The Bayes Net Toolbox for Matlab

Kevin P. Murphy
Department of Computer Science
University of California, Berkeley
Berkeley, CA, 94720-1776

Abstract

The Bayes Net Toolbox (BNT) is an open-source Matlab package for directed graphical models. BNT supports many kinds of nodes (probability distributions), exact and approximate inference, parameter and structure learning, and static and dynamic models. BNT is widely used in teaching and research: the web page has received over 28,000 hits since May 2000. In this paper, we discuss a broad spectrum of issues related to graphical models (directed and undirected), and describe, at a high-level, how BNT was designed to cope with them all. We also compare BNT to other software packages for graphical models, and discuss the nascent OpenBayes effort.

1 Introduction

Over the last decade or so, graphical models have emerged as a powerful unifying formalism for many probabilistic models which are widely used in statistics, machine learning and engineering, ranging from mixture models to hidden Markov models (HMMs), from factor analysis (PCA) to Kalman filters. The reason for this is well-described in the following quotation [Jor99]:

> 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.

What the field has lacked is a corresponding general purpose software package. This article describes one attempt to build such a package, called the Bayes Net Toolbox (BNT). First we will describe some of the issues that arise in the representation, inference and learning of graphical models. Then we will describe what

approaches existing software packages adopt towards these issues. Finally, we describe BNT, which overcomes some of the shortcomings of existing packages.

The reader is assumed to already be familiar with the basics of graphical models: see for example the books [Jor02, CDLS99, Edw00, Jor99, Fre98].

2   Representation

Graphical models come in two main "flavors": directed and undirected. It is also possible to combine directed and undirected graphs. We will discuss each in turn.

2.1   Directed graphical models

Directed acyclic graph (DAG) models, also known as Bayesian or belief networks, are popular in the AI community, partly because they lend themselves to a causal interpretation [Pea00], which makes their structure easy to design by hand (as in an expert system). DAG models are also useful for modelling temporal data and dynamical systems, because they can encode "the arrow of time". (Such models are sometimes called DBNs, or dynamic Bayesian networks.)

DAG models are also popular in the Bayesian statistics community, since parameters can be represented explicitly as nodes (random variables), and endowed with distributions (priors). The resulting graph not only provides a concise specification of the model, but can also be exploited computationally, e.g., by Gibbs sampling. This is what the well-known BUGS[1] package does.

A DAG model which includes decision and utility nodes, as well as chance nodes, is known as an influence (decision) diagram, and can be used for optimal decision making.

There is also a kind of graphical model, called a dependency network [HCM$^+$00], which allows directed cycles. It can be useful for data visualization, but does not always define a unique joint distribution: see [HCM$^+$00] for details.

2.1.1   Parameterization of directed models

A directed model (both Bayes nets and dependency nets) can be pararameterized by specifying all the local Conditional Probability Distributions (CPDs), i.e., the distributions $P(X_i|\text{Pa}_i)$, where X_i represents node i and Pa_i are its parents. Most packages assume all the nodes represent discrete (categorical) random variables, and further, that the CPDs are multinomials, which can be represented as tables.

Tabular CPDs are simple to represent, learn and use for inference (see Section 3.1.1), but have the disadvantage of requiring a number of parameters that is exponential in the number of parents. Other representations, which only require a linear number of parameters, have been proposed, including noisy-OR [Pea88] and its generalizations [MH97], and the logistic (sigmoid) function [Nea92].

Decision trees [BFGK96] can be used to represent CPDs with a variable (data-dependent) number of parameters; they are also useful for variable (parent) selection inside a structure learning algorithm (see Section 4.2). Regression trees are similar, but are used when X_i is a continuous random variable. Feedforward neural networks (multi-layer perceptrons, or MLPs), conditional linear Gaussian (CLG) and generalized linear models (GLMs) can also be used to model CPDs for continuous nodes.

---

[1]`www.mrc-bsu.cam.ac.uk/bugs`

For Bayesian modelling, we need a richer variety of CPDs, e.g., Dirichlet priors for multinomial parameters, Wishart (Gamma) priors for variance/ precision parameters, as well as Gaussian priors for weight matrices. We can also use non-parametric distributions.

It is easy to let users define exotic CPDs. The difficulties arise when we wish to do inference on the corresponding model, and/or when we try to learn the parameters from data. We will discuss these issues below.

2.2   Undirected graphical models

Undirected graphical models, also known as Markov networks, are common in the physics and computer vision communities. For instance, Ising models and Markov Random Fields (MRFs) are just grid-structured Markov networks. In the statistics community, undirected models are often used to model multiway contingency tables, in which case they are called (hierarchical) log-linear models [Edw00].

2.2.1   Parameterization of undirected models

The parameters of a Markov network are the clique potentials. For instance, in a 2D grid, these correspond to the edge potentials $\phi(x_i, x_j)$ for neighboring pixels i, j. If all variables are discrete (categorical), these potentials can be represented as tables. However, as in the DAG case, this may require a large number of parameters: in a clique of n binary nodes, specifying ϕ requires 2^n parameters. Parameter tying (e.g., assuming ϕ is the same for all edges in the grid) can amelioriate this problem in some domains such as image processing. In other domains, such as text processing, maximum-entropy methods[2] are popular. In this approach, one defines a potential on nodes $x = x_1, \ldots, x_n$ using a Gibbs (exponential family) distribution on a set of features, $f_1(x), \ldots, f_k(x)$:

$$\phi(x) = \frac{1}{Z} \exp\left(\sum_{i=1}^{k} f_i(x)\right) \tag{1}$$

The domains (support) of the features implicitely define the graph structure. For example, suppose $n = 3$, $k = 2$, f_1 is a function of x_1, x_2, and f_2 is a function of x_2, x_3; then the graph has 3 nodes, and looks like $x_1 - x_2 - x_3$.

