The Abdus Salam International Centre for Theoretical Physics

United Nations Educational, Scientific and Cultural Organization

IAEA International Atomic Energy Agency

2361-13

School on Large Scale Problems in Machine Learning and Workshop on Common Concepts in Machine Learning and Statistical Physics

20 - 31 August 2012

BACKGROUND INFORMATION FOR LECTURES ON VARIATIONAL METHODS, DUALITY AND MESSAGE-PASSING "Graphical Models and Message-passing Algorithms: Some Introductory Lectures"

Martin WAINWRIGHT
Department of Statistics and EECS
UC Berkeley CA
U.S.A.

Graphical models and message-passing algorithms: Some introductory lectures

Martin J. Wainwright

1 Introduction

Graphical models provide a framework for describing statistical dependencies in (possibly large) collections of random variables. At their core lie various correspondences between the conditional independence properties of a random vector, and the structure of an underlying graph used to represent its distribution. They have been used and studied within many sub-disciplines of statistics, applied mathematics, and computer science, including statistical machine learning and artificial intelligence, communication and information theory, statistical physics, network control theory, computational biology, statistical signal processing, and computer vision.

The purpose of these notes is to provide an introduction to the basic material of graphical models and associated message-passing algorithms. We assume only that the reader has undegraduate-level background in linear algebra, probability theory (without needing measure theory), and some basic graph theory.

These introductory lectures should be viewed as complementary to the monograph [12], which focuses primarily on some more advanced aspects of the theory and methodology of graphical models.

2 Probability distributions and graphical structure

In this section, we define various types of graphical models, and discuss some of their properties. Before doing so, let us introduce the basic probabilistic notation used throughout these notes. Any graphical model corresponds to a family of probability distributions over a random vector $X = (X_1, \ldots, X_m)$. Here for each $i \in [m] = \{1, 2, \ldots, m\}$, the random variable X_i take values in some space $\mathcal{X}_i$, which

Martin J. Wainwright
Department of Statistics, UC Berkeley, Berkeley, CA 94720e-mail: wainwrig@stat.berkeley.edu

(depending on the application) may either be continuous (e.g., $\mathcal{X}_i = \mathbb{R}$) or discrete (e.g., $\mathcal{X}_i = \{0, 1, \ldots, k-1\}$). Lower case letters are used to refer to particular elements of $\mathcal{X}_i$, so that the notation $\{X_i = x_i\}$ corresponds to the event that the random variable X_i takes the value $x_i \in \mathcal{X}_i$. The random vector $X = (X_1, X_2, \ldots, X_m)$ takes values in the Cartesian product space $\mathcal{X}_1 \times \mathcal{X}_2 \times \ldots \times \mathcal{X}_m$. For any subset $A \subseteq [m]$, we define the subvector $X_A := (X_s,\ s \in A)$, corresponding to a random vector that takes values in the space $\mathcal{X}_A$. We use the notation $x_A := (x_s,\ s \in A)$ to refer to a particular element of the space $\mathcal{X}_A$. With this convention, note that $\mathcal{X}_{[m]}$ is shorthand notation for the Cartesian product $\mathcal{X}_1 \times \mathcal{X}_2 \times \ldots \times \mathcal{X}_m$.

2.1 Directed graphical models

We begin our discussion with directed graphical models, which (not suprisingly) are based on the formalism of directed graphs. In particular, a directed graph $D = (\mathcal{V}, \vec{\mathcal{E}})$ consists of a vertex set $\mathcal{V} = \{1, \ldots, m\}$ and a collection $\vec{\mathcal{E}}$ of directed pairs $(i \to j)$, meaning that i is connected by an edge directed to j. When there exists a directed edge $(j \to i) \in \mathcal{E}$, we say that node i is a child of node j, and conversely that node j is a parent of node i. We use the notation $\pi(i)$ to denote the set of all parents of node i (which might be an empty set). A directed cycle is a sequence of vertices $(i_1, i_2, \ldots, i_\ell)$ such that $(i_\ell \to i_1) \in \vec{\mathcal{E}}$, and $(i_k \to i_{k+1}) \in \vec{\mathcal{E}}$ for all $k = 1, \ldots, \ell - 1$. A *directed acyclic graph*, or DAG for short, is a directed graph that contains no directed cycles.

Any mapping $\rho : [m] \to [m]$ defines a particular ordering of the vertex set $\mathcal{V} = \{1, 2, \ldots, m\}$, and of interest to us are particular orderings of the vertex set of a DAG.

Definition 1. The ordering $\{\rho(1), \ldots, \rho(m)\}$ of the vertex set $\mathcal{V}$ is *topological* if for each $i \in \mathcal{V}$, we have $\rho(j) < \rho(i)$ for all $j \in \pi(i)$.

Alternatively stated, in a topological ordering, children always come after their parents. It is an elementary fact of graph theory that any DAG has at least one topological ordering, and this fact plays an important role in our analysis of directed graphical models. So as to lighten notation, we typically assume that the canonical ordering $\mathcal{V} = \{1, 2, \ldots, m\}$ is topological. Note that in this case, vertex 1 cannot have any parents (i.e., $\pi(1) = \emptyset$), and moreover vertex m cannot have any children.

A directed graphical model is obtained by associating each node i of a DAG with a random variable X_i, and requiring the joint probability distribution over $(X_1, \ldots, X_m)$ factorize according to the DAG. Consider the subset of vertices $(i, \pi(i))$ corresponding to a given vertex i and its parents $\pi(i)$. We may associate with this

subset a real-valued function $f_i : \mathcal{X}_{(i,\pi(i))} \to \mathbb{R}_+$ that maps any given configuration $(x_i, x_{\pi(i)}) \in \mathcal{X}_{(i,\pi(i))}$ to a real number $f_i(x_i, x_{\pi(i)}) \geq 0$. We assume moreover that f_i satisfies the normalization condition

$$\sum_{x_i} f_i(x_i, x_{\pi(i)}) = 1 \quad \text{for all } x_{\pi(i)} \in \mathcal{X}_{\pi(i)}. \tag{1}$$

With this set-up, the *directed graphical model* based on a given DAG D is the collection of probability distributions over the random vector $(X_1, \ldots, X_m)$ that have a factorization of the form

$$p(x_1, \ldots, x_m) = \frac{1}{Z} \prod_{i=1}^{m} f_i(x_i, x_{\pi(i)}), \tag{2}$$

for some choice of the parent-child functions $(f_1, \ldots, f_m)$ that satisfy the normalization condition (1). In the factorization (2), the quantity Z denotes a constant chosen to ensure that p sums to one.

Proposition 1. *For any directed acyclic graph D, the factorization* (2) *with $Z = 1$ defines a valid probability distribution such that $f_i(x_i, x_{\pi(i)}) = p(x_i \,|\, x_{\pi(i)})$ for all $i \in \mathcal{V}$.*

Proof. We proceed via induction on the number of nodes m. For $m = 1$, the claim is trivial. Let us assume that it is true for all DAGs with at most m nodes, and prove that it must hold for any DAG with $m+1$ nodes. Given a DAG with $m+1$ nodes, we assume without loss of generality (re-indexing as necessary) that $\{1, 2, \ldots, m+1\}$ is a topological ordering. Given such an ordering, it must be the case that x_{m+1} has no children, so that we may write

$$p(x_1, \ldots, x_m, x_{m+1}) = \frac{1}{Z} \Big[\prod_{i=1}^{m} f_i(x_i, x_{\pi(i)}) \Big] \, f_{m+1}(x_{m+1}, x_{\pi(m+1)}).$$

Marginalizing over x_{m+1} yields that

$$\begin{aligned} p(x_1, \ldots, x_m) &= \frac{1}{Z} \Big[\prod_{s=1}^{m} f(x_s, x_{\pi(s)}) \Big] \Big[\sum_{x_{m+1}} f_{m+1}(x_{m+1}, x_{\pi(m+1)}) \Big] \\ &= \frac{1}{Z} \prod_{s=1}^{m} f_i(x_i, x_{\pi(i)}), \end{aligned}$$

where we have used the facts that x_{m+1} appears only in one term (since $m+1$ is a leaf node), and that $\sum_{x_{m+1}} f_{m+1}(x_{m+1}, x_{\pi(m+1)}) = 1$. In graphical terms, marginalizing over x_{m+1} allows us to remove $m+1$ from the DAG, so that we have reduced the

problem to a DAG with m nodes. By the induction hypothesis, we have $Z = 1$, and $f_i(x_i, x_{\pi(i)}) = p(x_i \,|\, x_{\pi(i)})$ for all $i = 1, \ldots, m$. Moreover, we have

$$\frac{p(x_1, x_2, \ldots, x_{m+1})}{p(x_1, \ldots, x_m)} = f_{m+1}(x_{m+1}, x_{\pi(m+1)}) \quad \text{for all } (x_1, \ldots, x_{m+1}),$$

which shows that $p(x_{m+1} \,|\, x_1, \ldots, x_m)$ depends only on the subvector $(x_{m+1}, x_{\pi(m+1)})$. We conclude that $p(x_{m+1} \,|\, x_{\pi(m+1)})$ is proportional to $f_{m+1}(x_{m+1}, x_{\pi(m+1)})$, and the normalization condition on f_{m+1} implies that equality holds, thereby completing the proof. □

in the factorization (2) have a concrete interpretation as the child-parent conditional probability tables (i.e., $f_i(x_i, x_{\pi(i)})$ is equal to the conditional probability of $X_i = x_i$ given that $X_{\pi(i)} = x_{\pi(i)}$). This local interpretability, which (as we will see) is not shared by the class of undirected graphical models, has some important consequences. For instance, sampling a configuration $(\tilde{X}_1, \ldots, \tilde{X}_m)$ from any DAG model is straightforward: we first sample $\tilde{X}_1 \sim f_1(\cdot)$, and then for $i = 2, \ldots, m$, sample $\tilde{X}_i \sim f_i(\cdot, \tilde{X}_{\pi(i)})$. This procedure is well-specified: due due to the topological ordering, we are guaranteed that the variable $\tilde{X}_{\pi(i)}$ has been sampled before we move on to sampling $\tilde{X}_i$. Moveover, by construction, the random vector $(\tilde{X}_1, \ldots, \tilde{X}_m)$ is distributed according to the probability distribution (2).

2.2 Undirected graphical models

We now turn to discussion of undirected graphical models, which are also known as *Markov random fields* or *Gibbs distributions*. Naturally, these models are built using an undirected graphs, by which we mean a pair $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{1, \ldots, m\}$ is the vertex set (as before), and $\mathcal{E}$ is a collection of undirected edges, meaning that there is no distinction between the edge (i, j) and the edge (j, i).

