Probabilistic inference in graphical models

Michael I. Jordan
jordan@cs.berkeley.edu
Division of Computer Science and Department of Statistics
University of California, Berkeley

Yair Weiss
yweiss@cs.huji.ac.il
School of Computer Science and Engineering
Hebrew University

RUNNING HEAD: Probabilistic inference in graphical models

Correspondence:
Michael I. Jordan
EECS Computer Science Division
387 Soda Hall # 1776
Berkeley, CA 94720-1776
Phone: (510) 642-3806
Fax: (510) 642-5775
email: jordan@cs.berkeley.edu

INTRODUCTION

A "graphical model" is a type of probabilistic network that has roots in several different research communities, including artificial intelligence (Pearl, 1988), statistics (Lauritzen, 1996), error-control coding (Gallager, 1963), and neural networks. The graphical models framework provides a clean mathematical formalism that has made it possible to understand the relationships among a wide variety of network-based approaches to computation, and in particular to understand many neural network algorithms and architectures as instances of a broader probabilistic methodology.

Graphical models use graphs to represent and manipulate joint probability distributions. The graph underlying a graphical model may be directed, in which case the model is often referred to as a *belief network* or a *Bayesian network* (see BAYESIAN NETWORKS), or the graph may be undirected, in which case the model is generally referred to as a *Markov random field*. A graphical model has both a structural component—encoded by the pattern of edges in the graph—and a parametric component—encoded by numerical "potentials" associated with sets of edges in the graph. The relationship between these components underlies the computational machinery associated with graphical models. In particular, general *inference algorithms* allow statistical quantities (such as likelihoods and conditional probabilities) and information-theoretic quantities (such as mutual information and conditional entropies) to be computed efficiently. These algorithms are the subject of the current article. *Learning algorithms* build on these inference algorithms and allow parameters and structures to be estimated from data (see GRAPHICAL MODELS, PARAMETER LEARNING and GRAPHICAL MODELS, STRUCTURE LEARNING).

BACKGROUND

Directed and undirected graphical models differ in terms of their Markov properties (the relationship between graph separation and conditional independence) and their parameterization (the relationship between local numerical specifications and global joint probabilities). These differences are important in discussions of the family of joint probability distribution that a particular graph can represent. In the inference problem, however, we generally have a specific fixed joint probability distribution at hand, in which case the differences between directed and undirected graphical models are less important. Indeed, in the current article, we treat these classes of model together and emphasize their commonalities.

Let U denote a set of nodes of a graph (directed or undirected), and let X_i denote the random variable associated with node i, for $i \in U$. Let X_C denote the subset of random variables associated with a subset of nodes C, for any $C \subseteq U$, and let $X = X_U$ denote the collection of random variables associated with the graph.

The family of joint probability distributions associated with a given graph can be parameterized in terms of a product over *potential functions* associated with subsets of nodes in the graph. For directed graphs, the basic subset on which a potential is defined consists of a single node and its parents, and a potential turns out to be (necessarily) the conditional probability of the node given its parents. Thus, for a directed graph, we have the following representation for the joint probability:

$$p(x) = \prod_i p(x_i \mid x_{\pi_i}), \tag{1}$$

where $p(x_i \mid x_{\pi_i})$ is the *local conditional probability* associated with node i, and π_i is the set of indices labeling the parents of node i. For undirected graphs, the basic subsets are *cliques* of the graph—subsets of nodes that are completely connected. For a given clique C, let $\psi_C(x_C)$ denote a general potential function—a function that assigns a positive real number to each configuration x_C. We have:

$$p(x) = \frac{1}{Z} \prod_{C \in \mathcal{C}} \psi_C(x_C), \tag{2}$$

where $\mathcal{C}$ is the set of cliques associated with the graph and Z is an explicit normalizing factor, ensuring that $\sum_x p(x) = 1$. (We work with discrete random variables throughout for simplicity).

Eq. (1) can be viewed as a special case of Eq. (2). Note in particular that we could have included a normalizing factor Z in Eq. (1), but, as is easily verified, it is necessarily equal to one. Second, note that $p(x_i \mid x_{\pi_i})$ is a perfectly good example of a potential function, except that the set of nodes that it is defined on—the collection $\{i \cup \pi_i\}$—is not in general a clique (because the parents of a given node are not in general interconnected). Thus, to treat Eq. (1) and Eq. (2) on an equal footing, we find it convenient to define the so-called *moral graph* $\mathcal{G}^m$ associated with a directed graph $\mathcal{G}$. The moral graph is an undirected graph obtained by connecting all of the parents of each node in $\mathcal{G}$, and removing the arrowheads. On the moral graph, a conditional probability $p(x_i \mid x_{\pi_i})$ is a potential function, and Eq. (1) reduces to a special case of Eq. (2).

