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, electrical engineering and computer science, including statistical machine learning and artificial intelligence, communication and information theory, statistical physics, network control theory, computational biology, statistical signal processing, natural language processing and computer vision among others.

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 undergraduate-level background in linear algebra, multivariate calculus, probability theory (without needing measure theory), and some basic graph theory. These introductory lectures should be viewed as a pre-cursor to the monograph [67], 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 $X = (X_1, \ldots, X_N)$. Here for each

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

$s \in [N] := \{1, 2, \ldots, N\}$, the random variable X_s take values in some space $\mathcal{X}_s$, which (depending on the application) may either be continuous (e.g., $\mathcal{X}_s = \mathbb{R}$) or discrete (e.g., $\mathcal{X}_s = \{0, 1, \ldots, m-1\}$). Lower case letters are used to refer to particular elements of $\mathcal{X}_s$, so that the notation $\{X_s = x_s\}$ corresponds to the event that the random variable X_s takes the value $x_s \in \mathcal{X}_s$. The random vector $X = (X_1, X_2, \ldots, X_N)$ takes values in the Cartesian product space $\prod_{s=1}^N \mathcal{X}_s := \mathcal{X}_1 \times \mathcal{X}_2 \times \ldots \times \mathcal{X}_N$. For any subset $A \subseteq [N]$, we define the subvector $X_A := (X_s,\; s \in A)$, corresponding to a random vector that takes values in the space $\mathcal{X}_A = \prod_{s \in A} \mathcal{X}_s$. 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 $\mathcal{X}_{[N]}$ is shorthand notation for the full Cartesian product $\prod_{s=1}^N \mathcal{X}_s$. Given three disjoint subsets A, B, C of $[N]$, we use $X_A \perp\!\!\!\perp X_B \mid X_C$ to mean that the random vector X_A is conditionally independent of X_B given X_C. When C is the empty set, then this notion reduces to marginal independence between the random vectors X_A and X_B.

2.1 Directed graphical models

We begin our discussion with directed graphical models, which (not surprisingly) are based on the formalism of directed graphs. In particular, a directed graph $\mathcal{D} = (\mathcal{V}, \vec{\mathcal{E}})$ consists of a vertex set $\mathcal{V} = \{1, \ldots, N\}$ and a collection $\vec{\mathcal{E}}$ of directed pairs $(s \to t)$, meaning that s is connected by an edge directed to t. When there exists a directed edge $(t \to s) \in \mathcal{E}$, we say that node s is a *child* of node t, and conversely that node t is a *parent* of node s. We use $\pi(s)$ to denote the set of all parents of node s (which might be an empty set). A *directed cycle* is a sequence of vertices $(s_1, s_2, \ldots, s_\ell)$ such that $(s_\ell \to s_1) \in \vec{\mathcal{E}}$, and $(s_j \to s_{j+1}) \in \vec{\mathcal{E}}$ for all $j = 1, \ldots, \ell - 1$. A *directed acyclic graph*, or DAG for short, is a directed graph that contains no directed cycles. As an illustration, the graphs in panels (a) and (b) are both DAGs, whereas the graph in panel (c) is *not* a DAG, since it contains (among others) a directed cycle on the three vertices $\{1, 2, 5\}$.

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

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

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 simplify our presentation, we assume throughout these notes that the *canonical ordering* $\mathcal{V} = \{1, 2, \ldots, N\}$ is topological. Note that this assumption entails no loss of generality, since we can always re-index the vertices