Definition 2 (Markov property). A random vector $(X_1, \ldots, X_m)$ is *Markov with respect to a graph* $\mathcal{G}$ means that for all vertex cutsets S and associated components A and B, the conditional independence condition $X_A \perp\!\!\!\perp X_B \mid X_C$ holds.

A *clique* C of an undirected graph $\mathcal{G}$ is a fully connected subset C of the vertex set $\mathcal{V}$ (i.e., $(s, t) \in \mathcal{E}$ for all $s, t \in C$). A clique is *maximal* if it is not contained within any other clique.

Definition 3 (Factorization). We say that a random vector $(X_1, \ldots, X_m)$ factorizes over the graph $\mathcal{G}$ if its probability distribution can be written as

$$p(x_1, \ldots, x_m) = \frac{1}{Z} \prod_{C \in \mathfrak{C}} \psi_C(x_C), \tag{3}$$

corresponding to a factorization in terms of compatibility functions ψ_C associated with each clique $C \in \mathfrak{C}$ of the graph.

2.3 Factor Graphs

For large graphs, the factorization properties of a graphical model, whether undirected or directed, may be difficult to visualize from the usual depictions of graphs. The formalism of *factor graphs* provides an alternative graphical representation, one which emphasizes the factorization of the distribution [6, 8].

Let $\mathcal{F}$ represent an index set for the set of factors defining a graphical model distribution. In the undirected case, this set indexes the collection C of cliques, while in the directed case $\mathcal{F}$ indexes the set of parent–child neighborhoods. We then consider a bipartite graph $\mathcal{G} = (\mathcal{V}, \mathcal{F}, \mathcal{E})$, in which $\mathcal{V}$ is (as before) an index set for the variables, and $\mathcal{F}$ is an index set for the factors. Given the bipartite nature of the graph, the edge set $\mathcal{E}$ now consists of pairs (i, a) of nodes $i \in \mathcal{V}$ such that the fact $a \in N(i)$, or equivalently such that $a \in N(i)$. See Figure 1(b) for an illustration.

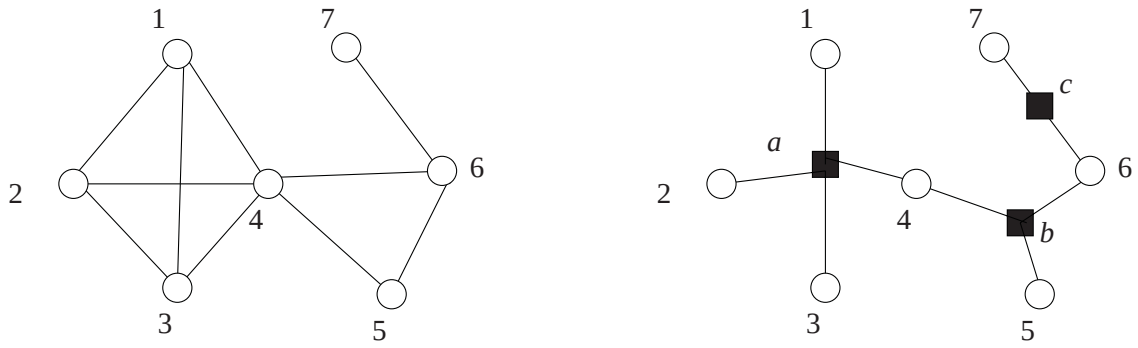

Fig. 1. Illustration of undirected graphical models and factor graphs. (a) An undirected graph on $m = 7$ vertices, with maximal cliques $\{1,2,3,4\}$, $\{4,5,6\}$ and $\{6,7\}$. (b) Equivalent representation of the undirected graph in (a) as a factor graph, assuming that we define compatibility functions only on the maximal cliques in (a). The factor graph is a bipartite graph with vertex set $\mathcal{V} = \{1, \ldots, 7\}$ and factor set $F = \{a, b, c\}$, one for each of the compatibility functions of the original undirected graph.

For undirected models, the factor graph representation is of particular value when C consists of more than the maximal cliques. Indeed, the compatibility functions for the nonmaximal cliques do not have an explicit representation in the usual representation of an undirected graph — however, the factor graph makes them explicit.

2.4 Equivalence between factorization and Markov properties

This section is devoted to a classical result that establishes a certain equivalence between factorization and Markov properties. For strictly positive distributions p, this equivalence is complete.

Theorem 1 (Hammersley-Clifford). *If a random vector $(X_1, \ldots, X_m)$ factorizes over a graph $\mathcal{G}$, then it is Markov with respect to $\mathcal{G}$. Conversely, if a random vector is Markov with respect to $\mathcal{G}$* and *$p(x) > 0$ for all $x \in \mathcal{X}^m$, then it factorizes over the graph.*

Proof. We begin by proving that (**F**) $\Longrightarrow$ (**M**). Suppose that the factorization (3) holds, and let S be an arbitrary vertex cutset of the graph such that subsets A and B are separated by S. We may assume without loss of generality that both A and B are non-empty, and we need to show that $X_A \perp\!\!\!\perp X_B \mid X_S$. Let us define subsets of cliques by $\mathfrak{C}_A := \{C \in \mathfrak{C} \mid C \cap A \neq \emptyset\}$, $\mathfrak{C}_B := \{C \in \mathfrak{C} \mid C \cap B \neq \emptyset\}$, and $\mathfrak{C}_S := \{C \in \mathfrak{C} \mid C \subseteq S\}$. We claim that these three subsets form a disjoint partition of the full clique set—namely, $\mathfrak{C} = \mathfrak{C}_A \cup \mathfrak{C}_S \cup \mathfrak{C}_B$. Given any clique C, it is either contained entirely within S, or must have non-trivial intersection with either A or B, which proves the union property. To establish disjointness, it is immediate that $\mathfrak{C}_S$ is disjoint from $\mathfrak{C}_A$ and $\mathfrak{C}_B$. On the other hand, if there were some clique $C \in \mathfrak{C}_A \cap \mathfrak{C}_B$, then there would exist nodes $a \in A$ and $b \in B$ with $\{a, b\} \in C$, which contradicts the fact that A and B are separated by the cutset S.

Consequently, we may write

$$p(x_A, x_S, x_B) = \frac{1}{Z} \underbrace{[\prod_{C \in \mathfrak{C}_A} \psi_C(x_C)]}_{\Psi_A(x_A, x_S)} \underbrace{[\prod_{C \in \mathfrak{C}_S} \psi_C(x_C)]}_{\Psi_S(x_S)} \underbrace{[\prod_{C \in \mathfrak{C}_B} \psi_C(x_C)]}_{\Psi_B(x_B, x_S)}.$$

Defining the quantities

$$Z_A(x_S) := \sum_{x_A} \Psi_A(x_A, x_S), \quad \text{and} \quad Z_B(x_S) := \sum_{x_B} \Psi_B(x_B, x_S),$$

we then obtain the following expresssions for the marginal distributions of interest

$$p(x_S) = \frac{Z_A(x_S)\, Z_B(x_S)}{Z}\, \Psi_S(x_S) \quad \text{and} \quad p(x_A, x_S) = \frac{Z_B(x_S)}{Z}\, \Psi_A(x_A, x_S)\, \Psi_S(x_S),$$

with a similar expression for $p(x_B, x_S)$. Consequently, for any x_S for which $p(x_S) > 0$, we may write

$$\begin{aligned} \frac{p(x_A, x_S, x_B)}{p(x_S)} &= \frac{\frac{1}{Z}\, \Psi_A(x_A, x_S)\Psi_S(x_S)\Psi_B(x_B, x_S)}{\frac{Z_A(x_S)\, Z_B(x_S)}{Z}\, \Psi_S(x_S)} \\ &= \frac{\Psi_A(x_A, x_S)\Psi_B(x_B, x_S)}{Z_A(x_S)\, Z_B(x_S)}. \end{aligned} \tag{4}$$

Similar calculations yield the relations

$$\frac{p(x_A, x_S)}{p(x_S)} = \frac{\frac{Z_B(x_S)}{Z}\, \Psi_A(x_A, x_S)\, \Psi_S(x_S)}{\frac{Z_A(x_S) Z_B(x_S)}{Z}\, \Psi_S(x_S)} = \frac{\Psi_A(x_A, x_S)}{Z_A(x_S)}, \qquad \text{and} \tag{5a}$$

$$\frac{p(x_B, x_S)}{p(x_S)} = \frac{\frac{Z_A(x_S)}{Z}\, \Psi_B(x_B, x_S)\, \Psi_S(x_S)}{\frac{Z_A(x_S) Z_B(x_S)}{Z}\, \Psi_S(x_S)} = \frac{\Psi_B(x_B, x_S)}{Z_B(x_S)}. \tag{5b}$$

Combining equation (4) with equations (5a) and (5b) yields

$$p(x_A, x_B \mid x_S) = \frac{p(x_A, x_B, x_S)}{p(x_S)} = \frac{p(x_A, x_S)}{p(x_S)}\, \frac{p(x_B, x_S)}{p(x_S)} = p(x_A \mid x_S)\, p(x_B \mid x_S),$$

thereby showing that $X_A \perp\!\!\!\perp X_B \mid X_S$, as claimed.