2.3   Mixed directed/ undirected graphical models

It is possible to combine directed and undirected graphs into what is called a chain graph [CDLS99]. A common example of this is in image processing, where the hidden nodes are connected in an undirected 2D grid, but each hidden node has a child which contains the observed value of that pixel (see Figure 1). As far as inference is concerned, the link from hidden pixel to observed pixel can be represented either as a directed or undirected arc; however, the directed arc representation is preferred when it comes to learning (see Section 4.1).

The factor graph formalism [KFL01] is a very general way of using graph structure to represent global models (not necessarily probabilistic) in terms of local terms, or factors.

---

[2]See `www-2.cs.cmu.edu/~aberger/maxent.html` for an excellent collection of tutorials and papers on maxent.
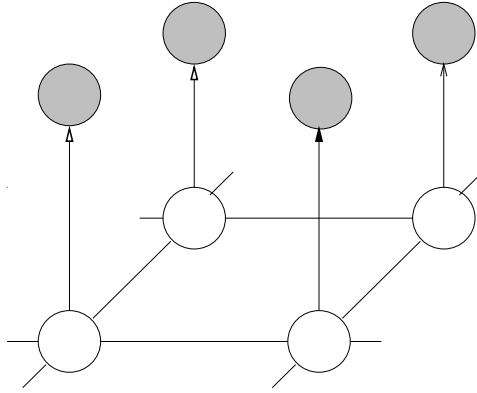
Figure 1: A chain graph for image processing. Each shaded node is an observed pixel, and is caused by its hidden parent (clear); the hidden causes are correlated with each other, as modelled by a Markov Random Field with pairwise potentials.

It is possible to inter-convert all of these representations [YFW01], although sometimes information is "lost" from the graph structure in the process (this information will be implicitly represented in the parameters). This can affect the computational complexity of inference, as well as the intrepretability of the model.

3 Inference

By inference, we mean computing

$$P(X_i|X_j) \propto \sum_{k \neq i,j} P(X_i, X_j, X_k)$$

where X_j represents a set of observed variables, X_i represents a set of hidden variables whose value we are interested in estimating, and X_k are the irrelevant (nuisance) hidden variables. For instance, X_i might represent whether a patient has disease i, and X_j might represent the observed symptoms. Or X_i might represent an unknown parameter, and X_j all the data.

There are two main kinds of inference: exact and approximate. We will discuss both below.

3.1 Exact inference

Exact inference (in the sense of having a closed form solution) is only possible in a very limited set of cases, most notably when all hidden nodes are discrete, or when all nodes (hidden and observed) have linear Gaussian distributions (in which case the network is just a sparse parameterization of a joint multivariate Gaussian [SK89, RG99]). Expert systems and hidden Markov models (HMMs) fall in the former category, while factor analysis and Kalman filters fall in the latter.

There are two main kinds of exact inference algorithms: those that only work on DAG models, and those that work on directed and undirected graphs.

The DAG-only inference algorithms exploit the chain-rule decomposition of the joint,

$$P(X) = P(X_1)P(X_2|X_1)P(X_3|X_1, X_2)\dots$$

and essentially "push sums inside products" to marginalize out the irrelevant hidden nodes efficiently [LD94, Dec96, AM00]; this is called the variable elimination algorithm. The result of the computation is a *single* marginal, $P(X_i|X_j)$.

The general inference algorithms are defined in terms of message passing on a tree. If the original graph has undirected cycles (loops), it must be converted to a so-called junction tree using triangulation [Kja90] or cutset conditioning [Dra95]. The tree may be directed or undirected, the messages may be passed in parallel or sequentially[3], and the computation of the messages may or may not involve a division operation. For instance, Pearl's algorithm [Pea88] was formulated for directed trees without division; the Hugin/JLO algorithm [JLO90] was formulated for undirected trees with division; and belief propagation [YFW01] was formulated for undirected networks without division. All of these algorithms are essentially equivalent.

The advantage of the message passing algorithms arises when one is interested in computing all marginals simultaneously (which is necessary e.g., for learning): they use dynamic programming to avoid the redundant computation that would be involved in calling variable elimination n times, once for each variable. However, variable elimination is somewhat easier to implement, and enables certain optimizations which exploit knowledge of the specific query.

3.1.1 Potentials

To implement any inference algorithm that works in terms of sums and products (e.g., variable elimination or message passing), it is necessary to represent each local distribution as an object that supports the operations (methods) of summation/integration, multiplication and optionally division. We will call such an object a "potential", which is just a non-negative function of the variables in its domain.

If all the random variables in the domain are discrete, we can represent a potential as a multidimensional array (a table). If all the random variables in the domain are jointly Gaussian, we can represent the potential as a multivariate Gaussian: we simply store the mean, covariance and a scale factor. If some variables are discrete and some are Gaussian, we can represent the potential as a conditional Gaussian (CG), which is a table of Gaussians, instead of a table of scalars. Finally, if some variables are discrete random variables and some are discrete utility variables, we can represent the potential as a pair of tables; this is useful for influence diagrams. All of these types of potentials are described in [CDLS99]; see also [Mur98b].

CG potentials can represent finite mixtures of Gaussians. Unfortunately, this representation is not closed. That is, if we have a potential ϕ with domain (D, C), where D is a discrete variable with k possible values, and C is a continuous variable, then $\sum_D \phi(D, C)$ is still a mixture of k Gaussians: it hasn't got any smaller. Hence repeated applications of sums and products will cause the size of the representation to blow up. One approximation is to use "weak marginalisation" [Lau92], which reduces a mixture of Gaussians to a single Gaussian using moment matching. (The implementation of this in [Lau92] is numerically unstable, and has been improved in [LJ99].) See also [Min01].

Undirected models are already parameterized in terms of potentials on cliques, so no conversion is necessary. Directed models are parameterized in terms of CPDs, but we can simply define the potentials as $\phi(X_i, \mathrm{Pa}_i) = P(X_i|\mathrm{Pa}_i)$. However, this is only possible for some kinds of CPDs: see Section 7.2.1 for details.

---

[3]If messages are passed sequentially, the scheduling usually uses two passes, often called collect/distribute or forwards/backwards [PS91].

