tractable.) We will now see how we can use the conditional independence assumptions encoded in the graph to speed up exact inference.

3.1 Variable elimination

Consider the problem of computing the normalizing constant $P(W=1)$ for the water sprinkler model. Using the factored representation of the joint implied by the graph, we may write this as

$$\begin{aligned}\Pr(W=w) &= \sum_c \sum_s \sum_r \Pr(C=c, S=s, R=r, W=w) \\ &= \sum_c \sum_s \sum_r \Pr(C=c) \times \Pr(S=s|C=c) \times \Pr(R=r|C=c) \times \Pr(W=w|S=s, R=r)\end{aligned}$$

The key idea of the variable elimination algorithm (and many others) is to "push" the sums in as far as possible, thus:

$$P(W=w) = \sum_c \Pr(C=c) \sum_s \Pr(S=s|C=c) \sum_r \Pr(R=r|C=c) \times \Pr(W=w|S=s, R=r)$$

When we perform the innermost sum

$$\Pr(W=w) = \sum_c \Pr(C=c) \sum_s \Pr(S=s|C=c) \times T1(c, w, s)$$

we create a new term which does not depend on the term summed out:

$$T1(c, w, s) = \sum_r \Pr(R=r|C=c) \times \Pr(W=w|S=s, R=r)$$

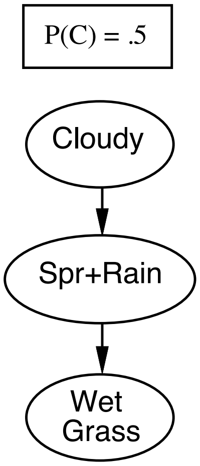
Performing the second sum we get

$$\Pr(W=w) = \sum_c \Pr(C=c) \times T2(c, w)$$

where

$$T2(c, w) = \sum_s \Pr(S=s|C=c) \times T1(c, w, s)$$

The principle of distributing sums over products can be generalized greatly to apply to any commutative semi-ring. This forms the basis of many common algorithms, such as Viterbi decoding and the Fast Fourier Transform [AM00]. The amount of work we perform is bounded by the size of the largest term that we create. Choosing a summation (elimination) ordering to minimize this is NP-hard, although greedy algorithms work well in practice.



C	P(S+R=x) TT	TF	FT	FF
T	.08	.02	.72	.18
F	.40	.10	.40	.10

S+R	P(W)
T T	.99
T F	.90
F T	.90
F F	.00

Figure 8: A clustered version of Figure 1. From [RN95].

3.2 Dynamic programming

If we wish to compute several marginals at the same time, we can use dynamic programming to avoid the redundant computation that would be involved if we used variable elimination repeatedly. If the underlying undirected graph of the BN is acyclic (i.e., a tree), we can use a local message passing algorithm due to Pearl [Pea88, PS91]. This is a generalization of the well-known forwards-backwards algorithm for HMMs (chains) [Rab89].

If the BN has (undirected) cycles or "loops", as in the water sprinkler example, local message passing algorithms run the risk of double counting, and may not converge. For example, the information from S and R flowing into W is not independent, because it came from a common cause, C. Analogous problems plague undirected graphical models with loops.

Assuming for now that one is interested in exact inference, the most common approach is therefore to convert the graphical model into a tree, by clustering nodes together (see Figure 8). The algorithms used to convert a (possibly directed) graph to a tree are somewhat complicated, involving various graph-theoretic operations (see [CDLS99, HD96] for details). After creating this tree of clusters, one can once again run a local message passing algorithm. The message passing scheme could be Pearl's algorithm, but it is more common to use a variant designed for undirected models, called the junction tree algorithm [CDLS99, HD96].

3.3 The need for approximate inference

The running time of these exact algorithms is exponential in the size of the largest cluster (assuming all hidden nodes are discrete); this size is called the induced width of the graph, and minimizing it is NP-hard. For many graphs, such as grids or directed graphs which contain nodes with high fan-in, the induced width is large, so it is necessary to use approximate inference.