In order to prove the converse implication (**M**) $\Longrightarrow$ (**F**), we require a version of the *inclusion-exclusion formula*. Given a set $[m] = \{1, 2, \ldots, m\}$, let $\mathcal{P}([m])$ its power set, meaning the set of all subsets of $[m]$. With this notation, the following inclusion-exclusion formula is classical:

Lemma 1 (Inclusion-exclusion). *For any two real-valued functions Ψ and Φ defined on the power set $\mathcal{P}([m])$, the following statements are equivalent:*

$$\Phi(A) = \sum_{B \subseteq A} (-1)^{|A \setminus B|}\, \Psi(B) \quad \textit{for all } A \in \mathcal{P}([m]). \tag{6a}$$

$$\Psi(A) = \sum_{B \subseteq A} \Phi(B) \quad \textit{for all } A \in \mathcal{P}([m]). \tag{6b}$$

Proofs of this result can be found in standard texts in combinatorics (e.g., [7]). Returning to the main thread, let $y \in \mathcal{X}^m$ be some fixed element, and for each subset $A \in \mathcal{P}([m])$, define the function $\phi : \mathcal{X}^m \to \mathbb{R}$ via

$$\phi_A(x) := \sum_{B \subseteq A} (-1)^{|A \setminus B|} \log \frac{p(x_B, y_{B^c})}{p(y)}. \tag{7}$$

(Note that taking logarithms is meaningful since we have assumed $p(x) > 0$ for all $x \in \mathcal{X}^m$). From the inclusion-exclusion formula, we have $\log \frac{p(x_A, y_{A^c})}{p(y)} = \sum_{B \subseteq A} \phi_B(x)$, and setting $A = [m]$ yields

$$p(x) = p(y) \exp\{ \sum_{B \in \mathcal{P}([m])} \phi_B(x) \}. \tag{8}$$

From the definition (7), we see that ϕ_A is a function only of x_A. In order to complete the proof, it remains to show that $\phi_A = 0$ for any subset A that is *not* a graph clique. As an intermediate result, we claim that for any $t \in A$, we can write

$$\phi_A(x) = \sum_{B \subseteq A \setminus \{t\}} (-1)^{|A - B|} \log \frac{p(x_t \,|\, x_B, y_{B^c \setminus \{t\}})}{p(y_t \,|\, x_B, y_{B^c \setminus \{t\}})}. \tag{9}$$

To establish this claim, we write

$$\begin{aligned}
\phi_A(x) &= \sum_{\substack{B \subset A \\ B \not\ni t}} (-1)^{|A \setminus B|} \log \frac{p(x_B, y_{B^c})}{p(y)} + \sum_{\substack{B \subset A \\ B \ni t}} (-1)^{|A \setminus B|} \log \frac{p(x_B, y_{B^c})}{p(y)} \\
&= \sum_{B \subseteq A \setminus \{t\}} (-1)^{|A \setminus B|} \left\{ \log \frac{p(x_B, y_{B^c})}{p(y)} - \log \frac{p(x_{B \cup t}, y_{B^c \setminus \{t\}})}{p(y)} \right\} \\
&= \sum_{B \subseteq A \setminus \{t\}} (-1)^{|A \setminus B|} \log \frac{p(x_B, y_{B^c})}{p(x_{B \cup t}, y_{B^c \setminus \{t\}})}
\end{aligned} \tag{10}$$

Note that for any $B \subseteq A \setminus \{t\}$, we are guaranteed that $t \notin B$, whence

$$\frac{p(x_B, y_{B^c})}{p(x_{B \cup t}, y_{B^c \setminus \{t\}})} = \frac{p(y_t \,|\, x_B, y_{B^c \setminus \{t\}})}{p(x_t \,|\, x_B, y_{B^c \setminus \{t\}})}.$$

Substituting into equation (10) yields the claim (9).

We can now conclude the proof. If A is not a clique, then there must some exist some pair (s,t) of vertices *not* joined by an edge. Using the representation (9), we can write ϕ_A as a sum of four terms (i.e., $\phi_A = \sum_{i=1}^4 T_i$), where

$$T_1(x) = \sum_{B \subseteq A \setminus \{s,t\}} (-1)^{|A \setminus B|} \log p(y_t \,|\, x_B, y_{B^c \setminus \{t\}}),$$

$$T_2(x) = \sum_{B \subseteq A \setminus \{s,t\}} (-1)^{|A \setminus (B \cup \{t\})|} \log p(x_t \,|\, x_B, y_{B^c \setminus \{t\}})$$

$$T_3(x) = \sum_{B \subseteq A \setminus \{s,t\}} (-1)^{|A \setminus (B \cup \{s\})|} \log p(y_t \,|\, x_{B \cup \{s\}}, y_{B^c \setminus \{s,t\}}), \quad \text{and}$$

$$T_4(x) = \sum_{B \subseteq A \setminus \{s,t\}} (-1)^{|A \setminus (B \cup \{s,t\})|} \log p(x_t \,|\, x_{B \cup \{s\}}, y_{B^c \setminus \{s,t\}}).$$

Combining these separate terms, we obtain

$$\phi_A(x) = \sum_{B \subseteq A \setminus \{s,t\}} (-1)^{|A \setminus B|} \log \frac{p(y_t \,|\, x_B, y_{B^c \setminus \{t\}}) p(x_t \,|\, x_{B \cup \{s\}}, y_{B^c \setminus \{s,t\}})}{p(x_t \,|\, x_B, y_{B^c \setminus \{t\}}) p(y_t \,|\, x_{B \cup \{s\}}, y_{B^c \setminus \{s,t\}})}.$$

But using the Markov properties of the graph, each term in this sum is zero. Indeed, since $s \notin N(t)$, we have

$$p(x_t \,|\, x_B, y_{B^c \setminus \{t\}}) = p(x_t \,|\, x_{B \cup \{s\}}, y_{B^c \setminus \{s,t\}}), \quad \text{and} \quad p(y_t \,|\, x_B, y_{B^c \setminus \{t\}}) = p(y_t \,|\, x_{B \cup \{s\}}, y_{B^c \setminus \{s,t\}}),$$

which completes the proof. □

3 Exact algorithms for inference

In applications of graphical models, one is typically interested in addressing some kind of statistical inference problem, in which one uses partial or noisy observations to draw inferences about the underlying model. Approaching any such statistical inference problem typically requires solving one of a core set of computational problems. This set of core problems includes *likelihood computation*, which arises (for instance) as a part of hypothesis testing or parameter estimation. A closely related problem is that of computing the marginal distribution $p(x_A)$ over a particular subset $A \subset \mathcal{V}$ of nodes, or similarly, computing the conditional distribution $p(x_A \mid x_B)$, for disjoint subsets A and B of the vertex set. This *marginalization problem* is needed to in order to perform filtering or smoothing of time series (for chain-structured graphs), or the analogous operations for more general graphical models. A third problem is that of *mode computation*, in which the goal is to find an element $\widehat{x}$ in the set $\arg\max_{x \in \mathcal{X}^m} p(x)$.

At first sight, these problems—-likelihood computation, marginalization and mode computation—might seem quite simple and indeed, for small graphical models (e.g., $m \approx 10$—20), they can be solved directly with brute force approaches. However, many applications involve graphs with thousands (if not millions) of nodes, and the computational complexity of brute force approaches scales very poorly in the graph size. More concretely, let us consider the case of a discrete random vector $(X_1, \ldots, X_m) \in \mathcal{X}^m$ such that, for each node $s \in \mathcal{V}$, the random variable X_s

takes values in the state space $\mathcal{X}_s = \{0, 1, \ldots, k-1\}$. A naive approach to computing a marginal at a single node — say $p(x_s)$ — entails summing over all configurations of the form $\{x' \in \mathcal{X}^m \mid x'_s = x_s\}$. Since this set has k^{m-1} elements, it is clear that a brute force approach will rapidly become intractable. Given a graph with $m = 100$ vertices (a relatively small problem) and binary variables ($k = 2$), the number of terms is 2^{99}, which is already larger than estimates of the number of atoms in the universe. This is a vivid manifestation of the "curse-of-dimensionality" in a computational sense.

A similar curse applies to the problem of mode computation for discrete random vectors, since it is equivalent to solving an integer programming problem. Indeed, the mode computation problem for Markov random fields includes many well-known instances of NP-complete problems (e.g., 3-SAT, MAX-CUT etc.) as special cases. Unfortunately, for continuous random vectors, the problems are no easier and typically harder, since the marginalization problem involves computing a high-dimensional integral, whereas mode computation corresponds to a generic (possibly non-convex) optimization problem. An important exception to this statement is the Gaussian case where the problems of both marginalization and mode computation can be solved in polynomial-time for any graph (via matrix inversion), and in linear-time for tree-structured graphs via the Kalman filter.

In the following sections, we develop various approaches to these computational inference problems, beginning with discussion of a relatively naive but pedagogically useful scheme known as elimination, moving onto more sophisticated message-passing algorithms, and culminating in our derivation of the junction tree algorithm.

3.1 Elimination algorithm

Vertex elimination algorithm

1. Initialize all vertices marked as undeleted, and the edge set $\mathcal{E}$. Choose a permutation of the vertex set $\mathcal{V}$ corresponding to the order in which to perform elimination.
2. For $i = 1, 2, \ldots, m$:
   (a) Set current vertex $k = \rho(i)$.
   (b) Augment the current edge set with the collection of edges $\{(j,k), \quad \forall\ j \in N(k)\}$.
3. Return the augmented graph $\widetilde{\mathcal{G}} = (\mathcal{V}, \widetilde{\mathcal{E}})$ that contains the original edge set $\mathcal{E}$ and all edges added during the procedure.

3.2 Message-passing algorithms on trees

We now turn to discussion of message-passing algorithms for models based on graphs without cycles, also known as trees. So as to highlight the essential ideas in the derivation, we begin by deriving message-passing algorithms for trees with only pairwise interactions, in which case the distribution has form

$$p(x_1, \ldots, x_m) = \frac{1}{Z} \prod_{i \in \mathcal{V}} \psi_i(x_i) \prod_{(i,j) \in \mathcal{E}} \psi_{ij}(x_i, x_j), \tag{11}$$

where $T = (\mathcal{V}, \mathcal{E})$ is a given tree, and $\{\psi_i, i \in \mathcal{V}\}$ and $\{\psi_{ij}, (i,j) \in \mathcal{E}\}$ are compatibility functions. (In Section 3.2.2, we provide extensions to general factor trees that allow for higher-order interactions.) For a tree, the elimination algorithm takes a very simple form—assuming that the appropriate ordering is followed. Recall that a node in a graph is called a *leaf* if it has degree one.

The key fact is that for any tree, it is always possible to choose an elimination ordering $\rho(\mathcal{V}) = \{\rho(1), \rho(2), \ldots, \rho(m)\}$ such that:

1. The vertex $\rho(1)$ is a leaf node of the original graph $\mathcal{G}$, and
2. For $t = 2, 3, \ldots, m-1$, the vertex $\rho(t)$ is a leaf node of reduced graph $\mathcal{G}_t$ at step t of the elimination algorithm.

This claim follows because any tree always has at least one leaf node [4], so that the algorithm can be initialized in Step 1. Moreover, an inductive argument shows that the elimination algorithm will never introduce any additional compatibility functions into the factorization, so that the graph $\mathcal{G}_t$ at each stage of elimination remains a tree, and we can always find a leaf node to continue the process. We refer to such an ordering as a *leaf-exposing ordering*.