The level of abstraction afforded by potentials allows us to reuse the same code for many models. For instance, by simply replacing discrete potentials with Gaussian potentials, the same code can be used either to implement the forwards-backwards algorithm for HMMs or the RTS smoother for linear dynamical systems [Mur98a].

3.2 Approximate inference

Even in the cases where exact inference is mathematically possible, it might not be computationally feasible: the cost of inference depends on the treewidth W of the graph (i.e., the size of the maximal clique in the corresponding optimally triangulated graph). In particular, if all (hidden) nodes are discrete and binary-valued, inference takes $O(2^W)$ time. (For jointly Gaussian distributions, inference always takes at most $O(n^3)$, where n is the number of nodes, regardless of W; however, sometimes (e.g., in image processing applications), n is very large.)

For trees (graphs with no undirected cycles), the treewidth is constant (namely the maximal fan-in (number of parents) of any node in the graph), and inference takes $O(n)$ time. However, for other graphs, especially those with repeating structure, such as grids, the treewidth grows with the number of variables (e.g., in an $n = m \times m$ grid, the treewidth is $O(m) = O(\sqrt{n})$), and exact inference is often infeasible.

There are at least two reasons why one might need approximate inference: either because computing the exact solution would take too long, or because there is no closed-form (analytic) solution. (Exact inference is known to be NP-hard in general.) The former kind of complexity arises from certain kinds of graph structures, the latter from certain kinds of distributions. (In general, almost all distributions on continuous random variables yield intractable posteriors, the fully observed conjugate-exponential case being a notable exception.) Below we list some techniques that can be used to tackle both kinds of intractability.

- Sampling (Monte Carlo) methods. The simplest kind is importance sampling, where we draw random samples x from the prior $P(X)$, 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 Monte Carlo Markov Chain (MCMC), which allows us to draw samples from the posterior, $P(X|y)$, even when we can't compute the normalizing constant, $\int_x P(y|x)P(x)$. MCMC includes as special cases Gibbs sampling and the Metropolis-Hasting algorithm (see e.g., [Nea93, GRS96, Mac98]). MCMC is the dominant method for approximate inference in the Bayesian statistics community.

- 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, and iteratively update these 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. Recently this technique has been extended to do approximate Bayesian inference, using a technique called Variational Bayes [GB00].

- Belief propagation (BP). This entails applying the message passing algorithm to the original graph, even if it has loops (undirected cycles). Originally this was believed to be unsound, but the outstanding empirical success of turbocodes [BGT93], which have been shown to be using the BP algorithm [MMC98], led to a lot of theoretical analysis, which has shown how BP is closely related to variational methods [YFW01, SO01]. Recently this technique has been extended to do approximate Bayesian inference, using a technique called Expectation Propagation [Min01].

4 Learning

There are two main kinds of learning: parameter learning (also called model fitting) and structure learning (also called model selection). We will discuss each in turn.

4.1 Parameter learning

If we adopt a Bayesian approach, parameters are treated just like any other random variable, and hence learning is the same as inference. If we adopt a frequentist approach, the goal of learning is to find a point estimate of the parameters, either maximum likelihood (ML) or maximum a posteriori (MAP).

The learning problem is much harder when we have partial observability, i.e., missing values and/or hidden (latent) variables in the model. In this case, the exact parameter posterior is generally multimodal, so we must settle for finding a locally optimal solution. We start by considering the fully observed case, where we can compute the global optimum.

4.1.1 Fully observed case

If the model is a DAG, the CPDs can be estimated independently of each other (the problem is said to fully decompose). If the CPDs are in the exponential family (e.g., Gaussians or tables), we can compute the ML estimates in closed form. (If we use conjugate priors, we can also compute the MAP estimate in closed form.) For more complex CPDs (e.g., trees, GLMs or neural networks), local hill-climbing methods may be necessary.

If the model is undirected, the ML estimates of the clique potentials can be estimated in closed form iff the graph is triangulated (decomposable) and the potentials are tabular (fully parameterized). If the graph is not triangulated, but the potentials are still tabular, we can use Iterative Proportional Fitting (IPF). Finally, if the potentials are represented in terms of features (no matter what the graph structure), we must use the generalized iterative scaling (GIS) algorithm [DR72], which requires that the features sum to a constant, or the improved iterative scaling (IIS) algorithm [PPL97], which makes no assumptions about the features. Usually one uses Monte Carlo sampling to compute the expectations needed by these scaling algorithms, but if the graph implied by the features has small treewidth, one can use exact inference, which is much faster [BJ01]. Alternatively, one could use deterministic approximation schemes such as belief propagation [TW01].

4.1.2 Partially observed case

If there are missing values or latent variables, we can use the EM algorithm to find a locally optimal ML/MAP estimate [Lau95]. The E step requires calling an inference

routine (exact or approximate) to compute the expected sufficient statistics, and the M step is similar to the fully observed case. (If the M step only takes a step in the right direction of parameter space, instead of finding the optimum, it is called the generalized EM algorithm. One can also imagine doing a "partial E step" [NH98].) Other methods for handling partial observability, such as "bound and collapse" [RS97] or gradient-based methods [BKRK97], are of course possible. The advantages of EM compared to gradient methods are its simplicity, its lack of step size parameters, and the fact that it deals with constraints automatically. It can be combined with gradient methods for increased speed [JJ93].

4.2 Structure learning

If we adopt a Bayesian approach, structure learning means returning a posterior distribution over all possible graphs, or at least computing the expected value of functions (e.g., which indicate the presence of certain features, such as edges) with respect to this distribution. If we adopt a non-Bayesian approach, structure learning means finding the single "best" model. "Best" can either mean the one that satisfies all (and only) the conditional independencies observed in the data (the constraint-based approach), or one that maximizes some scoring function, such as penalized likelihood (e.g., MDL/BIC) or marginal likelihood.