Approximate inference is also often necessary even for graphs with low induced width (e.g., chains and trees) if some of the nodes represent continuous random variables; in many cases, the corresponding integrals needed for implementing Bayes' rule cannot be performed in closed form. For instance, this is nearly always the case with Bayesian approaches, where one is trying to estimate a distribution over a continuous-valued parameter node (see e.g., Figure 4(b)).

3.4 Summary of approximate inference methods

Here we provide a brief summary of some popular approximate inference methods.

- Sampling (Monte Carlo) methods. The simplest kind is importance sampling, where we draw random samples x from $P(X)$, the (unconditional) distribution on the hidden variables, and then weight the samples by their likelihood, $P(y|x)$, where y is the evidence. A more efficient approach in high dimensions is called Markov Chain Monte Carlo (MCMC), and includes as special cases Gibbs sampling and the Metropolis-Hastings algorithm. See e.g., [GRS96] for an introduction.
- Variational methods. The simplest example is the mean-field approximation, which 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. We then iteratively update these variational parameters so as to minimize the cross-entropy (KL distance) between the approximate and true probability distributions. Updating the variational parameters becomes a proxy for inference. The mean-field approximation produces a lower bound on the likelihood. More sophisticated methods are possible, which give tighter lower (and upper) bounds. See [JGJS98] for a tutorial.
- Loopy belief propagation. This entails applying Pearl's algorithm (see Section 3.2) to the original graph, even if it has loops (undirected cycles). This algorithm was inspired by the outstanding empirical success of turbo codes, which have been shown [MMC98] to be an instance of this algorithm. Since then, further empirical work (e.g., [MWJ99, FP00]) has shown that the algorithm works well in other contexts, such as low-level vision. There has also been a lot of recent theoretical work analysing this algorithm [WF99, Wei00, FW00], and figuring out the connections to other methods from statistical mechanics [YFW00].

4 Learning

Learning might refer to the structure (topology) of the model, or the parameters, or both. (Parameter estimation is sometimes called "systems identification".) Another important distinction is whether all the variables are observed, or whether some of them are hidden. This gives rise to the following four-way contigency table for classifying learning methods (EM stands for Expectation Maximization).

	Observability	
Structure	Full	Partial
Known	Closed form	EM
Unknown	Local search	Structural EM

A third distinction is whether the goal is to find a single "best" model/ set of parameters (point estimation), or to return a posterior *distribution* over models/ parameters (Bayesian estimation). In the case of known structure, Bayesian parameter estimation is the same as inference, where hidden nodes represent the parameters; this was briefly discussed in Section 2.1.2. Bayesian structure learning will be discussed in Section 4.3.2.

Before going into details, we mention that learning BNs is a huge subject, and this paper only touches the tip of the iceberg. For further details, see the review papers [Bun94, Bun96, Hec98].

4.1 Known structure, full observability

4.1.1 Directed graphical models

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 valyes which maximize the likelihood of the training data, which contains M cases (assumed to be independent). The normalized log-likelihood of the training set $D = \{D_1, \ldots, D_M\}$ is a sum of terms, one for each node:

$$L = \frac{1}{M} \log \prod_{m=1}^{M} \Pr(D_m|G) = \frac{1}{M} \sum_{i=1}^{n} \sum_{m=1}^{M} \log P(X_i|Pa(X_i), D_m)$$

where $Pa(X_i)$ are the parents of X_i. We see that the log-likelihood scoring function *decomposes* according to the structure of the graph, and hence we can maximize the contribution to the log-likelihood of each node independently. (It is simple to modify this to handle tied (shared) parameters.)

If there are a small number of training cases compared to the number of parameters, we can 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. In practice, this is not much harder computationally, as we will see below.

For example, consider estimating the CPT for the W node in the water sprinkler example. If we have a set of training data, we can just count the number of times the grass is wet when it is raining and the sprinler is on, $\#(W = 1, S = 1, R = 1)$, the number of times the grass is wet when it is raining and the sprinkler is off, $\#(W = 1, S = 0, R = 1)$, etc. Given these counts (which are the sufficient statistics for a multinomial distribution), we can find the MLE of the CPT as follows:

$$\hat{P}_{ML}(W = w|S = s, R = r) = \frac{\#(W = w, S = s, R = r)}{\#(S = s, R = r)}$$

where the denominator is $\#(S = s, R = r) = \#(W = 0, S = s, R = r) + \#(W = 1, S = s, R = r)$. Thus "learning" just amounts to counting (in the case of multinomial distributions).