In computational terms, at round $t = 1, \ldots, m-1$, the relevant leaf node is $v = \rho(t)$, so the algorithm marginalizes over x_v, and then passes the result to its unique parent $\pi(v)$. The result of this intermediate computation is a function of $x_{\pi(v)}$, so that we can represent it by a "message" of the form $M_{v \to \pi(v)}$ that provides one number for each possible setting of $x_{\pi(v)}$. Upon termination, the elimination algorithm will return the marginal distribution $p(x_r)$ at the root node $r \in \mathcal{V}$; more specifically, this marginal is specified in terms of the local compatiblity function ψ_r and the incoming messages as follows:

$$p(x_r) \propto \psi_r(x_r) \prod_{i \in N(r)} M_{i \to r}(x_r).$$

Assuming that each variable takes at most k values (i.e., $|\mathcal{X}_i| \leq k$ for all $i \in \mathcal{V}$), the overall complexity of running the elimination algorithm for a given root node scales as $O(k^2 m)$, since computing each message amounts to summing k numbers a total of k times, and there are m rounds in the elimination order.

3.2.1 Sum-product algorithm

In principle, by running the tree-based elimination algorithm m times—once for each node acting as the root in the elimination ordering—we could compute all singleton marginals in time $O(k^2m^2)$. However, as the attentive reader might suspect, such an approach is wasteful, since it neglects to consider that many of the intermediate operations in the elimination algorithm would be shared between different orderings. Herein lies the cleverness of the sum-product algorithm: it re-uses these intermediate results in the appropriate way, and thereby reduces the overall complexity of computing all singleton marginals to time $O(k^2m)$. For each edge $(i, j) \in \mathcal{E}$, the algorithm maintains two messages—$M_{i\to j}$ and $M_{j\to i}$—corresponding to two possible directions of the edge.[1] With this notation, the sum-product message-passing algorithm takes the following form:

Sum-product message-passing algorithm (pairwise tree):

1. At iteration $t = 0$:
   For all $(i, j) \in \mathcal{E}$, initialize messages

$$M^0_{j\to i}(x_i) = 1 \qquad \text{for all } x_i \in \mathcal{X}_i, \text{ and}$$
$$M^0_{i\to j}(x_j) = 1 \qquad \text{for all } x_j \in \mathcal{X}_j.$$

2. For iterations $t = 1, 2, \ldots$:
   (i) For each $(j, i) \in \mathcal{E}$, update messages:

$$M^t_{j\to i}(x_i) \leftarrow \kappa \sum_{x'_j} \Big\{ \psi_{ij}(x_i, x'_j)\psi_j(x'_j) \prod_{k\in N(j)/i} M^{t-1}_{k\to j}(x'_j) \Big\}, \qquad (12)$$

   where $\kappa > 0$ chosen such that $\sum_{x_i} M^t_{j\to i}(x_i) = 1$.
   (ii) If desired, compute current estimates of marginals

$$\mu^t(x_i) = \kappa_i\, \psi_i(x_i) \prod_{j\in N(i)} M^t_{j\to i}(x_i), \quad \text{and} \qquad (13a)$$

$$\mu^t_{ij}(x_i, x_j) = \kappa_{ij}\, \psi_{ij}(x_i, x_j) \prod_{k\in N(i)\setminus\{j\}} M^t_{k\to i}(x_i) \prod_{k\in N(j)\setminus\{i\}} M^t_{k\to j}(x_j), \qquad (13b)$$

   where $\kappa_i > 0$ and $\kappa_{ij} > 0$ are normalization constants chosen such that $\sum_{x_i} \mu^t_i(x_i) = 1 = \sum_{x_i,x_j} \mu^t_{ij}(x_i, x_j)$.

The initialization given in Step 1 is the standard "uniform" one; we note that the final output of the algorithm will be same for any initialization of the messages that

[1] Recall that the message $M_{i\to j}$ is a vector of $|\mathcal{X}_j|$ numbers, one for each value $x_j \in \mathcal{X}_j$.

has strictly positive components. The following result states the convergence and correctness guarantees associated with the sum-product algorithm on a tree:

Proposition 2. *For any tree T with diameter $d(T)$, the sum-product updates have a unique fixed point M^*. The algorithm converges to it after at most $d(T)$ iterations, and the marginals obtained by equations* (13a) *and* (13b) *are exact.*

Proof. We proceed via induction on the number of nodes. For $m = 1$, the claim is trivial. Now assume that the claim holds for all trees with at most $m-1$ nodes, and let us show that it also holds for any tree with m nodes. It is an elementary fact of graph theory [4] that any tree has at least one leaf node. By re-indexing as necessary, we may assume that node m is a leaf, and its unique parent is node 1. By definition of the sum-product updates, for all iterations $t = 1, 2, \ldots$, the message sent from m to 1 is given by

$$M^*_{m\to 1}(x_1) = \kappa \sum_{x_m} \psi_m(x_m)\psi_{m1}(x_m, x_1).$$

Given this fixed message, we may consider the probability distribution defined on variables $(x_1, \ldots, x_{m-1})$ given by

$$p(x_1, \ldots, x_{m-1}) \propto M^*_{m\to 1}(x_1) \Big[\prod_{i=1}^{m-1} \psi_i(x_i)\Big] \prod_{(i,j)\in\mathcal{E}\setminus\{(1,m)\}} \psi_{ij}(x_i, x_j).$$

Our notation is consistent, in that $p(x_1, \ldots, x_{m-1})$ is the marginal distribution obtained by summing out x_m from the original problem. It corresponds to an instance of our original problem on the new tree $T' = (\mathcal{V}', \mathcal{E}')$ with $\mathcal{V}' = \{1, \ldots, m-1\}$ and $\mathcal{E}' = \mathcal{E}\setminus\{(1,m)\}$. Since the message $M^*_{m\to 1}$ remains fixed for all iterations $t \geq 1$, the iterates of the sum-product algorithm on T' are indistinguishable from the iterates on tree (for all nodes $i = 1, \ldots, m-1$ and edges $(i, j) \in \mathcal{E}'$).

By the induction hypothesis, the sum-product algorithm applied to T' will converge after at most $d(T')$ iterations to a fixed point $M^* = \{M^*_{i\to j}, M^*_{j\to i} \mid (i, j) \in \mathcal{E}'\}$, and this fixed point will yield the correct marginals at all nodes $i = 1, \ldots, m-1$. As previously discussed, the message from m to 1 remains fixed for all iterations after the first, and the message from 1 to m will be fixed once the iterations on the sub-tree T' have converged. Consequently, we include the extra iteration for the messages between m and 1, we conclude that sum-product on T converges in at most $d(T) \leq d(T') + 1$ iterations as claimed.

It remains to show that the marginal at node m is proportional to the quantity $\mu_m(x_m) \propto \psi_m(x_m) M^*_{1\to m}(x_m)$, where

$$M^*_{1\to m}(x_m) := \kappa \sum_{x_1} \psi_1(x_1)\, \psi_{1m}(x_1, x_m) \prod_{k \in N(1)\setminus\{m\}} M^*_{k\to 1}(x_1).$$

By elementary probability theory, we have $p(x_m) = \sum_{x_1} p(x_m \mid x_1) p(x_1)$. Since m is a leaf node with unique parent 1, we have the conditional independence relation $X_m \perp\!\!\!\perp X_{\mathcal{V}\setminus\{1\}} \mid X_1$, and hence

$$p(x_m \mid x_1) = \frac{\psi_m(x_m)\psi_{1m}(x_1, x_m)}{\sum_{x_m} \psi_m(x_m)\psi_{1m}(x_1, x_m)} \propto \frac{\psi_m(x_m)\psi_{1m}(x_1, x_m)}{M^*_{m\to 1}(x_1)}. \tag{14}$$

By the induction hypothesis, the sum-product algorithm, when applied to the distribution (14), returns the marginal distribution

$$p(x_1) \propto [M^*_{m\to 1}(x_1)\psi_1(x_1)] \prod_{k \in N(1)\setminus\{m\}} M^*_{k\to 1}(x_1).$$

Combining the pieces yields

$$\begin{aligned} p(x_m) &= \sum_{x_1} p(x_m \mid x_1) p(x_1) \\ &\propto \psi_m(x_m) \sum_{x_1} \frac{\psi_{1m}(x_1, x_m)}{M^*_{m\to 1}(x_1)} [M^*_{m\to 1}(x_1)\psi_1(x_1)] \prod_{k \in N(1)\setminus\{m\}} M^*_{k\to 1}(x_1) \\ &\propto \psi_m(x_m)\, M^*_{1\to m}(x_m), \end{aligned}$$

as required. A similar argument yields that equation (13b) yields the correct form of the pairwise marginals for each edge $(i, j) \in \mathcal{E}$; we leave the details as an exercise for the reader. □

3.2.2 Sum-product on general factor trees

In the preceding section, we derived the sum-product algorithm for a tree with pairwise interactions; in this section, we discuss its extension to arbitrary tree-structured factor graphs. Before proceeding, it is worth noting that the resulting algorithm is *not* strictly more general than sum-product for pairwise interactions. Indeed, in the case of discrete variables considered here, any tree-structured factor graph—regardless of the order of interactions that it involves—can be converted to an equivalent tree with pairwise interactions.

We use $M_{i\to a}$ to denote the message from node i to the factor node $a \in N(i)$, and similarly $M_{a\to i}$ to denote the message from factor node a to $i \in N(a)$. Both messages (in either direction) are functions of x_i, where $M_{a\to i}(x_i)$ (respectively $M_{i\to a}(x_i)$) denotes the value taken for a given $x_i \in \mathcal{X}_i$. We let $x_a = \{x_i, i \in N(a)\}$ denote the subvector of random variables associated with factor node $a \in \mathcal{F}$. With this notation, the sum-product updates for a general tree factor graph take the following form:

Sum-product updates for tree factor graph:

$$M_{i\to a}(x_i) \leftarrow \kappa\, \psi_i(x_i) \prod_{b\in N(i)\setminus\{a\}} M_{b\to i}(x_i), \quad \text{and} \tag{15a}$$

$$M_{a\to i}(x_i) \leftarrow \kappa \sum_{x_j,\, j\in N(a)\setminus\{i\}} \Big[\psi_a(x_a) \prod_{j\in N(a)\setminus\{i\}} M_{j\to a}(x_j)\Big]. \tag{15b}$$

Here the quantity κ again represents a positive constant chosen to ensure that the messages sum to one. (As before, its value can differ from line to line.) Upon convergence, the marginal distributions over x_i and over the variables x_a are given, respectively, by the quantities

$$\mu_i(x_i) = \kappa\, \psi_i(x_i) \prod_{a\in N(i)} M_{a\to i}(x_i), \quad \text{and} \tag{16a}$$

$$\mu_a(x_a) = \kappa\, \psi_a(x_a) \prod_{j\in N(a)} M_{j\to a}(x_j). \tag{16b}$$

We leave it as an exercise for the reader to verify that the message-passing updates (15) will converge after a finite number of iterations (related to the diameter of the factor tree), and when equation (16) is applied using the resulting fixed point M^* of the message updates, the correct marginal distributions are obtained.