To find the best scoring model, we must in principle search through the space of all models (since the space is discrete, we cannot use gradient methods). For Bayes nets, the model may either be a DAG or a PDAG (partially directed acyclic graph), which represents a whole class of Markov equivalent DAGs. Given observational data alone, it is not possible to distinguish between members of the same equivalence class. However, it is simple to modify the scoring function to exploit interventional data: simply don't update the parameters of nodes that have been set by intervention [CY99]. This enables one to learn causal models from data, which is useful in such domains as bioinformatics.

An alternative to searching in graph/edge space is to search in feature space, and then use the Gibbs-exponential distribution (Equation 1) to define the model [PPL97, BJ01]. Features provide a much "finer granularity" than edges, and often lead to models that perform better in terms of density estimation or predictive power. However, they cannot be interpreted causally.

Structure learning is a very large subject: see e.g., [HGC94, Mur01] for more details.

5 Software issues

Aside from the various theoretical issues we have discussed above, there are various practical issues which make a big difference to the success of a software package. We mention a few below.

- Is the package free? A lot of packages are free for academic use only. Some commerical packages have free versions, which are restricted in various ways, e.g., they limit the model size.

- Is source code available? What language is it written in?

- Is the system easy to extend? For instance, can one add new kinds of CPDs or inference algorithms?

- Is an API available? Without an API (application program interface), the package cannot be integrated into other applications, i.e., it must be used as a standalone program.

- Is a GUI available? What systems does it run on? e.g., Java should run on any computer, but many GUIs only work with Microsoft Windows.

6   Comparison of existing software packages

In Figure 6, we review all of the existing software packages that we are aware of (as of September 2001) with respect to some of the issues we have mentioned above. What is not mentioned in this table (due to lack of space) is that nearly all of these packages using exact inference, either junction tree, variable elimination, or Pearl's algorithm. (Many of them are also limited to discrete random variables.) This severly restricts their applicability. Some packages (e.g., BUGS, CABeN and Hydra) use sampling, which is more flexible. Unfortunately, these sampling-based programs fail to exploit tractable substructure (Rao-Blackwellization), and are therefore usually too slow for really big problems in domains such as speech, vision and text.

As we will see in the next section, one of the strengths of BNT is the wide range of inference algorithms, both exact and approximate, that it makes available to the user. This enables one to use the same package for many different kinds of model. Furthermore, any of these inference algorithms can be "plugged in" to the learning algorithms in a modular way.

7   Design of BNT

We now discuss the design of BNT with respect to the above desiderata.

7.1   Representation

BNT was originally only designed for Bayes nets (DAG models) — hence the name. It was recently extended to handle influence diagrams. A Bayes net is represented as a structure, which contains the graph (represented as an adjacency matrix), as well as the CPDs (represented as a list of objects), and a few other pieces of information. (For DBNs, we store the intra-slice topology and the inter-slice topology.)

BNT supports a variety of CPD types, each of which is its own class: see Figure 3. (It does not yet support Bayesian models, so the parameters are "hidden" inside each CPD object.) Each class is required to implement a set of methods, depending on which inference and learning algorithms are to be used. This will be explained below.

7.2   Inference

One of the biggest strengths of BNT is that it offers a variety of inference algorithms, each of which makes different tradeoffs between accuracy, generality, simplicity, speed, etc. Some of the algorithms (e.g., brute force enumeration of a joint) are designed for pedagogical/ debugging purposes only. In some cases, BNT offers multiple implementations of the same algorithm (e.g., a Matlab version and a C version). All engines have the same API, and hence can be easily interchanged. (For instance, any of them can be used in the E step of EM.)

Name	Src	API	Cts	GUI	θ	G	$\sim$	U	UG	Free
Analytica	N	N	Y	W	N	N	N	Y	N	R
Bassist	C++	Y	Y	N	Y	N	Y	N	N	Y
Bayda	Java	-	Y	Y	Y	N	N	N	N	Y
BayesBuilder	N	N	N	Y	N	N	Y	N	N	R
B. Discoverer	N	N	D	Y	Y	Y	N	N	N	R
B-course	N	N	D	Y	Y	Y	N	N	N	Y
Bayonnet	Java	-	Y	Y	Y	N	N	N	N	Y
BN power constr.	N	W	N	Y	Y	CI	N	N	N	Y
BN Toolbox	Matlab	-	Y	N	Y	Y	Y	Y	N	Y
BN Toolkit	VBasic	-	N	Y	N	Y	N	N	N	Y
BucketElim	C++	-	N	N	N	N	N	N	N	Y
BUGS	N	N	Y	W	Y	N	Y	N	N	Y
Business Nav. 5	N	N	D	Y	Y	Y	N	N	N	R
CABeN	C	Y	N	N	N	N	Y	N	N	Y
CoCo+Xlisp	C/lisp	-	N	Y	Y	CI	N	N	Only	Y
CIspace	Java	N	N	Y	N	N	N	N	N	Y
Ergo	N	N	N	Y	N	N	N	N	N	R
FLoUE/BIFtoN	Java	-	N	N	N	N	N	N	N	Y
GDAGsim	C	-	Only	N	N	N	Y	N	N	Y
GMRFsim	C	-	Only	N	N	N	Y	N	Only	Y
Genie/Smile	N	WU	N	W	N	N	Y	Y	N	Y
Hugin Expert	N	Y	Y	W	Y	CI	Y	Y	Y	R
Ideal	Lisp	-	N	Y	N	N	N	Y	N	Y
Java Bayes	Java	-	N	Y	N	N	N	Y	N	Y
MIM	N	N	Y	Y	Y	Y	N	N	Y	R
MSBNx	N	Y	N	W	N	N	N	Y	N	R
Netica	N	WUM	Y	W	Y	N	Y	Y	N	R
Pulcinella	Lisp	-	N	Y	N	N	N	N	N	Y
RISO	Java	-	Y	Y	N	N	N	N	N	Y
Tetrad	N	N	Y	N	Y	CI	N	N	Y	Y
Web Weaver	Java	-	N	Y	N	N	N	Y	N	Y
WinMine	N	N	Y	Y	Y	Y	N	N	Y	R
XBAIES 2.0	N	N	N	Y	Y	N	N	Y	Y	Y