If a particular event is not seen in the training set, it will be assigned a probability of 0. To avoid this, we can use a Dirichlet prior; this just amounts to adding "pseudo counts" to the empirical counts. For example, using a uniform Dirichlet prior, the MAP estimate for the W node becomes

$$\hat{P}_{MAP}(W = w|S = s, R = r) = \frac{\#(W = w, S = s, R = r) + 1}{\#(S = s, R = r) + 2}$$

For Gaussian nodes, the MLE of the mean and covariance are the sample mean and covariance, and the MLE of the weight matrix is the least squares solution to the normal equations. For other kinds of CPDs (e.g., neural networks), more complex procedures are necessary for computing ML/MAP parameter estimates.

4.1.2 Undirected graphical models

The parameters of an undirected graphical model are the clique potentials. Maximum likelihood estimates of these can be computed using iterative proportional fitting (IPF) [JP95].

4.2 Known structure, partial observability

When some of the nodes are hidden, we can use the EM (Expectation Maximization) algorithm to find a (locally) optimal MLE. The basic idea behind EM is that, if we knew the values of all the nodes, learning (the M step) would be easy: we just use the techniques from Section 4.1. So in the E step, we compute the expected values of all the nodes using an inference algorithm, and then treat these expected values as though they were observed.

For example, in the case of the W node, we replace the observed counts of the events with the number of times we expect to see each event:

$$\hat{P}(W=w|S=s,R=r) = E\#(W=w,S=s,R=r)/E\#(S=s,R=r)$$

where $E\#(e)$ is the expected number of times event e occurs in the whole training set, given the current guess of the parameters. These expected counts can be computed as follows

$$E\#(e) = E\sum_m I(e|D_m) = \sum_m P(e|D_m)$$

where $I(e|D_m)$ is an indicator function which is 1 if event e occurs in training case m, and 0 otherwise. Given the expected counts, we maximize the parameters, and then recompute the expected counts, etc. This iterative procedure is guaranteed to converge to a local maximum of the likelihood surface. (Note: the Baum-Welch algorithm used for training HMMs [Rab89] is identical to the EM algorithm.)

It is also possible to do gradient ascent on the likelihood surface (the gradient expression also involves the expected sufficient statistics), but EM is usually faster (since it uses the natural gradient) and simpler (since it has no step size parameter and takes care of parameter constraints automatically[3]).

Whether we use EM or gradient ascent, inference becomes a subroutine which is called by the learning procedure; hence fast inference algorithms are crucial for learning.

4.3 Unknown structure, full observability

The maximum likelihood structure would be the complete graph, since this has the greatest number of parameters, and hence can achieve the highest likelihood. A well-principled way to avoid this kind of overfitting is to put a prior on models. By Bayes' rule, the MAP model is the one that maximizes

$$\Pr(G|D) = \frac{\Pr(D|G)\Pr(G)}{\Pr(D)}$$

[3] In the case of multinomials, the constraint is that all the parameters must be in $[0,1]$, and each row of the CPT must sum to one; in the case of Gaussians, the constraint is that the covariance matrix must be positive semi-definite; etc.

Taking logs, we find

$$L = \log \Pr(G|D) = \log \Pr(D|G) + \log \Pr(G) + c$$

where $c = -\log P(D)$ is a constant independent of G. 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.

In fact, it is not necessary to explicitly penalize complex structures through the structural prior. The marginal likelihood,

$$P(D|G) = \int P(D|G, \theta)P(\theta|G)d\theta,$$

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. (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]).

If we assume all the parameters are independent, the marginal likelihood decomposes into a product of local terms:

$$P(D|G) = \prod_{i=1}^{n} \int P(X_i|Pa(X_i), \theta_i)P(\theta_i)d\theta_i$$

In some cases (e.g., if the priors on the parameters are conjugate), each of these integrals can be performed in closed form, so the marginal likelihood can be computed very efficiently. (If not, one typically has to resort to sampling methods.)