3.2.3 Max-product algorithm

Thus far, we have discussed the sum-product algorithm that can be used to solve the problems of marginalization and likelihood computation. We now turn to the max-product algorithm, which is designed to solve the problem of mode computation. So as to clarify the essential ideas, we again present the algorithm in terms of a factor tree involving only pairwise interactions, with the understanding that the extension to a general factor tree is straightforward.

Recall that the mode computation problem amounts to finding an element

$$x^* \in \arg\max_{x\in\mathcal{X}^m} p(x_1,\ldots,x_m).$$

As an intermediate step, let us first consider the problem of computing the so-called max-marginals associated with any distribution defined by a tree $\mathcal{G} = (\mathcal{V},\mathcal{E})$. In particular, for each variable node $i \in \mathcal{V}$ and each $x_i \in \mathcal{X}_i$, we define the *singleton max-marginal*

$$\nu_i(x_i) := \max_{\{x'\in\mathcal{X}^m \,|\, x'_i = x_i\}} p(x'_1,\ldots,x'_m), \tag{17}$$

For each edge $(i, j) \in \mathcal{E}$, the *pairwise max-marginal* is defined in an analogous manner:

$$\nu_{ij}(x_i, x_j) := \max_{\{x' \in \mathcal{X}^m \mid (x'_i, x'_j) = (x_i, x_j)\}} p(x'_1, \ldots, x'_m), \tag{18}$$

Note that the singleton and pairwise max-marginals are the natural analogs of the usual marginal distributions, in which the summation operation has been replaced by the maximization operation.

Before describing how these max-marginals can be computed by the max-product algorithm, first let us consider how max-marginals are relevant for the problem of mode computation. This connection is especially simple if the singleton max-marginals satisfy the *unique maximizer condition*—namely, if for all $i \in \mathcal{V}$, the set $\arg\max_{x_i \in \mathcal{X}_i} \nu_i(x_i)$ consists of a single element, say $x_i^* \in \mathcal{X}_i$. In this case, we claim that the element $(x_1^*, \ldots, x_m^*) \in \mathcal{X}^m$ is the unique maximizer of $p(x_1, \ldots, x_m)$. Let us establish this claim. Note that for any $i \in \mathcal{V}$, by the unique maximizer condition and the definition of the max-marginals, we have

$$\nu_i(x_i^*) \;=\; \max_{x_i \in \mathcal{X}_i} \nu_i(x_i) \;=\; \max_{x \in \mathcal{X}^m} p(x).$$

Now for any $\tilde{x} \neq x^*$, there must be some index i such that $x_i^* \neq \tilde{x}_i$. For this index, we have

$$\max_{x \in \mathcal{X}^m} p(x) \;=\; \nu_i(x_i^*) \;>\; \nu_i(\tilde{x}_i) \;\geq\; p(\tilde{x}),$$

showing that $p(\tilde{x}) < \max_{x \in \mathcal{X}^m} p(x)$.

If the unique maximizer condition fails to hold, then the distribution p must have more than one mode, and it requires a bit more effort to determine one. Most importantly, it is *no longer* sufficient to extract *some* $x_i^* \in \arg\max_{x_i \in \mathcal{X}_i} \nu_i(x_i)$. To appreciate this fact, consider the distribution over a pair of binary variables given by

$$p(x_1, x_2) = \begin{cases} 0.45 & \text{if } x_1 \neq x_2, \text{ and} \\ 0.05 & \text{otherwise.} \end{cases} \tag{19}$$

In this case, we have $\nu_1(x_1) = \nu_2(x_2) = 0.45$ for all $(x_1, x_2) \in \{0, 1\}^2$, but only configurations with $x_1 \neq x_2$ are globally optimal. As a consequence, a naive procedure that looks only at the singleton max-marginals has no way of determining such a globally optimal configuration.

Consequently, when the unique maximizer condition no longer holds, it is necessary to also take into account the information provided by the pairwise max-marginals (18). In order to do so, we consider a back-tracking procedure that samples some configuration $x^* \in \arg\max_{x \in \mathcal{X}^m} p(x)$. Let us assume that the tree is rooted at node 1, and that $\mathcal{V} = \{1, 2, \ldots, m\}$ is a topological ordering (i.e., such that $\pi(i) < i$ for all $i = 2, \ldots, m$). Using this topological ordering we may generate an optimal configuration x^* as follows:

Back-tracking for choosing a mode x^*

1. Initialize the procedure at node 1 by choosing any $x_1^* \in \arg\max_{x_1 \in \mathcal{X}_1} \nu_1(x_1)$.
2. For each $i = 2, \ldots, m$, choose any $x_i^* \in \mathcal{X}_i$ such that $x_i^* \in \arg\max_{x_i \in \mathcal{X}_i} \nu_{i,\pi(i)}(x_i, x_{\pi(i)}^*)$.

We establish the correctness of this back-tracking procedure in Proposition 3 below.

Having established the utility of the max-marginals for computing modes, let us now turn to an efficient algorithm for computing them. As alluded to earlier, the max-marginals are the analogs of the ordinary marginals when summation is replaced with maximization. Accordingly, it is natural to consider the analog of the sum-product updates with the same replacement; doing so leads to the following *max-product updates*:

$$M_{j \to i}(x_i) \leftarrow \kappa \max_{x_j'} \Big\{ \psi_{ij}(x_i, x_j') \psi_j(x_j') \prod_{k \in N(j)/i} M_{k \to j}(x_j') \Big\}, \tag{20}$$

where $\kappa > 0$ chosen such that $\max_{x_i} M_{j \to i}(x_i) = 1$. When the algorithm converges to a fixed point M^*, we can compute the max-marginals via equations (13a) and (13b).

Proposition 3. *For any tree T with diameter $d(T)$, the max-product updates converge to their unique fixed point after at most $d(T)$ iterations, and the max-marginals obtained by equations* (13a) *and* (13b) *are exact. Moreover, the back-tracking procedure yields an element $x^* \in \arg\max_{x \in \mathcal{X}^m} p(x)$.*

Proof. The proof of convergence and correct computation of the max-marginals is formally identical to that of Proposition 2, so we leave the details to the reader as an exercise. Let us verify the remaining claim—namely, the optimality of any configuration $x^* \in \mathcal{X}^m$ returned by the back-tracking procedure.

In order to do so, recall that we have assumed without loss of generality (re-indexing as necessary) that $\mathcal{V} = \{1, 2, \ldots, m\}$ is a topological ordering, with vertex 1 as the root. Under this ordering, let us use the max-marginals to define the following cost function:

$$J(x; \nu) = \nu_1(x_1) \prod_{i=2}^{m} \frac{\nu_{i\pi(i)}(x_i, x_{\pi(i)})}{\nu_{\pi(i)}(x_{\pi(i)})}. \tag{21}$$

(In order for division to be defined in all cases, we take $0/0 = 0$.) We first claim that there exists a constant $\kappa > 0$ such that $J(x; \nu) = \kappa\, p(x)$ for all $x \in \mathcal{X}^m$, which implies

that $\arg\max_{x \in \mathcal{X}^m} J(x;\nu) = \arg\max_{x \in \mathcal{X}^m} p(x)$. (By virtue of this property, we say that ν defines a *reparameterization* of the original distribution.) To establish this property, we first note that the cost function J can also be written in the symmetric form

$$J(x;\nu) = \prod_{i=1}^{m} \nu_i(x_i) \prod_{(i,j)\in\mathcal{E}} \frac{\nu_{ij}(x_i,x_j)}{\nu_i(x_i)\nu_j(x_j)}. \tag{22}$$

We then recall that the max-marginals are specified by the message fixed point M^* via equations (13a) and (13b). Substituting these relations into the symmetric form (22) of J yields

$$\begin{aligned} J(x;\nu) &\propto \Big[\prod_{i\in\mathcal{V}} \psi_i(x_i) \prod_{k\in N(i)} M^*_{k\to i}(x_i)\Big]\Big[\prod_{(i,j)\in\mathcal{E}} \frac{\psi_{ij}(x_i,x_j)}{M^*_{i\to j}(x_j) M^*_{j\to i}(x_i)}\Big] \\ &= \prod_{i\in\mathcal{V}} \psi_i(x_i) \prod_{(i,j)\in\mathcal{E}} \psi_{ij}(x_i,x_j) \\ &\propto p(x), \end{aligned}$$

as claimed.

Consequently, it suffices to show that any $x^* \in \mathcal{X}^m$ returned by the back-tracking procedure is an element of $\arg\max_{x\in\mathcal{X}^m} J(x;\nu)$. We claim that

$$\frac{\nu_{i\pi(i)}(x^*_i, x^*_{\pi(i)})}{\nu_{\pi(i)}(x^*_{\pi(i)})} \geq \frac{\nu_{i\pi(i)}(x_i, x_{\pi(i)})}{\nu_{\pi(i)}(x_{\pi(i)})} \quad \text{for all } (x_i, x_{\pi(i)}). \tag{23}$$

By definition of the max-marginals, the left-hand side is equal to one. On the other hand, we have $\nu_{\pi(i)}(x_{\pi(i)}) \geq \nu_{i\pi(i)}(x_i, x_{\pi(i)})$, showing that the right-hand side is less than or equal to one.