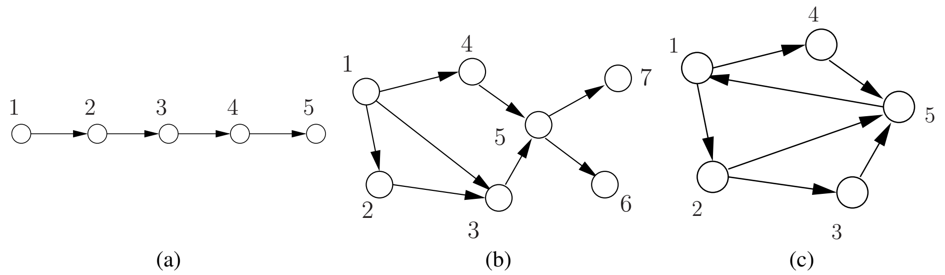

Fig. 1. (a) The simplest example of a DAG is a chain, which underlies the familiar Markov chain. The canonical ordering $\{1,2,\ldots,N\}$ is the only topological one. (b) A more complicated DAG. Here the canonical ordering $\{1,2,\ldots,7\}$ is again topological, but it is no longer unique: for instance, $\{1,4,2,3,5,7,6\}$ is also topological. (c) A directed graph with cycles (non-DAG). It contains (among others) a directed cycle on vertices $\{1,2,5\}$.

so that it holds. With this choice of topological ordering, vertex 1 cannot have any parents (i.e., $\pi(1) = \emptyset$), and moreover vertex N cannot have any children.

With this set-up, we are now ready to introduce probabilistic notions into the picture. A directed graphical model is a family of probability distributions defined by a DAG. This family is built by associating each node s of a DAG with a random variable X_s, and requiring the joint probability distribution over $(X_1,\ldots,X_N)$ factorize according to the DAG. Consider the subset of vertices $(s,\pi(s))$ corresponding to a given vertex s and its parents $\pi(s)$. We may associate with this subset a real-valued function $f_s : \mathcal{X}_s \times \mathcal{X}_{\pi(s)} \to \mathbb{R}_+$ that maps any given configuration $(x_s, x_{\pi(s)}) \in \mathcal{X}_s \times \mathcal{X}_{\pi(s)}$ to a real number $f_s(x_s, x_{\pi(s)}) \geq 0$. We assume moreover that f_s satisfies the normalization condition

$$\sum_{x_s} f_s(x_s,\, x_{\pi(s)}) = 1 \quad \text{for all } x_{\pi(s)} \in \mathcal{X}_{\pi(s)}. \tag{1}$$

Definition 2 (Factorization for directed graphical models). The *directed graphical model* based on a given DAG $\mathcal{D}$ is the collection of probability distributions over the random vector $(X_1,\ldots,X_N)$ that have a factorization of the form

$$p(x_1,\ldots,x_N) = \frac{1}{Z}\prod_{s=1}^{N} f_s(x_s,\, x_{\pi(s)}), \tag{2}$$

for some choice of non-negative parent-to-child functions $(f_1,\ldots,f_N)$ that satisfy the normalization condition (1). We use $\mathcal{F}_{\text{Fac}}(\mathcal{D})$ to denote the set of all distributions that factorize in the form (2).

In the factorization (2), the quantity Z denotes a constant chosen to ensure that p sums to one.

Let us illustrate this definition with some examples.

Example 1 (Markov chain as a directed graphical model). Perhaps the simplest example of a directed acyclic graph is the chain on N nodes, as shown in panel (a) of Figure 1. Such a graph underlies the stochastic process $(X_1, \ldots, X_N)$ known as a Markov chain, used to model various types of sequential dependencies. By definition, any Markov chain can be factorized in the form

$$p(x_1, \ldots, x_N) = p(x_1)\, p(x_2 \,|\, x_1)\, p(x_3 \,|\, x_2) \cdots p(x_N \,|\, x_{N-1}). \tag{3}$$

Note that this is a special case of the factorization (2), based on the functions $f_s(x_s,\, x_{\pi(s)}) = p(x_s \,|\, x_{s-1})$ for each $s = 2, \ldots, N$, and $f_1(x_1,\, x_{\pi(1)}) = p(x_1)$. ♣

We now turn to a more complex DAG.