PROBABILISTIC INFERENCE

Let (E, F) be a partitioning of the node indices of a graphical model into disjoint subsets, such that (X_E, X_F) is a partitioning of the random variables. There are two basic kinds of inference problem that we wish to solve:

- *Marginal probabilities*:

$$p(x_E) = \sum_{x_F} p(x_E, x_F). \tag{3}$$

- *Maximum a posteriori (MAP) probabilities*:

$$p^*(x_E) = \max_{x_F} p(x_E, x_F). \tag{4}$$

From these basic computations we can obtain other quantities of interest. In particular, the *conditional probability* $p(x_F \mid x_E)$ is equal to:

$$p(x_F \mid x_E) = \frac{p(x_E, x_F)}{\sum_{x_F} p(x_E, x_F)}, \tag{5}$$

and this is readily computed for any x_F once the denominator is computed—a marginalization computation. Moreover, we often wish to combine conditioning and marginalization, or conditioning, marginalization and MAP computations. For example, letting (E, F, H) be a partitioning of the node indices, we may wish to compute:

$$p(x_F \mid x_E) = \frac{p(x_E, x_F)}{\sum_{x_F} p(x_E, x_F)} = \frac{\sum_{x_H} p(x_E, x_F, x_H)}{\sum_{x_F} \sum_{x_H} p(x_E, x_F, x_H)}. \tag{6}$$

We first perform the marginalization operation in the numerator and then perform a subsequent marginalization to obtain the denominator.

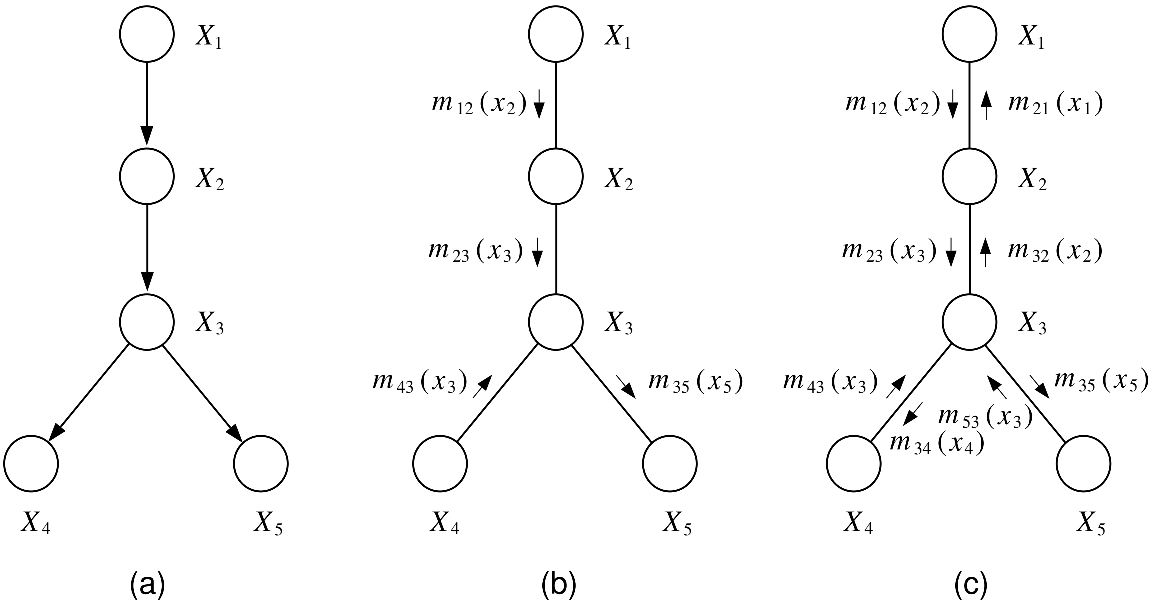

Figure 1: (a) A directed graphical model. (b) The intermediate terms that arise during a run of ELIMINATE can be viewed as messages attached to the edges of the moral graph. Here the elimination order was $(1, 2, 4, 3)$. (c) The set of all messages computed by the sum-product algorithm.

Elimination

In this section we introduce a basic algorithm for inference known as "elimination." Although elimination applies to arbitrary graphs (as we will see), our focus in this section is on trees.

We proceed via an example. Referring to the tree in Figure 1(a), let us calculate the marginal probability $p(x_5)$. We compute this probability by summing the joint probability with respect to $\{x_1, x_2, x_3, x_4\}$. We must pick an order over which to sum, and with some malice of forethought, let us choose the order $(1, 2, 4, 3)$. We have:

$$\begin{aligned}
p(x_5) &= \sum_{x_3}\sum_{x_4}\sum_{x_2}\sum_{x_1} p(x_1, x_2, x_3, x_4, x_5) \\
&= \sum_{x_3}\sum_{x_4}\sum_{x_2}\sum_{x_1} p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_2)p(x_4 \mid x_3)p(x_5 \mid x_3) \\
&= \sum_{x_3} p(x_5 \mid x_3) \sum_{x_4} p(x_4 \mid x_3) \sum_{x_2} p(x_3 \mid x_2) \sum_{x_1} p(x_1)p(x_2 \mid x_1) \\
&= \sum_{x_3} p(x_5 \mid x_3) \sum_{x_4} p(x_4 \mid x_3) \sum_{x_2} p(x_3 \mid x_2) m_{12}(x_2),
\end{aligned}$$