Figure 2: Comparison of some software packages for graphical models. The columns have the following meanings. Src: is source code available? API: is an application program interface available, or must the program be used as a standalone blackbox? (- = source is included, so no API is needed; W=Windows, U=Unix, M=Mac) Cts: does the package support continuous random variables (D means it discretizes them). GUI: does the package have a GUI? θ: does the package learn parameters? G: does the package learn structure? (N = no, Y = uses search and score, CI = uses conditional independence tests) $\sim$: does the package support sampling? U: does the package support utility/action nodes (i.e., decision diagrams)? UG: does the package support undirected graphs? Free: is the package freely available? (Y=yes, N=no, R=restricted, which means this is a commerical product, but is free for non-commercial use, and/or the free version has limited functionality). See www.cs.berkeley.edu/∼murphyk/Bayes/bnsoft.html for an online version of this table, complete with URLs and additional information.

generic

discrete   gaussian   root

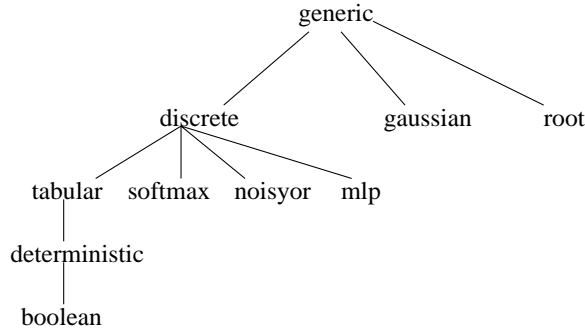tabular   softmax   noisyor   mlp

deterministic

boolean

Figure 3: Class hierarchy of the CPDs supported by BNT. A root node is one which has no parameters — it represents an observed exogenous input, and is used to define conditional density models. MLP stands for multilayer perceptron (i.e., feedforward neural network). Currently this can only be used for classification (i.e., discrete nodes), but could easily be extended to regression (i.e., continuous nodes). Softmax is essentially a special case of an MLP with no hidden layers. We use netlab to implement learning for both MLPs and softmax. Strictly speaking, deterministic and boolean CPDs are not implemented as subtypes of the tabular CPD class — their constructors actually create tabular CPD objects. Thus they merely provide some "syntactic sugar" to the tabular CPD class.

Each algorithm is represented as an "inference engine" object. The inference engines differ in many ways, including

- Works for all topologies or makes restrictions?

- Works for all CPD types or makes restrictions?

- Exact or approximate inference?

In terms of topology, most engines handle any kind of DAG. The exceptions are as follows: `belprop_fg` does belief propagation on factor graphs (FG), `pearl` only works on polytrees, and `quickscore` only works on BN2O models such as QMR.

In terms of CPD types: sampling algorithms can essentially handle any kind of node (distribution); algorithms that use sum-products (e.g., variable elimination, junction tree and belief propagation) must convert the CPD to a potential, which imposes some restrictions: see Section 7.2.1.

Finally, most algorithms are designed to give the exact answer. The exceptions are belief propagation (which is only exact if applied to trees and in some other cases) and sampling (which is only "exact" in the limit of an infinite number of samples).

The situation is summarized in Figure 4.

7.2.1   Potentials

To implement any inference algorithm that works in terms of sums and products (e.g., variable elimination or message passing), it is necessary to represent each CPD $P(X_i|\text{Pa}_i)$ as a potential, as discussed in Section 3.1.1. BNT supports four kinds of potentials, each of which is its own class: discrete `dpot`, Gaussian (represented either in moment form, `mpot`, or canonical form, `cpot`), conditional Gaussian `cgpot`, and utility `upot`.

Engine	Exact?	CPD types	Topology
belprop	approx	D,G,CG	DAG
belprop_fg	approx	D,G,CG	FG
cond_gauss	exact	CG	DAG
enumerative	exact	D	DAG
gaussian	exact	G	DAG
global_joint	exact	D,G,CG	DAG
jtree	exact	D,G,CG	DAG
jtree_C	exact	D	DAG
likelihood_weighting	approx	any	DAG
loopy_pearl	approx	D	DAG
pearl	exact	D	polytree
quickscore	exact	noisy-or	BN2O
var_elim	exact	D,G,CG	DAG

Figure 4: Inference engines for static models supported by BNT. D means all hidden nodes must be discrete; G means all hidden nodes must have linear-Gaussian CPDs; CG means all hidden nodes must have conditional linear Gaussian CPDs. For details of the algorithms, please consult the online documentation.

If X_i and Pa_i are both discrete, we can enumerate all their values and represent $\phi(X_i, \mathrm{Pa}_i)$ as a Conditional Probability Table. (This is what the BNT function CPD_to_CPT does.) However, this might destroy any special structure in the CPD. For instance, if $P(X_i|\mathrm{Pa}_i)$ is a noisy-OR distribution, requiring only $k = |\mathrm{Pa}_i|$ parameters, the resulting CPT has $O(2^k)$ entries. This will not affect the sample complexity (i.e., number of training cases needed to learn), but will affect the computational complexity (i.e., amount of time needed to do inference). One way to avoid this is to explicitly (i.e., graphically) represent the conditional independence assumptions made by the noisy-OR distribution [RD98].

If $P(X_i|\mathrm{Pa}_i)$ is a conditional linear Gaussian distribution, we can represent $\phi(X_i, \mathrm{Pa}_i)$ as a CG potential. In general, however, such mixed discrete/continuous distributions cannot be represented as potentials. For example, if X_i is discrete but Pa_i is continuous, the CPD (e.g., logistic or softmax) cannot be represented as a CG potential. However, if Pa_i is *observed* to have value u, we can evaluate $P(X_i = j|\mathrm{Pa}_i = u)$ as a function of j, and represent the result as a table. (This is what the BNT function CPD_to_table does.) This suggests that we should convert CPDs to potentials *after* we have seen the evidence, an idea first proposed in [Mur99]. This is the approach adopted by BNT, which lets it apply exact algorithms to a much greater range of models (e.g., mixtures of experts, input-output HMMs) than most other software packages. (Unfortunately this technique cannot be used for the stable CG algorithm of [LJ99].)