4.3.1 Search algorithms

Having defined a scoring function, we still need a way to search for the highest scoring graph. Unfortunately, the number of DAG's on n variables is super-exponential in n. (There is no closed form formula for this, but to give you an idea, there are 543 DAGs on 4 nodes, and $O(10^{18})$ DAGs on 10 nodes.) The usual approach is therefore to use local search algorithms (e.g., greedy hill climbing, possibly with multiple restarts) or perhaps branch and bound techniques, to search through the space of graphs. (For directed graphical models, we must ensure the graphs have no directed cycles; for undirected graphical models, we must ensure the graphs are decomposable.)

The fact that the scoring function is a product of local terms makes local search more efficient, since to compute the relative score of two models that differ by only a few arcs (i.e., neighbors in the space), it is only necessary to compute the terms which they do not have in common — the other terms cancel when taking the ratio.

4.3.2 Bayesian methods

In the Bayesian approach, the goal is to compute the posterior $P(G|D)$. Since this is super-exponentially large, a more realistic goal is to return a set of graphs sampled from this distribution. The standard approach is to use an MCMC search procedure; see [Mur01] for a review.

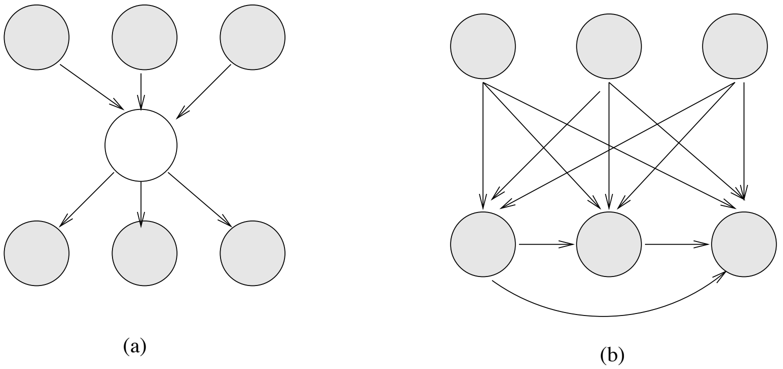


Figure 9: (a) A BN with a hidden variable H. (b) The simplest network that can capture the same distribution without using a hidden variable (created using arc reversal and node elimination). If H is binary and the other nodes are trinary, and we assume full CPTs, the first network has 45 independent parameters, and the second has 708.

4.4 Unknown structure, partial observability

Finally, we come to the hardest case of all, where the structure is unknown and there are hidden variables and/or missing data. The problem is that the marginal likelihood is intractable, requiring that we sum out all the latent variables Z as well as integrate out all the parameters θ:

$$P(X|G) = \sum_Z \int P(X, Z|G, \theta) P(\theta|G) d\theta$$

This score does not decompose into a product of local terms. One approach is to use a Laplace approximation to the posterior of the parameters [CH97]. After some algebra [Hec98], this leads to the following expression:

$$\log \Pr(D|G) \approx \log \Pr(D|G, \hat{\theta}_G) - \frac{d}{2} \log M$$

where M is the number of samples, $\hat{\theta}_G$ is the ML estimate of the parameters (computed using EM) and d is the dimension of the model. (In the fully observable case, the dimension of a model is the number of free parameters. In a model with hidden variables, it might be less than this.) 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.)

Although the BIC score decomposes, local search algorithms (e.g., hill climbing) are still expensive, because we need to run EM at each step to compute $\hat{\theta}$. An alternative approach is to do the local search steps inside of the M step of EM — this is called Structural EM, and provably converges to a local maximum of the BIC score [Fri97].

4.4.1 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 9.

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, dense clique-like graphs (such as that in Figure 9(b)) suggest that a hidden node should be added [ELFK00].

5 Decision making under uncertainty

It is sometimes said that "Decision Theory = Probability Theory + Utility Theory" [DW91, RN95]. We have outlined above how we can model joint probability distributions in a compact way by using sparse graphs to reflect conditional independence relationships. It is also possible to decompose multi-attribute utility functions in a similar way. Let the global utility be a sum of local utilities, $U = \sum_{i=1}^{n} U_i$. We create a node for each U_i term, which has as parents all the attributes (random variables) on which it depends; typically, the utility nodes will also have action (control) nodes as parents, since the utility depends both on the state of the world and the action we perform. The resulting graph is called an influence diagram. We can then use algorithms, similar to the inference algorithms discussed in Section 3, to compute the optimal (sequence of) action(s) to perform so as to maximimize expected utility [CDLS99]. (See [KLS01] for a recent application of this to multi-person game theory.)