where we introduce the notation $m_{ij}(x_j)$ to refer to the intermediate terms that arise in performing the sum. The index i refers to the variable being summed over, and the index j refers to the other variable appearing in the summand (for trees, there will never be more than two variables appearing in any summand). The resulting term is a function of x_j. We continue the derivation:

$$\begin{aligned}
p(x_5) &= \sum_{x_3} p(x_5 \mid x_3) \sum_{x_4} p(x_4 \mid x_3) \sum_{x_2} p(x_3 \mid x_2) m_{12}(x_2) \\
&= \sum_{x_3} p(x_5 \mid x_3) \sum_{x_4} p(x_4 \mid x_3) m_{23}(x_3) \\
&= \sum_{x_3} p(x_5 \mid x_3) m_{23}(x_3) \sum_{x_4} p(x_4 \mid x_3) \\
&= \sum_{x_3} p(x_5 \mid x_3) m_{23}(x_3) m_{43}(x_3) \\
&= m_{35}(x_5).
\end{aligned}$$

The final expression is a function of x_5 only and is the desired marginal probability.

This computation is formally identically in the case of an undirected graph. In particular, an undirected version of the tree in Figure 1(a) has the parameterization:

$$p(x) = \frac{1}{Z}\psi_{12}(x_1, x_2)\psi_{23}(x_2, x_3)\psi_{34}(x_3, x_4)\psi_{35}(x_3, x_5). \tag{7}$$

The first few steps of the computation of $p(x_5)$ are then as follows:

$$\begin{aligned} p(x_5) &= \frac{1}{Z}\sum_{x_3}\psi_{35}(x_3, x_5)\sum_{x_4}\psi_{34}(x_3, x_4)\sum_{x_2}\psi_{23}(x_2, x_3)\sum_{x_1}\psi_{12}(x_1, x_2) \\ &= \frac{1}{Z}\sum_{x_3}\psi_{35}(x_3, x_5)\sum_{x_4}\psi_{34}(x_3, x_4)\sum_{x_2}\psi_{23}(x_2, x_3)m_{12}(x_2), \end{aligned}$$

and the remainder of the computation proceeds as before.

These algebraic manipulations can be summarized succinctly in terms of a general algorithm that we refer to here as Eliminate (see Figure 2). The algorithm maintains an "active list" of potentials, which at the outset represent the joint probability and at the end represent the desired marginal probability. Nodes are removed from the graph according to an elimination ordering that must be specified. The algorithm applies to both directed and undirected graphs. Also, as we will see shortly, it is in fact a general algorithm, applying not only to trees but to general graphs.

Message-passing algorithms

In many problems we wish to obtain more than a single marginal probability. Thus, for example, we may wish to obtain both $p(x_4)$ and $p(x_5)$ in Figure 1(a). Although we could compute each marginal with a separate run of Eliminate, this fails to exploit the fact

ELIMINATE(G)
  place all potentials $\psi_C(x_C)$ on the active list
  choose an ordering I of the indices F
  for each X_i in I
    find all potentials on the active list that reference X_i
      and remove them from the active list
    define a new potential as the sum (with respect to x_i) of the
      product of these potentials
    place the new potential on the active list
  end
  return the product of the remaining potentials

Figure 2: A simple elimination algorithm for marginalization in graphical models.

that common intermediate terms appear in the different runs. We would like to develop an algebra of intermediate terms that allows them to be reused efficiently.

Suppose in particular that we wish to compute $p(x_4)$ in the example in Figure 1(a). Using the elimination order $(1, 2, 5, 3)$, it is easily verified that we generate the terms $m_{12}(x_2)$ and $m_{23}(x_3)$ as before, and also generate new terms $m_{53}(x_3)$ and $m_{34}(x_4)$.

As suggested by Figure 1(b), the intermediate terms that arise during elimination can be viewed as "messages" attached to edges in the moral graph. Rather than viewing inference as an elimination process, based on a global ordering, we instead view inference in terms of local computation and routing of messages. The key operation of summing a product can be written as follows:

$$m_{ij}(x_j) = \sum_{x_i} \psi_{ij}(x_i, x_j) \prod_{k \in N(i) \setminus j} m_{ki}(x_i), \tag{8}$$

where $N(i)$ is the set of neighbors of node i. Thus, summing over x_i creates a message

$m_{ij}(x_j)$ which is sent to the node j. The reader can verify that each step in our earlier computation of $p(x_5)$ has this form.

A node can send a message to a neighboring node once it has received messages from all of its other neighbors. As in our example, a message arriving at a leaf node is necessarily a marginal probability. In general, the marginal probability at a node is given by the product of all incoming messages:

$$p(x_i) \propto \prod \prod_{k \in N(i)} m_{ki}(x_i). \tag{9}$$

The pair of equations given by Eq. (8) and Eq. (9) define an algorithm known as "sum-product algorithm" or the "belief propagation algorithm." It is not difficult to prove that this algorithm is correct for trees.

The set of messages needed to compute all of the individual marginal probabilities for the graph in Figure 1(a) is shown in Figure 1(b). Note that a pair of messages is sent along each edge, one message in each direction.