By construction, any x^* returned by back-tracking satisfies $\nu_1(x^*_1) \geq \nu_1(x_1)$ for all $x_1 \in \mathcal{X}_1$. This fact, combined with the pairwise optimality (23) and the definition (21), implies that $J(x^*;\nu) \geq J(x;\nu)$ for all $x \in \mathcal{X}^m$, showing that x^* is an element of the set $\arg\max_{x\in\mathcal{X}^m} J(x;\nu)$, as claimed. □

In summary, the sum-product and max-product are closely related algorithms, both based on the basic principle of "divide-and-conquer". A careful examination shows that both algorithms are based on repeated exploitation of a common algebraic property, namely the *distributive law*. More specifically, for arbitrary real numbers $a, b, c \in \mathbb{R}$, we have

$$a \cdot (b+c) = a\cdot b + a \cdot c, \quad \text{and} \quad a \cdot \max\{b,c\} = \max\{a\cdot b,\, a\cdot c\}.$$

The sum-product (respectively max-product) algorithm derives its power by exploiting this distributivity to re-arrange the order of summation and multiplication (respectively maximization and multiplication) so as to minimize the number of steps required. Based on this perspective, it can be shown that similar updates apply to any pair of operations $(\oplus, \otimes)$ that satisfy the distributive law $a \otimes (b \oplus c) = a \otimes b \,\oplus\, a \otimes c$,

and for which $\otimes$ is a commutative operation (meaning $a \otimes b = b \otimes a$). Here the elements a, b, c need no longer be real numbers, but can be more exotic objects; for instance, they could elements of the ring of polynomials, where $(\otimes, \oplus)$ correspond to multiplication or addition with polynomials. We refer the interested reader to the papers [1, 5, 9, 11] for more details on such generalizations of the sum-product and max-product algorithms.

4 Junction tree framework

Thus far, we have derived the sum-product and max-product algorithms, which are exact algorithms for tree-structured graphs. Consequently, given a graph with cycles, it is natural to think about some type of graph transformation—for instance, such as grouping its nodes into clusters—-so as to form a tree to which the fast and exact algorithms can be applied. It turns out that this "clustering", if not done carefully, can lead to incorrect answers. Fortunately, there is a theory that formalizes and guarantees correctness of such a clustering procedure, which is known as the *junction tree framework.*

4.1 Clique trees and running intersection

In order to motivate the development to follow, let us begin with a simple (but cautionary) example. Consider the graph on five vertices shown in Figure 2(a), and note that it has four maximal cliques—namely $\{2,3,4\}$, $\{1,2\}$, $\{1,5\}$ and $\{4,5\}$. The so-called *clique graph* associated with this graph has four vertices, one for each of these maximal cliques, as illustrated in panel (b). The nodes corresponding to cliques C_1 and C_2 are joined by an edge in the clique graph if and only if $C_1 \cap C_2$ is not empty. In this case, we label the edge joining C_1 and C_2 with the set $S = C_1 \cap C_2$, which is known as the *separator set*. The separator sets are illustrated in gray boxes in panel (b). Finally, one possible tree contained within this clique graph—known as a *clique tree*—is shown in panel (c).

Now suppose that we were given a MRF distribution over the single cycle in panel (a); by the Hammersley-Clifford theorem, it would have the form

$$p(x_1, \ldots x_5) \propto \psi_{234}(x_2, x_3, x_4)\psi_{12}(x_1, x_2)\psi_{15}(x_1, x_5)\psi_{45}(x_4, x_5). \qquad (24)$$

Now we could provide an alternative factorization of (essentially) the same model on the clique graph in panel (b) by making two changes: first, introducing extra copies of variables (i.e., x'_i for $i = 1, \ldots 5$) so as to account for the fact that each variable from the original graph appears in multiple places on the clique graph, and second, introducing indicator functions on the edges to enforce equality between the different copies. Doing so yields a distribution $q(x, x') = q(x_1, \ldots, x_4, x_5 x'_1, \ldots, x'_4, x'_5)$

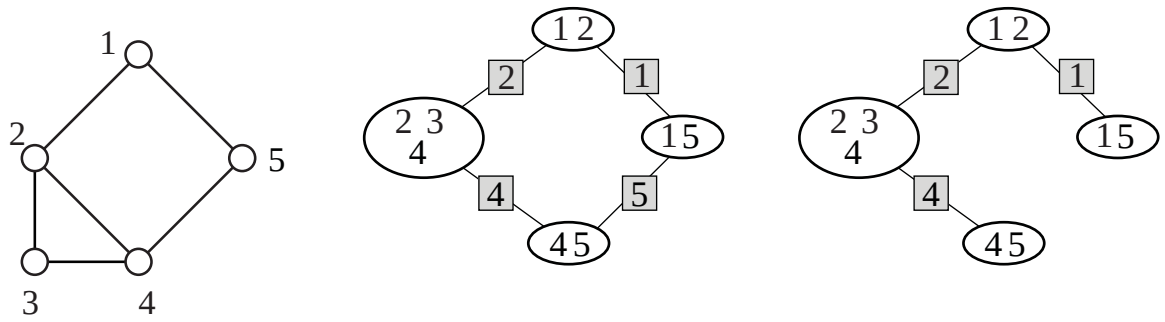

Fig. 2. (a) A graph with cycles on five vertices. (b) Associated clique graph in which circular nodes are maximal cliques of $\mathcal{G}$; square gray boxes sitting on the edges represent separator sets, corresponding to the intersections between adjacent cliques. (c) One clique tree extracted from the clique graph. It can be verified that this clique tree fails to satisfy the running intersection property.

that factorizes as

$$q(x,x') \propto \psi_{234}(x_2,x_3,x_4)\psi_{12}(x_1,x_2')\psi_{15}(x_1',x_5)\psi_{45}(x_4',x_5) \times \prod_{i=1}^{5} \mathbb{I}[x_i = x_i'], \quad (25)$$

where $\mathbb{I}[x_i = x_i']$ is a $\{0,1\}$-valued indicator function for the event $\{x_i = x_i'\}$. By construction, for any $i = 1,2,\ldots,4$, if we were to compute the marginal distribution $q(x_i)$ in the expanded model (25), it would be equal to the marginal $p(x_i)$ from the original model (24). (Although the expanded model (25) involves additional copies x_i', the indicator functions associated with the edges of the clique graph enforce the needed equalities to maintain model consistency.)

However, the clique graph in panel (b) and the distribution (25) are not still trees, which leads us to the key question. When is it possible to extract a tree from the clique graph, and have a factorization over the resulting clique tree that remains consistent with the original model (24)? Panel (c) of Figure 2 shows one clique tree, obtained by dropping the edge labeled with separator set $\{5\}$ that connects the nodes labeled with cliques $\{4,5\}$ and $\{1,5\}$. Accordingly, the resulting factorization $r(\cdot)$ over the tree would be obtained by dropping the indicator function $\mathbb{I}[x_5 = x_5']$, and take the form

$$r(x,x') \propto \psi_{234}(x_2,x_3,x_4)\psi_{12}(x_1,x_2')\psi_{15}(x_1',x_5)\psi_{45}(x_4',x_5) \times \prod_{i=1}^{4} \mathbb{I}[x_i = x_i'], \quad (26)$$

Of course, the advantage of this clique tree factorization is that we now have a distribution to which the sum-product algorithm could be applied, so as to compute the exact marginal distributions $r(x_i)$ for $i = 1,2,\ldots,5$. However, the drawback is that these marginals will *not be equal* to the marginals $p(x_i)$ of the original distribution (24). This problem occurs because—in sharp contrast to the clique graph factorization (25)—the distribution (26) might assign non-zero probability to some configuration for which $x_5 \neq x_5'$. Of course, the clique tree shown in Figure 2(c) is

only one of spanning trees contained within the clique graph in panel (b). However, the reader can verify that none of these clique trees have the desired property.

A bit more formally, the property that we require is the clique tree factorization always have enough structure to enforce the equivalences $x_i = x_i' = x_i'' = \ldots$, for all copies of a given variable $i \in \mathcal{V}$. (Although variables only appeared twice in the example of Figure 2, the clique graphs obtained from more complicated graphs could have any number of copies.) Note that there are copies x_i and x_i' of a given variable x_i if and only if i appears in at least two distinct cliques, say C_1 and C_2. In any clique tree, there must exist a unique path joining these two nodes, and what we require is that the equivalence $\{x_i = x_i'\}$ be propagated along this path. In graph-theoretic terms, the required property can be formalized as follows:

Definition 4. A clique tree has the *running intersection property* if for any two clique nodes C_1 and C_2, all nodes on the unique path joining them contain the intersection $C_1 \cap C_2$. A clique tree with this property is known as a *junction tree*.

To illustrate this definition, the clique tree in Figure 2 fails to satisfy running intersection since 5 belongs to both cliques $\{4,5\}$ and $\{1,5\}$, but does not belong to the clique $\{2,3,4\}$ along the path joining these two cliques in the clique tree. Let us now consider a modification of this example so as to illustrate a clique tree that does satisfy running intersection, and hence is a junction tree.

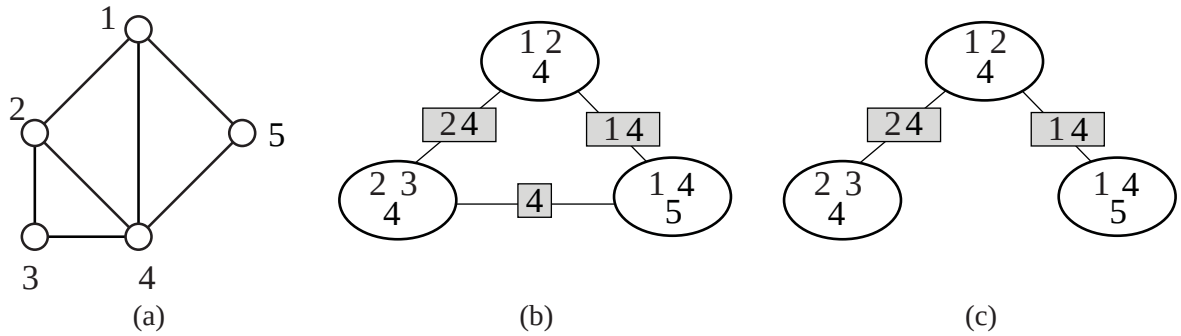

Fig. 3. (a) A modified version of the graph from Figure 2(a), containing the extra edge $(1,4)$. The modified graph has three maximal cliques, each of size three. (b) The associated clique graph has one node for each of three maximal cliques, with gray boxes representing the separator sets. (c) A clique tree extracted from the clique graph; it can be verified that this clique tree satisfies the running intersection property.

Figure 3(a) shows a modified version of the graph from Figure 2(a), obtained by adding the extra edge $(1,4)$. Due to this addition, the modified graph now contains three maximal cliques, each of size three. Panel (b) shows the associated clique graph; it contains three vertices, one for each of the three maximal cliques, and the

gray boxes on the edges represent the separator sets. Panel (c) shows one clique tree extracted from the clique graph in panel (b). In contrast to the tree from Figure 2(a), this clique tree does satisfy the running intersection property, and hence is a junction tree. (For instance, vertex 4 belongs to both cliques $\{2,3,4\}$ and $\{1,4,5\}$, and to every clique on the unique path joining these two cliques in the tree.)