In *sequential* decision theory, the agent (decision maker) is assumed to be interacting with the environment which is modelled as a dynamical system (see Figure 5(a)). If this dynamical system is linear with Gaussian noise, and the utility function is negative quadratic loss[4], then techniques from control theory can be used to compute the optimal policy, i.e., mapping from percepts to actions. If the system is non-linear, the standard approach is to locally linearize the system.

Linear dynamical systems (LDSs) enjoy the separation property, which states that the optimal behavior can be obtained by first doing state estimation (i.e., infer the hidden states), and then using the expected value of the hidden states as input to a regular LQG (linear quadratic Gaussian) controller. In general, however, the separation property does not hold. For example, consider controlling a mobile robot. The optimal action should take into account the robot's uncertainty about the state of the world (e.g., its location), and not just use the mode of the posterior as if it were the true state of the world. The latter strategy (which is called a certainty-equivalent controller) would never perform information-gathering moves. In other words, the robot might not choose to look before it leapt, no matter how uncertain it was.

In general, finding good controllers for non-linear, partially observed systems, usually with unknown parameters, is extremely challenging. One approach that shows some promise is reinforcement learning. Most of the work has been on systems which are fully observed (e.g., [SB98]), but there has been some work on partially observed systems (e.g., [KLC98]). Recent policy search methods (e.g., [NJ00]) show particular promise.

6 Applications

Special cases of Bayes nets were independently invented by many different communities many years ago, e.g., genetics (linkage analysis), speech recognition (HMMs), tracking (Kalman fitering), data compression (density estimation), channel coding (turbocodes), etc.

The general framework was developed by Pearl [Pea88] and various European researchers [JLO90, CDLS99], who used it to make probabilistic expert systems. Many of these systems were used for medical diagnosis.

[4] For example, consider a missile tracking an airplane, where the goal is to minimize the squared distance between itself and the target.

The same kind of diagnosis technology has since been widely adopted by Microsoft, e.g., the Answer Wizard of Office 95, the Office Assistant (the bouncy paperclip guy) of Office 97, and over 30 technical support troubleshooters. In fact, Microsoft now offers a Bayesian network API for Windows developers.

Another interesting fielded application is the Vista system, developed by Eric Horvitz. The Vista system is a decision-theoretic system that has been used at NASA Mission Control Center in Houston for several years. The system uses Bayesian networks to interpret live telemetry and provides advice on the likelihood of alternative failures of the space shuttle's propulsion systems. It also considers time criticality and recommends actions of the highest expected utility. The Vista system also employs decision-theoretic methods for controlling the display of information to dynamically identify the most important information to highlight. Horvitz has gone on to attempt to apply similar technology to Microsoft products, e.g., the Lumiere user interface project [HBH+98].

7 Acknowledgements

I would like to thank Charles Twardy for comments on this manuscript.

References

[AM00] S. M. Aji and R. J. McEliece. The generalized distributive law. *IEEE Trans. Info. Theory*, 46(2):325–343, March 2000.

[Att98] H. Attias. Independent factor analysis. *Neural Computation*, 1998. In Press.

[Bun94] W. L. Buntine. Operations for learning with graphical models. *J. of AI Research*, pages 159–225, 1994.

[Bun96] W. Buntine. A guide to the literature on learning probabilistic networks from data. *IEEE Trans. on Knowledge and Data Engineering*, 8(2), 1996.

[CDLS99] R. G. Cowell, A. P. Dawid, S. L. Lauritzen, and D. J. Spiegelhalter. *Probabilistic Networks and Expert Systems.* Springer, 1999.

[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.

[DW91] T. Dean and M. Wellman. *Planning and Control.* Morgan Kaufmann, 1991.

[ELFK00] G. Elidan, N. Lotner, N. Friedman, and D. Koller. Discovering hidden variables: A structure-based approach. In *Advances in Neural Info. Proc. Systems*, 2000.