Neural network also involve message-passing algorithms and local numerical operations. An important difference, however, is that in the neural network setting each node generally has a single "activation" value that it passes to all of its neighbors. In the sum-product algorithm, on the other hand, individual messages are prepared for each neighbor. Moreover, the message $m_{ij}(x_j)$ from i to j is not included in the product that node j forms in computing a message to send back to node i. The sum-product algorithm avoids "double-counting."

Maximum a posteriori (MAP) probabilities

Referring again to Figure 1(a), let us suppose that we wish to compute $p^*(x_5)$, the maximum probability configuration of the variables (X_1, X_2, X_3, X_4), for a given value of X_5. Again choosing a particular ordering of the variables, we compute:

$$
\begin{aligned}
p^*(x_5) &= \max_{x_3} \max_{x_4} \max_{x_2} \max_{x_1} p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_2)p(x_4 \mid x_3)p(x_5 \mid x_3) \\
&= \max_{x_3} p(x_5 \mid x_3) \max_{x_4} p(x_4 \mid x_3) \max_{x_2} p(x_3 \mid x_2) \max_{x_1} p(x_1)p(x_2 \mid x_1),
\end{aligned}
$$

and the remaining computation proceeds as before. We see that the algebraic operations involved in performing the MAP computation are isomorphic to those in the earlier marginalization computation. Indeed, both the elimination algorithm and the sum-product algorithm extend immediately to MAP computation; we simply replace "sum" with "max" throughout in both cases. The underlying justification is that "max" commutes with products just as "sum" does.

General graphs

Our goal in this section is to describe the *junction tree algorithm*, a generalization of the sum-product algorithm that is correct for arbitrary graphs. We derive the junction tree algorithm by returning to the elimination algorithm. Note that ELIMINATE is correct for arbitrary graphs—the algorithm simply describes the creation of intermediate terms in a chain of summations that compute a marginal probability. Thus the algorithm is correct, but it is limited to the computation of a single marginal probability.

To show how to generalize the elimination algorithm to allow all individual marginals to be computed, we again proceed by example. Referring to the graph in Figure 3(a), suppose that we wish to calculate the conditional probability $p(x_1)$. Let us use the elimination ordering $(5, 4, 3, 2)$. At the first step, in which we sum over x_5, we remove the potentials

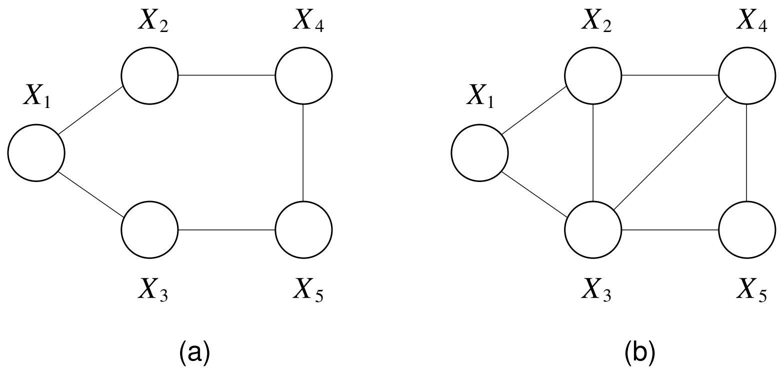

Figure 3: (a) An undirected graphical model. (b) The same model, with additional edges that reflect the dependencies created by the elimination algorithm.

$\psi_{35}(x_3, x_5)$ and $\psi_{45}(x_4, x_5)$ from the active list and form the sum:

$$m_{32}(x_3, x_4) = \sum_{x_5} \psi_{35}(x_3, x_5)\psi_{45}(x_4, x_5), \tag{10}$$

where the intermediate term, which is clearly a function of x_3 and x_4, is denoted $m_{32}(x_3, x_4)$. (We explain the subscripts below). The elimination of X_5 has created an intermediate term that effectively links X_3 and X_4, variables that were not linked in the original graph. Similarly, at the following step, we eliminate X_4:

$$m_{21}(x_2, x_3) = \sum_{x_4} \psi_{24}(x_2, x_4)m_{32}(x_3, x_4) \tag{11}$$

and obtain a term that links X_2 and X_3, variables that were not linked in the original graph.

A graphical record of the dependencies induced during the run of ELIMINATE is shown in Figure 3(b). We could also have created this graph according to a simple graph-theoretic algorithm in which nodes are removed in order from a graph, where, when a node is removed, its remaining neighbors are linked. Thus, for example, when node 5 is removed, nodes 3 and

4 are linked. When node 4 is removed, nodes 2 and 3 are linked. Let us refer to this algorithm as GraphEliminate.