4.2 Triangulation and junction trees

Thus far, we have studied in detail two particular graphs, one (Figure 2(a)) for which it was impossible to obtain a junction tree, and a second (Figure 3(a)) for which a junction tree could be found. In this section, we develop a principled basis on which to produce graphs for which the associated clique graph has a junction tree, based on the graph-theoretic notion of *triangulation*.

Recall that a cycle in a graph is a sequence of vertices $(s_1, s_2, \ldots s_\ell, s_1)$ such that $(s_\ell, s_1) \in \mathcal{E}$ and $(s_i s_{i+1}) \in \mathcal{E}$ for all $i = 1, \ldots, \ell - 1$. The cycle is *chordless* if there are no edges in the graph joining non-successive vertices in the cycle—that is, the graph does not contain any edges *apart from* those listed above that form the cycle. For example, the cycle $(1,2,4,5,1)$ in the graph from Figure 2(a) is chordless, whereas in contrast, the cycle $(1,2,3,4,5,1)$ contains the chord $(2,4)$.

Definition 5. A graph is *triangulated* if it contains no chordless cycles of length greater than three.

Thus, the graph in Figure 2(a) is not triangulated (due to the chordless cycle $(1,2,4,5,1)$), whereas it can be verified that the graph in Figure 3(a) is triangulated. We now state the fundamental connection between triangulation and junction trees:

Theorem 2. *The clique graph associated with a graph $\mathcal{G}$ has a junction tree if and only if $\mathcal{G}$ is triangulated.*

We provide a proof of this result as a part of a more general set of graph-theoretic equivalences in Appendix B. The practical significance of Theorem 2 is that it enables us to construct a modified graph $\widetilde{\mathcal{G}}$—by triangulating the original graph $\mathcal{G}$—such that the clique graph associated with $\widetilde{\mathcal{G}}$ is guaranteed to have at least one junction tree. There are a variety of algorithms for obtaining triangulated graphs. For instance, recall the *vertex elimination procedure* discussed in Section 3.1; it takes as input a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and then processes the vertices in a pre-specified order so as to produce a new graph.

Lemma 2. *The output $\widetilde{\mathcal{G}} = (\mathcal{V}, \widetilde{\mathcal{E}})$ of the vertex elimination algorithm is a triangulated graph.*

We leave the proof of Lemma 2 as an exercise for the reader. This fact establishes that there is a connection between the elimination algorithm and the property of triangulation.

Example 1 (Triangulation and junction tree). To illustrate the triangulation procedure, consider the 3×3 grid shown in Figure 4(a). If we run the vertex elimination algorithm using the ordering $\rho(\mathcal{V}) = \{1, 3, 7, 9, 4, 6, 2, 8\}$, then we obtain the graph $\widetilde{\mathcal{G}}$ shown in panel (b). (Note that the graph would not be triangulated if the additional edge joining nodes 2 and 8 were not present. Without this edge, the 4-cycle $\{2, 4, 8, 6, 2\}$ would lack a chord). Note that $\widetilde{\mathcal{G}}$ has six maximal cliques: two 4-cliques

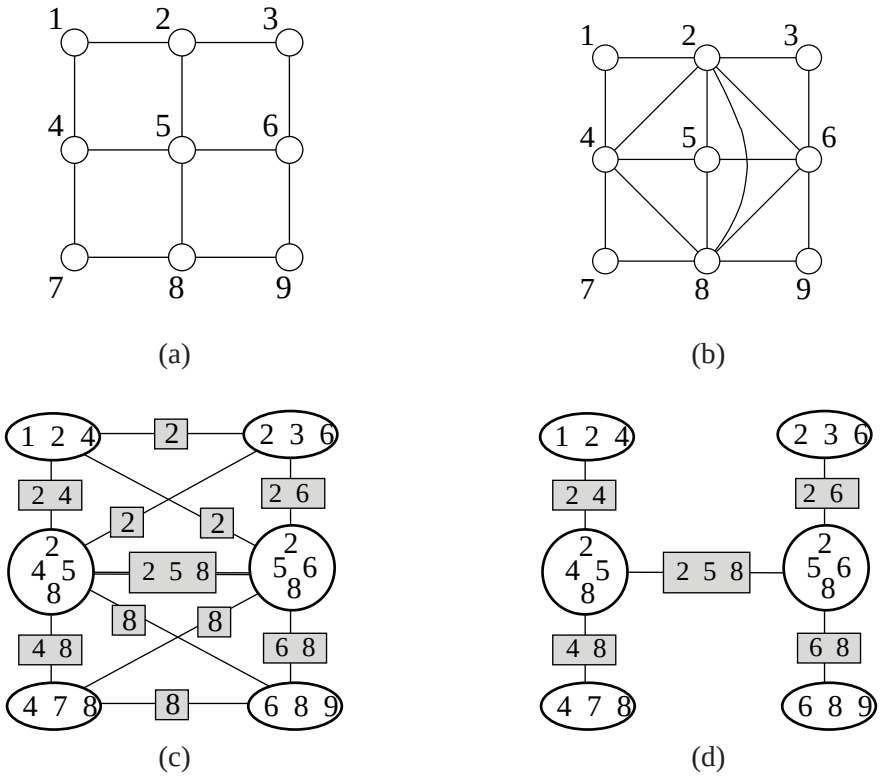

Fig. 4. Illustration of junction tree construction. (a) Original graph is a 3×3 grid. (b) Triangulated version of original graph. Note the two 4-cliques in the middle. (c) Corresponding junction tree for triangulated graph in (b), with maximal cliques depicted within ellipses, and separator sets within rectangles.

in the middle of the graph, and four 3-cliques on the boundary. The clique graph associated with $\widetilde{\mathcal{G}}$ is illustrated in panel (c), and one clique tree is shown in panel (d). It can be verified

4.3 Constructing the junction tree

Given that any triangulated graph has a junction tree (JT), the first step in the JT procedure is to triangulate the graph. At least in principle, this operation is straightforward since it entails adding edges so as to remove chordless cycles from the graph. (However, see the discussion at the end of Section 4 for discussion of optimal triangulations.) The focus of this section is the next step of the JT procedure—namely, how to form a junction tree from a triangulated graph. Any graph with cycles has more than one clique tree. Given a triangulated graph, we are guaranteed that at least one of these clique trees has the running intersection property, and so is a junction tree. How to find such a junction tree? It turns out that there is a simple procedure, based on solving a certain maximum weight spanning tree problem, that is always guaranteed to find a junction tree.

Given a triangulated graph, we again consider the clique graph, as previously defined. Recall that nodes C_1 and C_2 (corresponding to cliques from the original graph) are connected by an edge if and only if the associated separator set $S = C_1 \cap C_2$ is non-empty. Accordingly, for each edge $e = (C_1, C_2)$ of the clique graph, we can associate the positive weight $w(e) = \text{card}(C_1 \cap C_2)$. We refer to the resulting object as the *cardinality-weighted clique graph*. Given this weighted graph, the *maximum weight spanning tree* problem corresponds to finding the tree T (whose vertex set includes every node) such that the weight $w(T) := \sum_{e \in T} w(e)$ is maximized.

Proposition 4. *Any maximal weight spanning tree of the cardinality-weighted clique graph is a junction tree for the original graph.*

Proof. Suppose that the triangulated graph $\widetilde{\mathcal{G}}$ has M maximal cliques, so that (by definition) the weighted clique graph has M nodes. Let us consider an arbitrary spanning tree T of the weighted clique graph, say with nodes $\{C_1, \ldots, C_M\}$ and an associated edge set $\{S_1, \ldots, S_{M-1}\}$, where we have identified each edge with its associated separator set. The weight of this spanning tree can be written as $w(T) = \sum_{j=1}^{M-1} \text{card}(S_j)$. Letting $\mathbb{I}[t \in S_j]$ be an $\{0-1\}$-valued indicator function for the event $\{t \in S_j\}$, we may write $\text{card}(S_j) = \sum_{t \in \mathcal{V}} \mathbb{I}[t \in S_j]$, and hence we have

$$w(T) = \sum_{j=1}^{M-1} \sum_{t \in \mathcal{V}} \mathbb{I}[t \in S_j] \; = \; \sum_{t \in \mathcal{V}} \sum_{j=1}^{M-1} \mathbb{I}[t \in S_j] \tag{27}$$

We now claim that for any $t \in \mathcal{V}$, we have the inequality

$$\sum_{j=1}^{M-1} \mathbb{I}[t \in S_j] \leq \sum_{i=1}^{M} \mathbb{I}[t \in C_i] - 1, \tag{28}$$

with equality if and only if the subgraph induced by t is connected. To establish this claim, consider the subgraph of T that induced by vertex t—meaning the subgraph formed by the clique nodes C_i that include t, and the associated edges or separator sets S_j that also include t. Since it is a subgraph of the tree T, it must also be acyclic, from which the inequality (28) follows. If the subgraph has a single connected component, then equality holds. When T is actually a junction tree, this equality will hold for any vertex t.

Substituting the bound (28) into the earlier inequality (27), we conclude that

$$w(T) \leq \sum_{t \in \mathcal{V}} \Big\{ \sum_{i=1}^{M} \mathbb{I}[t \in C_i] - 1 \Big\},$$

where equality holds if and only T is a junction tree. Since the given tree T was arbitrary, the claim follows. □

Junction tree algorithm Given an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$:

1. Run a triangulation procedure to obtain a triangulated graph $\widetilde{\mathcal{G}}$.
2. Form the weighted junction graph, and find a maximum weight spanning tree via a standard algorithm (e.g., Kruskal's algorithm, Dykstra's algorithm).
3. Given the junction tree, define appropriate compatibility functions on its nodes and edges to reproduce the original distribution, and run the sum-product algorithm.

There are various techniques for finding *some* triangulation of a graph; in fact, running the vertex elimination algorithm (using any ordering) yields a triangulation. Note, however, that the complexity of tree inference depends on the size of the state spaces at each node in the junction tree, which are determined by the clique sizes in the triangulated graph. It would be desirable, then, to obtain a triangulation with minimal maximal clique size, or equivalently, with minimal treewidth.

Definition 6. The *treewidth* of a graph is the size of largest clique (minus one) in the best triangulation.

As an illustration, any ordinary tree has treewidth one, whereas the graph in Figure 3(a) has treewidth two, and the grid in Figure 4 has treewidth three.