[FP00] W. T. Freeman and E. C. Pasztor. Markov networks for super-resolution. In *Proc. 34th Annual Conf. on Information Sciences and Systems (CISS 2000)*, 2000.

[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 *Proc. of the Conf. on Uncertainty in AI*, 1997.

[FW00] W. Freeman and Y. Weiss. On the fixed points of the max-product algorithm. *IEEE Trans. on Info. Theory*, 2000. To appear.

[GC99] C. Glymour and G. Cooper, editors. *Computation, Causation and Discovery.* MIT Press, 1999.

[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.

[GRS96] W. Gilks, S. Richardson, and D. Spiegelhalter. *Markov Chain Monte Carlo in Practice.* Chapman and Hall, 1996.

[HBH$^+$98] E. Horvitz, J. Breese, D. Heckerman, D. Hovel, and K. Rommelse. The lumiere project: Bayesian user modeling for inferring the goals and needs of software users. In *Proc. of the Conf. on Uncertainty in AI*, 1998.

[HD96] C. Huang and A. Darwiche. Inference in belief networks: A procedural guide. *Intl. J. Approx. Reasoning*, 15(3):225–263, 1996.

[Hec98] D. Heckerman. A tutorial on learning with Bayesian networks. In M. Jordan, editor, *Learning in Graphical Models.* MIT Press, 1998.

[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.

[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.

[Jor99] M. I. Jordan, editor. *Learning in Graphical Models.* MIT Press, 1999.

[JP95] R. Jirousek and S. Preucil. On the effective implementation of the iterative proportional fitting procedure. *Computational Statistics & Data Analysis*, 19:177–189, 1995.

[KLC98] L. P. Kaelbling, M. Littman, and A. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101, 1998.

[KLS01] M. Kearns, M. Littman, and S. Singh. Graphical models for game theory. In *Proc. of the Conf. on Uncertainty in AI*, 2001.

[Mac95] D. MacKay. Probable networks and plausible predictions — a review of practical Bayesian methods for supervised neural networks. *Network*, 1995.

[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 Selected Areas in Comm.*, 16(2):140–152, 1998.

[Mur01] K. Murphy. Learning Bayes net structure from sparse data sets. Technical report, Comp. Sci. Div., UC Berkeley, 2001.

[MWJ99] K. Murphy, Y. Weiss, and M. Jordan. Loopy belief propagation for approximate inference: an empirical study. In *Proc. of the Conf. on Uncertainty in AI*, 1999.

[NJ00] A. Ng and M. Jordan. PEGASUS: A policy search method for large MDPs and POMDPs. In *Proc. of the Conf. on Uncertainty in AI*, 2000.

[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.

[PS91] M. Peot and R. Shachter. Fusion and propogation with multiple observations in belief networks. *Artificial Intelligence*, 48:299–318, 1991.

[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.

[RG99] S. Roweis and Z. Ghahramani. A Unifying Review of Linear Gaussian Models. *Neural Computation*, 11(2), 1999.

[RN95] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Englewood Cliffs, NJ, 1995.

[SB98] R. Sutton and A. Barto. *Reinforcment Learning: An Introduction*. MIT Press, 1998.

[SGS00] P. Spirtes, C. Glymour, and R. Scheines. *Causation, Prediction, and Search*. MIT Press, 2000. 2nd edition.

[Shi00] B. Shipley. *Cause and Correlation in Biology: A User's Guide to Path Analysis, Structural Equations and Causal Inference*. Cambridge, 2000.

[TB99] M. Tipping and C. Bishop. Mixtures of probabilistic principal component analyzers. *Neural Computation*, 11(2):443–482, 1999.

[Wei00] Y. Weiss. Correctness of local probability propagation in graphical models with loops. *Neural Computation*, 12:1–41, 2000.

[WF99] Y. Weiss and W. T. Freeman. Correctness of belief propagation in Gaussian graphical models of arbitrary topology. In *NIPS-12*, 1999.

[YFW00] J. Yedidia, W. T. Freeman, and Y. Weiss. Generalized belief propagation. In *NIPS-13*, 2000.