We can also obtain the desired marginal $p(x_1)$ by working with the "filled-in" graph in Figure 3(b) from the outset. Noting that the cliques in this graph are $C_1 = \{x_1, x_2, x_3\}$ $C_2 = \{x_2, x_3, x_4\}$ and $C_3 = \{x_3, x_4, x_5\}$, and defining the potentials:

$$\begin{aligned}
\psi_{C_1}(x_1, x_2, x_3) &= \psi_{12}(x_1, x_2)\psi_{13}(x_1, x_3) \\
\psi_{C_2}(x_2, x_3, x_4) &= \psi_{24}(x_2, x_4) \\
\psi_{C_3}(x_3, x_4, x_5) &= \psi_{35}(x_3, x_5)\psi_{45}(x_4, x_5),
\end{aligned}$$

we obtain exactly the same product of potentials as before. Thus we have:

$$\begin{aligned}
p(x) = \frac{1}{Z}\psi_{12}(x_1, x_2)\psi_{13}(x_1, x_3)\psi_{C_2}(x_2, x_3, x_4) &= \psi_{24}(x_2, x_4)\psi_{35}(x_3, x_5)\psi_{45}(x_4, x_5) \\
& \frac{1}{Z}\psi_{C_1}\psi_{C_2}\psi_{C_3}.
\end{aligned} \tag{12}$$

We have essentially transferred the joint probability distribution from Figure 3(a) to Figure 3(b). Moreover, the steps of the elimination algorithm applied to Figure 3(b) are exactly the same as before, and we obtain the same marginal. An important difference, however, is that in the case of Figure 3(b) all of the intermediate potentials created during the run of the algorithm are also supported by cliques in the graph.

Graphs created by GraphEliminate are known as *triangulated graphs*, and they have a number of special properties. In particular, they allow the creation of a data structure known as a *junction tree* on which a generalized message-passing algorithm can be defined.

A junction tree is a tree, in which each node is a clique from the original graph. Messages, which correspond to intermediate terms in Eliminate, pass between these cliques.

Although a full discussion of the construction of junction trees is beyond the scope of the article, it is worth noting that a junction tree is not just any tree of cliques from a triangulated graph. Rather, it is a maximal spanning tree (of cliques), with weights given by the cardinalities of the intersections between cliques.

Given a triangulated graph, with cliques $C_i \in \mathcal{C}$ and potentials $\psi_{C_i}(x_{C_i})$, and given a corresponding junction tree (which defines links between the cliques), we send the following "message" from clique C_i to clique C_j:

$$m_{ij}(x_{S_{ij}}) = \sum_{C_i \setminus S_{ij}} \psi_{C_i}(x_{C_i}) \prod_{k \in \mathcal{N}(i) \setminus j} m_{ki}(x_{S_{ki}}), \tag{13}$$

where $S_{ij} = C_i \cap C_j$, and where $\mathcal{N}(i)$ are the neighbors of clique C_i in the junction tree. Moreover, it is possible to prove that we obtain marginal probabilities as products of messages. Thus:

$$p(x_{C_i}) \propto \prod_{k \in \mathcal{N}(i)} m_{ki}(x_{S_{ki}}) \tag{14}$$

is the marginal probability for clique C_i. (Marginals for single nodes can be obtained via further marginalization: i.e., $p(x_i) = \sum_{C \setminus i} p(x_C)$, for $i \in C$).

The junction tree corresponding to the triangulated graph in Figure 3(b) is shown in Figure 4, where the corresponding messages are also shown. The reader can verify that the leftward-going messages are identical to the intermediate terms created during the run of Eliminate. The junction tree algorithm differs from Eliminate, however, in that mes-

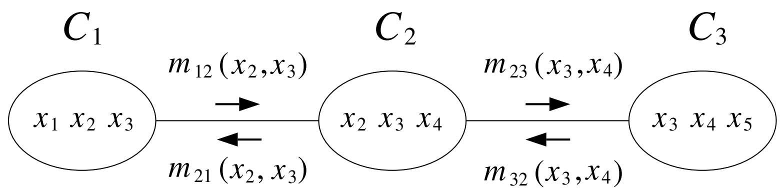

Figure 4: The junction tree corresponding to the triangulated graph in Figure 3(b).

sages pass in all directions, and the algorithm yields all clique marginals, not merely those corresponding to a single clique.

The sum-product algorithm described earlier in Eq. (8) and Eq. (9) is a special case of Eq. (13) and Eq. (14), obtained by noting that the original tree in Figure 1(a) is already triangulated, and has pairs of nodes as cliques. In this case, the "separator sets" S_{ij} are singleton nodes.

Once again, the problem of computing MAP probabilities can be solved with a minor change to the basic algorithm. In particular, the "sum" in Eq. (13) is changed to a "max."

There are many variations on exact inference algorithms, but all of them are either special cases of the junction tree algorithm or are close cousins. The basic message from the research literature on exact inference is that the operations of triangulating a graph and passing messages on the resulting junction tree capture in a succinct way the basic algebraic structure of probabilistic inference.

Computational complexity

The computational complexity of the junction tree algorithm is a function of the size of the cliques upon which message-passing operations are performed. In particular, summing a clique potential is exponential in the number of nodes in the clique.

The problem of finding the optimal triangulation—the triangulation yielding the smallest maximal clique—turns out to be NP-hard. Clearly, if we had to search over all possible elimination orderings, the search would take exponential time. Triangulation can also be defined in other ways, however, and practical triangulation algorithms need not search over orderings. But the problem is still intractable, and can be a practical computational bottleneck.