For any fixed treewidth k, there are efficient algorithms—meaning polynomial-time in the graph size—to test whether the graph has treewidth k, and if so, to determine an associated junction tree [10, 3]. However, the complexity of these algorithms grows exponentially in the treewidth k, so they are feasible only for graphs of bounded treewidth. For a general graph, the problem of finding an optimal triangulation is NP-hard [**?**, 2]. Despite this negative result, there are a variety of heuristic algorithms for obtaining "good" triangulations of graph. One of most widely studied

is the minimum degree or minimum fill heuristic [**?**, **?**], which is based on following the elimination ordering obtained by choosing a minimum degree vertex (in the partially reduced graph) at each step.

5 Basics of parameter estimation

Acknowledgements This work was partially supported by NSF grants CCF-0545862, DMS-0605165 and CCF-0635372, and AFOSR grant 09NL184.

Appendix

A: Basics of graph theory

.....

B: Triangulation and equivalent graph-theoretic properties

In this Appendix, we prove Theorem 2 as part of a more general discussion of triangulation and related graph-theoretic properties. Having already defined the notions of triangulations and junction tree, let us now define the closely related notions of *decompsable* and *recursively simplicial*. The following notion serves to formalize the "divide-and-conquer" nature of efficient algorithms:

Definition 7. A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is *decomposable* if either it is complete, or its vertex set $\mathcal{V}$ can be split into the disjoint union of three sets three sets $A \cup B \cup S$ such that (a) A and B are non-empty; (b) the set S separates A and B in $\mathcal{G}$, and is complete (i.e., $(s,t) \in \mathcal{E}$ for all $s,t \in S$); and (c) $A \cup S$ and $B \cup S$ are also decomposable.

Recall from our discussion of the elimination algorithm in Section 3.1 that when a vertex is removed from the graph, the algorithm always connects together all of its neighbors, thereby creating additional edges in the reduced graph. The following property characterizes when there is an elimination ordering such that no edges are added by the elimination algorithm.

Definition 8. A vertex is *simplicial* if its neighbors form a complete subgraph. A non-empty graph is *recursively simplicial* if it contains a simplicial vertex, and when s is removed, any graph that remains is recursively simplicial.

It should be intuitively clear that these four properties—namely, triangulated, decomposable, recursively simplicial, and having a junction tree—are related. We now show that all four properties are actually equivalent:

Theorem 3. *The following properties of an undirected graph $\mathcal{G}$ are all equivalent:*

Property (T): $\mathcal{G}$ *is triangulated.*
Property (D): $\mathcal{G}$ *is decomposable.*
Property (R): $\mathcal{G}$ *is recursively simplicial.*
Property (J): $\mathcal{G}$ *has a junction tree.*

We will prove the sequence of implications $(T) \Rightarrow (D) \Rightarrow (R) \Rightarrow (J) \Rightarrow (T)$.

(T) $\Rightarrow$ (D): We proceed via induction on the graph size m. The claim is trivial for $m = 1$, so let us assume it for all graphs with m vertices, and prove that it also holds for any graph $\mathcal{G}$ with $m+1$ vertices. If $\mathcal{G}$ is complete, then it is certainly recursively simplicial. Moreover, if $\mathcal{G}$ has more than one connected component, each of which is complete, then it is also recursively simplicial. Otherwise, we may assume that at least one connected component of $\mathcal{G}$ is not complete. (Without loss of generality in the argument to follow, we assume that $\mathcal{G}$ has a single connected component which is not complete.) Since $\mathcal{G}$ is not complete, it contains two non-adjacent nodes a, b. Let S be a minimal set that separates a and b; the set S must be non-empty since $\mathcal{G}$ has a single connected component. Define A as the set of all nodes connected to a in $\mathcal{V}\backslash S$, and set $B := \mathcal{V}\backslash(A \cup S)$. Clearly, S separates A from B in $\mathcal{G}$.

Now need to show that S is complete. If $|S| = 1$, the claim is trivial. Otherwise, for any two distinct nodes $s, t \in \mathcal{S}$, there exist paths $(s, a_1, \ldots, a_i, t)$ and $(s, b_1, \ldots, b_j, t)$ where $a_k \in A$, $b_k \in B$ and $i, j \geq 1$. (This follows since S is minimal, so there must exist paths from a to s and from a to t; otherwise, S would not be minimal.) We claim that s and t are joined by an edge. If not, take the path from s to t through A with minimal length, and similarly for B. This pair of paths forms a cycle of length at least four, which must have a chord. The chord cannot be in A or B, since this would contradict minimality. It cannot be between nodes in A and B since S separates these two sets. Therefore, s and t are joined, and S is complete.

Finally, we need to show that $A \cup S$ and $B \cup S$ are also decomposable. But they are chordal (otherwise $\mathcal{G}$ is not chordal) and strictly smaller than $\mathcal{V}$, so the result follows by induction.

(D) $\Rightarrow$ (R): Proof by induction on graph size m. Trivial for $m = 1$. To complete the induction step, we require the follwing lemma:

Lemma 3. *Every decomposable graph with at least two nodes has at least two simplicial nodes. If the graph is not complete, these nodes can be chosen to be non-adjacent.*

Proof. Proof by induction on graph size m. Trivial for $m = 2$. Consider a decomposable graph with $m+1$ nodes. If the graph is complete, all nodes are simplicial. Otherwise, decompose the graph into disjoint sets A, B and S. The subgraphs $A \cup S$ and $B \cup S$ are also chordless, and hence we have two simplicial nodes in $A \cup S$. If $A \cup S$ is not complete, these can be chosen to be non-adjacent. Given that S is complete, one of the nodes can be taken in A. Otherwise, if $A \cup S$ is complete, choose any node in A. Proceed in a symmetric fashion for B. The simplicial nodes thus chosen will not be connected, since S separates A and B.

Thus, given a decomposable graph, we can find some simplicial vertex s to remove. We need to show that the remaining graph is also decomposable, so as to apply the induction hypothesis. In particular, we prove that $\mathcal{G}$ decomposable implies that any vertex-induced subgraph $\mathcal{G}[U]$ is also decomposable. We prove this induction on $|U|$. Trivial for $|U| = 1$. Trivially true if $\mathcal{G}$ is complete; otherwise, break into $A \cup S \cup B$. Removing a node from S leaves $S \backslash \{s\}$ complete, and $A \cup S$ and $B \cup S$ decomposable by the induction hypothesis. Removing a node from A does not change $B \cup S$, and either leaves A empty (in which case remainder $B \cup S$ is decomposable), or leaves $A \cup S$ decomposable by induction.

(R) $\Rightarrow$ (J): Proof by induction on graph size m. Trivial for $m = 1$. Let s be a simplicial vertex, and consider subgraph $\mathcal{G}'$ obtained by removing s. By induction, $\mathcal{G}'$ has a junction tree T', which we will extend to a junction tree T for $\mathcal{G}$. Let C' be a maximal clique in T' that contains all the neighbors of s; this must exist since s is simplicial. If C' is precisely the neighbors of s, then we can add s to C' so as to obtain T, which is a junction tree for $\mathcal{G}$.

If not (i.e., if C' contains the neighbors of s as a proper subset), then we can add a new clique containing s and its neighbors to T', with an edge to C'. Since s is in no other clique of T and $C \backslash \{s\}$ is a subset of C', the tree T' is a junction tree for $\mathcal{G}$¿

(J) $\Rightarrow$ (T): Proof by induction on number of nodes M in junction tree. For $M = 1$, $\mathcal{G}$ is complete and hence triangulated. Consider a junction tree T with $M+1$ nodes. For a fixed leaf C of T, let C' be the unique neighbor of C in T, and let T' be the tree that remains when C is removed.

Step 1: If $C \subseteq C'$, then T' is a junction tree for $\mathcal{G}$, and result follows by induction.

Step 2: If $C \cap C' \subset C$ (in a *strict* sense), then consider the subgraph $\mathcal{G}'$ formed by removing the non-empty set $R := C \backslash C'$ from $\mathcal{V}$. We claim that it is chordal. First, observe that R has an empty intersection with every clique in T' (using junction tree property). (I.e., say $R \cap D \neq 0$ for some clique node D in T'. Then there exists $s \in C \cap D$, but $s \notin C'$, with violates running intersection.) Follows that T' is a junction tree for $\mathcal{G}'$, and so $\mathcal{G}'$ is chordal (by applying induction hypothesis).

Step 3: Now claim that $\mathcal{G}$ is chordal. Any cycle entirely contained in $\mathcal{G}'$ is chordless by induction. If the cycle is entirely within the complete subgraph $\mathcal{G}[C]$, it is also chordless. Any other cycle must intersect R, $C \cap C'$ and $\mathcal{V} \backslash C$. In particular, it must cross $C \cap C'$ twice, and since this set is complete, it has a chord.

References

1. Aji, S., McEliece, R.: The generalized distributive law. IEEE Trans. Info. Theory **46**, 325–343 (2000)
2. Arnborg, S.: Complexity of finding embeddings in a *k*-tree. SIAM Jour. Alg. Disc. Math **3**(2), 277–284 (1987)
3. Bodlaender, H.L.: A linear-time algorithm for finding tree decompositions of small treewidth. SIAM Journal of Computing **25**, 1305–1317 (1996)
4. Bollobás, B.: Graph theory: an introductory course. Springer-Verlag, New York (1979)
5. Dawid, A.P.: Applications of a general propagation algorithm for probabilistic expert systems. Statistics and Computing **2**, 25–36 (1992)
6. Kschischang, F., Frey, B., Loeliger, H.A.: Factor graphs and the sum-product algorithm. IEEE Trans. Info. Theory **47**(2), 498–519 (2001)
7. van Lint, J.H., Wilson, R.M.: A course in combinatorics. Cambridge University Press, Cambridge (1992)
8. Loeliger, H.A.: An introduction to factor graphs. IEEE Signal Processing Magazine **21**, 28–41 (2004)
9. Shafer, G.R., Shenoy, P.P.: Probability propagation. Annals of Mathematics and Artificial Intelligence **2**, 327–352 (1990)
10. Tarjan, R.E., Yannakakis, M.: Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. SIAM Journal of Computing **13**(3), 566–579 (1984)
11. Verdú, S., Poor, H.V.: Abstract dynamic programming models under commutativity conditions. SIAM J. Control and Optimization **25**(4), 990–1006 (1987)
12. Wainwright, M.J., Jordan, M.I.: Graphical models, exponential families and variational inference. Foundations and Trends in Machine Learning **1**(1–2), 1—305 (2008)