Another advantage of the BNT approach to handling evidence is as follows. When a discrete variable is observed, it can only take on one possible value — namely, the observed value. Rather than representing the potential as a sparse table with many zeroes, we can simply allocate the observed variable a dimension of size 1; the resulting table will be much smaller, and hence faster to manipulate. This is much simpler than approaches such as zero compression [JA90, HD96].

7.2.2 Belief propagation

Belief propagation (BP) can be implemented by converting all CPDs to potentials as above, and then using the standard sum-product operations [KFL01, YFW01]. This is simple, but slow. There are two reasons it is slow: practical and theoretical. The practical reason is that the overhead of manipulating objects in Matlab is quite high, and this becomes a real problem with BP, since it is an iterative algorithm. The theoretical reason is that any structure in the CPDs is lost, as mentioned above. Hence some CPDs (notably noisy-OR) implement special purpose methods for computing the messages needed by BP (these methods are called `compute_lambda_msg` and `compute_pi`). Currently this is only exploited by `loopy_pearl_inf_engine`.

7.2.3 Sampling

To implement likelihood weighting (importance sampling), we need a way of sampling from $P(X_i|\text{Pa}_i^*)$, where the $*$ means the parents are observed. A generic way to do this is to convert the partially observed CPD to a potential (see Section 7.2.1) and sample from $\phi(X_i)$. Individual CPD classes may implement more efficient methods.

To implement Gibbs sampling, we need a way of sampling from the Markov blanket of each node:

$$P(X_i|M_i^*) = P(X_i|\text{Pa}_i^*) \prod_{j \in \text{Children}(i)} P(X_j^*|\text{Pa}_j^*)$$

This is not yet implemented, partly because it would be very slow in Matlab, and partly because the BUGS system already does such a good job of this. (One plan for the future is to add an interface between BNT and BUGS.)

7.3 Learning

BNT supports both parameter and structure learning, but currently only for DAG models.

7.3.1 Parameter learning

The parameter estimation routines in BNT can be classified into 4 types, depending on whether the goal is to compute a full (Bayesian) posterior over the parameters or just a point estimate (e.g., Maximum Likelihood or Maximum A Posteriori), and whether all the variables are fully observed or there is missing data/ hidden variables (partial observability). The routines are listed below.

	Full	Partial
Point	`learn_params`	`learn_params_em`
Bayes	`bayes_update_params`	not yet supported

Currently only the `tabular_CPD` class supports Bayesian updating (which is useful, amongst other things, for sequential/on-line learning) using Dirichlet priors. Also, the MAP estimate can only be computed for tabular CPDs. (Priors are especially important for multinomials to avoid problems with sparse data.)

7.3.2 Structure learning

The structure learning routines in BNT can be classified into 4 types, analogously to the parameter learning case. Algorithms that have not yet been implemented are indicated in parentheses. The notation will be explained below.

	Full	Partial
Point	K2, IC/PC, (MI), (hill climb)	(structural EM), (hill climb + $\tilde{S}$)
Bayes	MCMC	(MCMC + $\tilde{S}$)

By $\tilde{S}$, we mean an approximation to the Bayesian scoring function, such as those mentioned in [CH97]. The structural EM algorithm is described in [Fri97, Fri98]. The K2 algorithm [CH92] assumes a total ordering of the nodes is given. In principle one can search over this ordering [FK00]; this is more efficient than searching in the (much larger) space of graphs.

Unlike all the others, the IC/PC algorithm [SGS01] is a constraint-based algorithm that does not need to do search: it starts with a fully connected graph, and then removes edges as determined by conditional independence tests applied to the data; directions (arrows) are then added to the arcs up to Markov equivalence. The disadvantage of independence tests such as χ^2 is the need to specify a signifance threshold; this can be avoided by using a Bayesian criterion [MT01]. Alternatively, one can use a mutual information (MI) criterion [CBL97, MT99]. Both options are supported by BNT.

7.4 Software issues

We now summarize some other relevant attributes of BNT.

- Free? Yes (GNU GPL license).

- Source code available? Yes (Matlab and some C).

- Easy to extend? Yes. Adding new CPDs or inference engines is as simple as creating a new class.

- API available? Yes. Matlab can be called from a variety of other languages.

- GUI available? Coming soon (see Section 8.3).

8 Past, present and future of BNT

8.1 Past

BNT was started while I was a summer intern at Compaq (formerly DEC) Cambridge Research Labs (CRL) in 1997. I was working with Jim Rehg, who wanted to apply Bayes nets to computer vision problems. At the time there were very few software packages available, and none had all the features that we needed, such as learning and the ability to handle continuous random variables and dynamical systems.

The CRL system was written in C++ and was very limited. In fact, it could not handle learning, continuous random variables, or dynamical systems! When I returned to Berkeley I decided to rewrite the whole system in Matlab.

8.1.1 Why Matlab?

I chose Matlab because of the ease with which it can handle Gaussian random variables (e.g., the Kalman filter can be implemented in about 10 lines!). This proved to be a very good decision: although Matlab is often perceived as merely a package for numerical linear algebra and data visualization, with the release of version 5 in the mid '90s, Matlab became a fully-fledged programming language, with flexible data-structures such as cell arrays and objects. (This is what prevents BNT from running on Octave, an open-source version of Matlab 4 which lacks such features.)

We can identify various pros and cons of Matlab. On the positive side,

- Matlab is a scripting language, which makes it very suitable for rapid proto-typing. In addition, it provides a good debugger and profiler.

- Matlab has excellent numerical algorithms (e.g., SVD).

- Matlab has excellent data visualization/ plotting routines.

- Matlab has many other excellent toolboxes (neural nets, optimization, etc.). For instance, BNT uses netlab[4] to learn MLP and softmax CPDs.

- Matlab code is high level and easy to read.

- Matlab is widely used in education (at least in engineering departments).

On the negative side,

- Matlab is slow (it is an untyped interpreted language), and the compiler is not very effective.