An even more serious problem is that in practical graphical models the original graph may have large cliques, or long loops, and even the optimal triangulation would yield unacceptable complexity. This problem is particularly serious because it arises not during the "compile-time" operation of triangulation, but during the "run-time" operation of message-passing. Problems in error-control coding and image processing are particularly noteworthy for yielding such graphs, as are discretizations of continuous-time problems, and layered graphs of the kinds studied in the neural network field. To address these problems, we turn to the topic of approximate probabilistic inference.

APPROXIMATE INFERENCE

The junction tree algorithm focuses on the algebraic structure of probabilistic inference, exploiting the conditional independencies present in a joint probability distribution, as encoded in the pattern of (missing) edges in the graph. There is another form of structure in probability theory, however, that is not exploited in the junction tree framework, and which leads us to hope that successful approximate inference algorithms can be developed. In particular, laws of large numbers and other concentration theorems in probability theory show that sums and products of large numbers of terms can behave in simple, predictable ways, despite the apparent combinatorial complexity of these operations. Approximate algorithms attempt to exploit these numerical aspects of probability theory.

We discuss two large classes of approximate inference algorithms in this section—Monte Carlo algorithms and variational algorithms. While these classes do not exhaust all of the approximation techniques that have been studied, they capture the most widely-used examples.

Monte Carlo algorithms

Monte Carlo algorithms are based on the fact that while it may not be feasible to compute expectations under $p(x)$, it may be possible to obtain samples from $p(x)$, or from a closely-related distribution, such that marginals and other expectations can be approximated using sample-based averages. We discuss three examples of Monte Carlo algorithms that are commonly used in the graphical model setting—Gibbs sampling, the Metropolis-Hastings algorithm and importance sampling (for a comprehensive presentation of these methods and others, see Andrieu, et al., in press).

Gibbs sampling is an example of a Markov chain Monte Carlo (MCMC) algorithm. In an MCMC algorithm, samples are obtained via a Markov chain whose stationary distribution is the desired $p(x)$. The state of the Markov chain is a set of assignments of values to each of the variables, and, after a suitable "burn-in" period so that the chain approaches its stationary distribution, these states are used as samples.

The Markov chain for the Gibbs sampler is constructed in a straightforward way: (1) at each step one of the variables X_i is selected (at random or according to some fixed sequence), (2) the conditional distribution $p(x_i \mid x_{U \setminus i})$ is computed, (3) a value x_i is chosen from this distribution, and (4) the sample x_i replaces the previous value of the ith variable.

The implementation of Gibbs sampling thus reduces to the computation of the conditional distributions of individual variables given all of the other variables. For graphical models, these conditionals take the following form:

$$p(x_i \mid x_{U \setminus i}) = \frac{\prod_{C \in \mathcal{C}} \psi_C(x_C)}{\sum_{x_i} \prod_{C \in \mathcal{C}} \psi_C(x_C)} = \frac{\prod_{C \in \mathcal{C}_i} \psi_C(x_C)}{\sum_{x_i} \prod_{C \in \mathcal{C}_i} \psi_C(x_C)}, \tag{15}$$

where $\mathcal{C}_i$ denotes the set of cliques that contain index i. This set is often much smaller than the set $\mathcal{C}$ of all cliques, and in such cases each step of the Gibbs sampler can be implemented efficiently. Indeed, the conditional of node i depends only on the neighbors of node i in the graph, and thus the computation of the conditionals often takes the form of a simple message-passing algorithm that is reminiscent of the sum-product algorithm.

A simple example of a Gibbs sampler is provided by the *Boltzmann machine*, an undirected graphical model in which the potentials are defined on pairwise cliques. Gibbs sampling is often used for inference in the Boltzmann machine, and the algorithm in Eq. (15) takes the form of the classical computation of the logistic function of a weighted sum of the values of neighboring nodes.

When the computation in Eq. (15) is overly complex, the *Metropolis-Hastings algorithm* can provide an effective alternative. Metropolis-Hastings is an MCMC algorithm that is not based on conditional probabilities, and thus does not require normalization. Given the current state x of the algorithm, Metropolis-Hastings chooses a new state $\tilde{x}$ from a "proposal distribution," which often simply involves picking a variable X_i at random and choosing a new value for that variable, again at random. The algorithm then computes the "acceptance probability":

$$\alpha = \min \left(1, \frac{\prod_{C \in \mathcal{C}_i} \psi_C(\tilde{x}_C)}{\prod_{C \in \mathcal{C}_i} \psi_C(x_C)} \right). \tag{16}$$

With probability α the algorithm accepts the proposal and moves to $\tilde{x}$, and with probability $1 - \alpha$ the algorithm remains in state x. For graphical models, this computation also turns out to often takes the form of a simple message-passing algorithm.

While Gibbs sampling and Metropolis-Hastings aim at sampling from $p(x)$, *importance sampling* is a Monte Carlo technique in which samples are chosen from a simpler distribution $q(x)$, and these samples are reweighted appropriately. In particular, we approximate the expectation of a function $f(x)$ as follows:

$$\begin{aligned} E[f(x)] &= \sum_x p(x) f(x) \\ &= \sum_x q(x) \left(\frac{p(x)}{q(x)} f(x) \right) \\ &\approx \frac{1}{N} \sum_{i=1}^{N} \frac{p(x^{(t)})}{q(x^{(t)})} f(x^{(t)}), \end{aligned}$$

where the values $x^{(t)}$ are samples from $q(x)$. The choice of $q(x)$ is in the hands of the designer, and the idea is that $q(x)$ should be chosen to be relatively simple to sample from, while reasonably close to $p(x)$ so that the weight $p(x^{(t)})/q(x^{(t)})$ is reasonably large. In the graphical model setting, natural choices of $q(x)$ are often provided by simplifying the graph underlying $p(x)$ in some way, in particular by deleting edges.

The principal advantages of Monte Carlo algorithms are their simplicity of implementation and their generality. Under weak conditions, the algorithms are guaranteed to converge.

A problem with the Monte Carlo approach, however, is that convergence times can be long, and it can be difficult to diagnose convergence.

We might hope to be able to improve on Monte Carlo methods in situations in which laws of large numbers are operative. Consider, for example, the case in which a node i has many neighbors, such that the conditional $p(x_i \mid x_{U \setminus i})$ has a single, sharply-determined maximum for most configurations of the neighbors. In this case, it would seem wasteful to continue to sample from this distribution; rather, we would like to be able to compute the maximizing value directly in some way. This way of thinking leads to the variational approach to approximate inference.

Variational methods

The key to the variational approach lies in converting the probabilistic inference problem into an optimization problem, such that the standard tools of constrained optimization can be exploited. The basic approach has a similar flavor to importance sampling, but instead of choosing a single $q(x)$ a priori, a family of approximating distributions $\{q(x)\}$ is used, and the optimization machinery chooses a particular member from this family.

We begin by showing that the joint probability $p(x)$ can be viewed as the solution to an optimization problem. In particular, define the *energy* of a configuration x by $E(x) = -\log p(x) - \log Z$, and define the *variational free energy* as follows:

$$\begin{aligned} F(\{q(x)\}) &= \sum_x q(x) E(x) + \sum_x q(x) \log q(x) \\ &= -\sum_x q(x) \log p(x) + \sum_x q(x) \log q(x) - \log Z. \end{aligned}$$

The variational free energy is equal (up to an additive constant) to the Kullback-Leibler divergence between $q(x)$ and $p(x)$. It is therefore minimized when $q(x) = p(x)$ and attains the value of $-\log Z$ at the minimum. We have thus characterized $p(x)$ variationally.

Minimizing F is as difficult as doing exact inference, and much effort has been invested into finding approximate forms of F that are easier to minimize. Each approximate version of F gives a approximate variational inference algorithm.

For example, the simplest variational algorithm is the *mean field* approximation, in which $\{q(x)\}$ is restricted to the family of factorized distributions: $q(x) = \prod_i q_i(x_i)$. In this case F simplifies to:

$$F_{MF}(\{q_i\}) = -\sum_C \sum_{x_C} \log \psi_C(x_C) \prod_{i \in C} q_i(x_i) + \sum_i \sum_{x_i} q_i(x_i) \log q_i(x_i), \tag{17}$$

subject to the constraint $\sum_{x_i} q_i(x_i) = 1$.

Setting the derivative with respect to $q_i(x_i)$ equal to zero gives:

$$q_i(x_i) = \alpha \exp\left(\sum_C \sum_{x_C \setminus i} \log \psi_C(x_C) \prod_{j \in C, j \neq i} q_j(x_j)\right) \tag{18}$$

where α is a normalization constant chosen so that $\sum_{x_i} q_i(x_i) = 1$. The sum over cliques C is over all cliques that node i belongs to.

Eq. (18) defines an approximate inference algorithm. We initialize approximate marginals $q(x_i)$ for all nodes in the graph and then update the approximate marginal at one node based on those at neighboring nodes (note that the right hand side of Eq. (18) depends only on cliques that node i belongs to). This yields a message-passing algorithm that is similar to neural network algorithms; in particular, the value $q(x_i)$ can be viewed as the "activation" of node i.

More elaborate approximations to the free energy give better approximate marginal probabilities. While the mean field free energy depends only on approximate marginals at single nodes, the *Bethe free energy* depends on approximate marginals at single nodes $q_i(x_i)$ as well as approximate marginals on cliques $q_C(x_C)$:

$$F_\beta(\{q_i, q_C\}) = \sum_C \sum_{x_C} q_C(x_C) \log \frac{q_C(x_C)}{\psi_c(x_C)} \tag{19}$$

$$- \sum_i (d_i - 1) \sum_{x_i} q_i(x_i) \log q_i(x_i) \tag{20}$$

where $d_i - 1$ denotes the number of cliques that node i belongs to.