- The Matlab object system is much less advanced than that of, say, Java or C++.

- The commercial Matlab license costs thousands of dollars (although the student edition is only $100 in the US).

- Matlab has no pointers. In particular, it only supports pass by value or pass by constant reference, so passing large arguments to functions can result in a lot of copying (e.g., if a function updates a single pixel in an image, the whole image must be copied in and out of the function!).

A comparison between Matlab and other interactive scientific languages, such as Mathematica, Splus and Gauss, is available online.[5]

8.2 Present

BNT has proved to be very popular: the web site has an average of 500 hits a week, and it is widely used throughout the world for teaching and research, in fields ranging from statistics to biology, from economics to mechanical engineering. Rockwell has even developed a commercial product[6] around it.

---

[4]`www.ncrg.aston.ac.uk/netlab`

[5]`www.cs.berkeley.edu∼murphyk/Bayes/compare.html`

[6]IPARP: see `www.killmine.com/prod01.htm`.

8.2.1 BNT and Intel

In the summer of 2001, while I was an intern, Intel decided to use BNT to jump-start their new research project into graphical models, being led by Gary Bradski. In the short term, their goal is to use graphical models for applications, such as data mining, (audio-visual) speech recognition and low level vision. In the medium term, their goal is to create an open-source graphical models library (optimized for Intel architectures) that will stimulate commercial interest in the application of graphical models. And in the long term, their goal is to identify how future microprocessor architecture design should be changed so as to enable large-scale applications of graphical models.

In terms of data mining, Intel has a team of 3 people in China who will add a variety of structure learning algorithms (see Section 7.3.2) to BNT. Intel also has a team of 3 people in the USA who are developing data visualization tools. The goal is to apply these tools to data from Intel's factories.

In terms of an open-source library, in Fall 2001 Intel will assemble a team of about 5 people in Russia who may work on translating BNT from Matlab to C/C++. (BNT is currently about 38,000 lines of code, of which 8000 are comments.) The ultimate goal is to have an open-source library that it is entirely written in C/C++. This will be callable from Matlab and perhaps other scripting languages such as R[7]. (Intel already has some similar experience with OpenCV[8], Intel's open-source computer vision library, which was also mostly developed in Russia.)

8.2.2 BNT and OpenBayes

In January 2001, Richard Dybowski founded the OpenBayes egroup[9] to discuss the creation of an open-source Bayes net library. Since that time, the group has had much lively discussion, but nothing concrete has come of it.

Richard does not consider BNT truly open-source because it relies on Matlab, which is not free. However, the BNT code itself is freely available (currently under the GNU Library General Public License), and various people have contributed to it; I would like to mention their contributions here. Pierpaolo Brutti is integrating glmlab[10] into BNT, to allow GLM CPDs. Taylan Cemgil contributed some simple graph visualization routines. Wei Hu (Intel China) implemented a C version of the junction tree algorithm. Shan Huang (Intel China) is adding stable CG inference. Tamar Kushnir implemented the IC* structure-learning algorithm. Ilya Shpitser has implemented graph triangulation in C. Yair Weiss contributed several routines relating to loopy belief propagation. Ron Zohar added code to compute the most probable explanation. In addition, numerous people have helped to find bugs, and Philippe LeRay translated the documentation into French.

8.3 Future

Currently I receive code contributions by email, and integrate them by hand, but this will clearly not scale up to handle many people simultaneously working on the project. Hence I plan to move BNT to sourceforge, or some other open-source development environment.

---

[7]`www.r-project.org`

[8]`www.intel.com/research/mrl/research/opencv/`

[9]See `groups.yahoo.com/group/openbayes` and
`www.cs.berkeley.edu/∼murphyk/OpenBayes/index.html`.

[10]`www.sci.usq.edu.au/staff/dunn/glmlab/glmlab.html`

I have a fairly long wish-list for features I would like to see added to BNT, which can be divided into 3 broad categories: new functionality, speedups, and I/O issues. The major additional functionality I would like to see includes

- Add tree-structured CPDs.

- Add support for Bayesian modeling, i.e., parameters can be explicitly represented as nodes.

- Add support for online inference and learning. (The current DBN support is designed for offline/batch processing.)

- Implement more approximate inference algorithms, such as expectation propagation [Min01] and variational Bayes [GB00].

- Add support for non-DAG graphical models, e.g., dependency nets, MRFs and feature-based (maxent) models. Each such model type will be its own class, and will have its own methods for inference and learning, as discussed in Section 7.1. The resulting program will be called GMT (Graphical Models Toolbox) to emphasize its generality, and the fact that it will not be completely backwards compatible with BNT.

In terms of I/O, I would like to see

- An interface to BUGS[11] and Hydra[12], which implement Gibbs sampling and Metropolis Hastings respectively.

- A file parser which can handle the proposed XML file format.[13] Ken Shan has implemented a web-based program to translate a BIF file to a sequence of calls to the BNT API.[14]

- A program that lets users create graphs interactively. Hao Wang has implemented a prototype in Matlab, which Kui Xu is currently extending (both people work for Intel China).

- A program that automatically lays out graphs in a way that makes them look pretty (e.g., by minimizing line crossings). This is essential for interpreting the results of structure learning. This could be implemented using GraphViz[15], an open-source graph visualization package. Bob Davies (Intel California) has developed a prototype using Tom Sawyer[16], a commercial version of GraphViz. His prototype lets the user interactively edit the results of the layout process, and hence it essentially supercedes Wang's Matlab program mentioned above.

- A program that lets users visualize and edit CPDs. This is particularly useful for tree-structured CPDs.

- A GUI that ties together the graph/CPD editor/visualizer and the Matlab backend, which does the inference and learning.

---

[11]www.mrc-bsu.cam.ac.uk/bugs/
[12]www.biostat.washington.edu/~warnes/Hydra
[13]See www.cs.berkeley.edu/~murphyk/OpenBayes/file_formats.html for a list of the various file formats that are commonly used or that have been proposed.
[14]www.digitas.harvard.edu/~ken/bif2bnt
[15]www.research.att.com/sw/tools/graphviz/
[16]www.tomsawyer.com

In terms of speed, the current major bottleneck is related to the creation and manipulation of potentials (see Sections 3.1.1 and 7.2.1). (This functionality is needed for any sum-product type inference algorithm, such as belief propagation and junction tree.) It is fairly simple to improve on the current implementation, even in Matlab, but ultimately a lot of the code will need to be translated to C/C++. We are looking into automatic code generation technology (c.f., NASA's AutoBayes project [FS01]). The goal is code that is fast enough to apply to large-scale applications, such as real-time computer vision.

Finally, next year BNT will be "bundled" with the textbook that Michael Jordan is currently writing on graphical models [Jor02], which is sure to be a best-seller.

Acknowledgements

References

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

[BFGK96]   C. Boutilier, N. Friedman, M. Goldszmidt, and D. Koller. Context-Specific Independence in Bayesian Networks. In *UAI*, 1996.

[BGT93]   C. Berrou, A. Glavieux, and P. Thitimajashima. Near Shannon limit error-correcting coding and decoding: Turbo codes. *Proc. IEEE Intl. Comm. Conf.*, 1993.

[BJ01]   F. Bach and M. Jordan. Thin junction trees. In *NIPS*, 2001.

[BKRK97]   J. Binder, D. Koller, S. J. Russell, and K. Kanazawa. Adaptive probabilistic networks with hidden variables. *Machine Learning*, 29:213–244, 1997.

[CBL97]   J. Cheng, D. Bell, and W. Liu. Learning belief networks from data: an information theory based approach. In *Proc. 6th ACM Intl. Conf. on Info. and Knowl. Mgmt*, 1997.

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

[CH92]   G. Cooper and E. Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9:309–347, 1992.

[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.

[CY99]   G. Cooper and C. Yoo. Causal discovery from a mixture of experimental and observational data. In *UAI*, 1999.

[Dec96]   R. Dechter. Bucket elimination: a unifying framework for probabilistic inference. In *UAI*, 1996.

[DR72]   J. Darroch and D. Ratcliff. Generalized iterative scaling for log-linear models. *Annals of Math. Statistics*, 43(5):1470–1480, 1972.

[Dra95]   D. L. Draper. Clustering without (thinking about) triangulation. In *UAI*, 1995.

[Edw00]   D. Edwards. *Introduction to graphical modelling*. Springer, 2000. 2nd edition.

[FK00]   N. Friedman and D. Koller. Being Bayesian about network structure. In *UAI*, 2000.

[Fre98]   B. Frey. *Graphical Models for Machine Learning and Digital Communication*. MIT Press, 1998.

[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.

[FS01]   B. Fischer and J. Schumann. Generating data analysis programs from statistical models. *J. Functional Programming*, 2001. Submitted.

[GB00]   Z. Ghahramani and M. Beal. Propagation algorithms for variational Bayesian learning. In *NIPS-13*, 2000.

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

[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.

[HGC94]   D. Heckerman, D. Geiger, and M. Chickering. Learning Bayesian networks: the combination of knowledge and statistical data. Technical Report MSR-TR-09-09, Microsoft Research, 1994.

[JA90]   F. Jensen and S. K. Andersen. Approximations in Bayesian belief universes for knowledge-based systems. In *UAI*, 1990.

[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.

[JJ93]   M. Jamshidian and R. I. Jennrich. Conjugate gradient acceleration of the EM algorithm. *JASA*, 88(421):221–228, 1993.

[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.

[Jor02]   M. Jordan. *Probabilistic Graphical Models*. MIT Press, 2002. In preparation.

[KFL01]   F. Kschischang, B. Frey, and H-A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Trans Info. Theory*, February 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.

[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.

[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.

[LJ99]   S. Lauritzen and F. Jensen. Stable local computation with conditional Gaussian distributions. Technical Report R-99-2014, Dept. Math. Sciences, Aalborg Univ., 1999.

[Mac98]   D. MacKay. Introduction to monte carlo methods. In M. Jordan, editor, *Learning in graphical models*. MIT Press, 1998.

[MH97]   C. Meek and D. Heckerman. Structure and parameter learning for causal independence and causal interaction models. In *UAI*, pages 366–375, 1997.

[Min01]   T. Minka. Expectation propagation for approximate Bayesian inference. In *UAI*, 2001.

[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.

[MT99]   D. Margaritas and S. Thrun. Bayesian Network Induction via Local Neighborhoods. In *NIPS-13*, 1999.

[MT01]   D. Margaritas and S. Thrun. A Bayesian Multiresolution Independence Test for Continuous Variables. In *UAI*, 2001.

[Mur98a]   K. P. Murphy. Filtering and smoothing in linear dynamical systems using the junction tree algorithm. Technical report, U.C. Berkeley, Dept. Comp. Sci, 1998.

[Mur98b]   K. P. Murphy. Inference and learning in hybrid Bayesian networks. Technical Report 990, U.C. Berkeley, Dept. Comp. Sci, 1998.

[Mur99]   K. P. Murphy. A variational approximation for Bayesian networks with discrete and continuous latent variables. In *UAI*, 1999.

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

[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.

[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.

[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.

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

[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.

[RS97]   M. Ramoni and P. Sebastiani. Learning Bayesian networks from incomplete databases. In *UAI*, 1997.

[SGS01]   P. Spirtes, C. Glymour, and R. Scheines. *Causation, Prediction, and Search*. MIT Press, 2001. 2nd edition.

[SK89]   R. Shachter and C. R. Kenley. Gaussian influence diagrams. *Managment Science*, 35(5):527–550, 1989.

[SO01]   D. Saad and M. Opper. *Advanced Mean Field Methods*. MIT Press, 2001.

[TW01]   Y-W. Teh and M. Welling. The unified propagation and scaling algorithm. In *NIPS*, 2001.

[YFW01]   J. Yedidia, W. T. Freeman, and Y. Weiss. Understanding belief propagation and its generalizations. In *IJCAI*, 2001.