The approximate clique marginals and the approximate singleton marginals must satisfy a simple marginalization constraint: $\sum_{x_C \backslash i} q_C(x_C) = q_i(x_i)$. When we add Lagrange multipliers and differentiate the Lagrangian we obtain a set of fixed point equations. Surprisingly, these equations end up being equivalent to the "sum-product" algorithm for trees in Eq. (8). The messages $m_{ij}(x_j)$ are simply exponentiated Lagrange multipliers. Thus the Bethe approximation is equivalent to applying the local message-passing scheme developed for trees to graphs that have loops (see Yedidia, Freeman, and Weiss, 2001). This approach to approximate inference has been very successful in the domain of error-control coding, allowing practical codes based on graphical models to nearly reach the Shannon limit.

DISCUSSION

The unified perspective on inference algorithms that we have presented in this article has arisen through several different historical strands. We briefly summarize these strands here and note some of the linkages with developments in the neural network field.

The elimination algorithm has had a long history. The "peeling" algorithm developed by geneticists is an early example (Cannings, Thompson, & Skolnick, 1978), as are the "decimation" and "transfer matrix" procedures in statistical physics (Itzykson & Drouffe, 1987). For a recent discussion of elimination algorithms, including more efficient algorithms than the simple ELIMINATE algorithm presented here, see Dechter (1999).

Belief propagation has also had a long history. An early version of the sum-product algorithm was studied by Gallager (1963) in the context of error-control codes. See McEliece, MacKay and Cheng (1996) and Kschischang, Frey and Loeliger (2001) for recent perspectives. Well-known special cases of sum-product include the forward-backward algorithm for hidden Markov models (see HIDDEN MARKOV METHODS), and the Kalman filtering/smoothing algorithms for state-space models. A systematic presentation of the sum-product algorithm was provided by Pearl (1988).

The variant of the junction tree algorithm that we have defined is due to Shafer and Shenoy (1990), and has also been called the "generalized distributive law" by Aji and McEliece (2000). A closely-related variant known as the "Hugin algorithm" is described by Jensen (2001).

Many neural network architectures are special cases of the general graphical model formalism, both representationally and algorithmically. Special cases of graphical models include essentially all of the models developed under the rubric of "unsupervised learning" (see UNSUPERVISED LEARNING WITH GLOBAL OBJECTIVE FUNCTIONS, INDEPENDENT COMPONENT ANALYSIS, and HELMHOLTZ MACHINES AND SLEEP-WAKE LEARNING), as well as Boltzmann machines (see SIMULATED ANNEALING AND BOLTZMANN MACHINES), mixtures of experts (see MODULAR AND HIERARCHICAL LEARNING SYSTEMS), and radial basis function networks (see RADIAL BASIS FUNCTION NETWORKS). Many other neural networks, including the classical multilayer perceptron (see PERCEPTRONS, ADALINES AND BACKPROPAGATION) can be profitably analyzed from the point of view of graphical models. For more discussion of these links, see the articles in Jordan (1999).

REFERENCES

Aji, S. M., & McEliece, R. J. (2000). The generalized distributive law. *IEEE Transactions on Information Theory, 46*, 325–343.

* Andrieu, C., De Freitas, J., Doucet, A., & Jordan, M. I. (in press). An introduction to MCMC for machine learning. *Machine Learning.*

Cannings, C., Thompson, E. A., & Skolnick, M. H. (1978). Probability functions on complex pedigrees. *Advances in Applied Probability, 10*, 26-91.

Dechter, R. Bucket elimination: A unifying framework for probabilistic inference. In Jordan, M. I. (Ed.). (1999). *Learning in Graphical Models*, Cambridge, MA: MIT Press.

Gallager, R. G. (1963). *Low-Density Parity Check Codes.* Cambridge, MA: MIT Press.

Itzykson, C., & Drouffe, J. (1991). *Statistical Field Theory.* Cambridge: Cambridge University Press.

* Jensen, F. V. (2001). *Bayesian Networks and Decision Graphs.* NY: Springer-Verlag.

* Jordan, M. I., Ghahramani, Z., Jaakkola, T. S., & Saul, L. K. (1999). An introduction to variational methods for graphical models. *Machine Learning*, *37*(2), 183–233.

* Jordan, M. I. (Ed.). (1999). *Learning in Graphical Models*, Cambridge, MA: MIT Press.

Kschischang, F. R., Frey, B. J., & Loeliger, H.-A. (2001). Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory 47*, 498-519.

Lauritzen, S. L. (1996). *Graphical Models*. Oxford: Oxford University Press.

McEliece, R. J., MacKay, D. J. C. & Cheng, J.-F. (1996). Turbo decoding as an instance of Pearl's "belief propagation algorithm." *IEEE Journal on Selected Areas in Communication*, *16*, 140–152.

* Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Mateo, CA: Morgan Kaufmann.

Shafer, G. R., & Shenoy, P. P. (1990). Probability propagation. *Annals of Mathematics and Artificial Intelligence, 2*, 327-352.

Yedidia, J.S. Freeman, W.T., & Weiss, Y. (2001). Bethe free energies, Kikuchi approximations, and belief propagation algorithms. MERL Technical Report 2001-16.