Example 2 (Another DAG). Consider the DAG shown in Figure 1(b). It defines the family of probability distributions that have a factorization of the form

$$p(x_1, \ldots, x_7) \propto f_1(x_1) f_2(x_2,\, x_1) f_3(x_3,\, x_1, x_2) f_4(x_4,\, x_1) f_5(x_5,\, x_3, x_4) f_6(x_6,\, x_5) f_7(x_7,\, x_5).$$

for some collection of non-negative and suitably normalized functions $\{f_s, s \in \mathcal{V}\}$. ♣

The factorization (3) of the classical Markov chain has an interesting property, in that the normalization constant $Z = 1$, and all the local functions f_s are equal to conditional probability distributions. It is not immediately apparent whether or not this property holds for the more complex DAG discussed in Example 2, but in fact, as shown by the following result, it is a generic property of directed graphical models.

Proposition 1. *For any directed acyclic graph $\mathcal{D}$, any factorization of the form factorization* (2) *with $Z = 1$ defines a valid probability distribution. Moreover, we necessarily have $f_s(x_s,\, x_{\pi(s)}) = p(x_s \,|\, x_{\pi(s)})$ for all $s \in \mathcal{V}$.*

Proof. Throughout the proof, we assume without loss of generality (re-indexing as necessary) that $\{1, 2, \ldots, N\}$ is a topological ordering. In order to prove this result, it is convenient to first state an auxiliary result.

Lemma 1. *For any distribution p of the form* (2)*, we have*

$$p(x_1,\ldots,x_t) = \frac{1}{Z}\prod_{s=1}^{t} f_s(x_s,\, x_{\pi(s)}) \qquad \textit{for each } t = 1,\ldots,N. \tag{4}$$

We first use this result to establish the main claims before returning to prove it. If we apply Lemma 1 with $t = 1$, then we obtain that $p(x_1) = f_1(x_1)/Z$, and hence that $Z = 1$ by the normalization condition on f_1. Otherwise, for any $t \in \{2,\ldots,N\}$, applying the representation (4) to both t and $t-1$ yields

$$\frac{p(x_1,\ldots,x_t)}{p(x_1,\ldots,x_{t-1})} = f_t(x_t,\, x_{\pi(t)}) \qquad \text{for all } (x_1,\ldots,x_t).$$

Since the right-hand side depends only on $x_{\pi(t)}$, so must the left-hand side. When the left-hand side depends only on $x_{\pi(t)}$, then it is equal to the conditional $p(x_t \,|\, x_{\pi(t)})$, and so we conclude that $p(x_t \,|\, x_{\pi(t)}) = f_t(x_t,\, x_{\pi(t)})$ as claimed.

It remains to prove Lemma 1. We may assume without loss of generality (re-indexing as necessary) that $\{1,2,\ldots,N\}$ is a topological ordering. Consequently, node N has no children, so that we may write

$$p(x_1,\ldots,x_{N-1},x_N) = \frac{1}{Z}\Big[\prod_{s=1}^{N-1} f_s(x_s,\, x_{\pi(s)})\Big] f_N(x_N,\, x_{\pi(N)}).$$

Marginalizing over x_N yields that

$$\begin{aligned} p(x_1,\ldots,x_{N-1}) &= \frac{1}{Z}\Big[\prod_{s=1}^{N-1} f(x_s,\, x_{\pi(s)})\Big]\Big[\sum_{x_N} f_N(x_N,\, x_{\pi(N)})\Big] \\ &= \frac{1}{Z}\prod_{s=1}^{N-1} f_s(x_s,\, x_{\pi(s)}), \end{aligned}$$

where we have used the facts that x_N appears only in one term (since N is a leaf node), and that $\sum_{x_N} f_N(x_N,\, x_{\pi(N)}) = 1$. We have thus shown that if the claim holds for $t = N$, then it holds for $t = N-1$. By recursively applying this same argument, the claim of Lemma 1 follows. □

Proposition 1 shows that the terms f_i in the factorization (2) have a concrete interpretation as the child-parent conditional probabilities (i.e., $f_s(x_s,\, x_{\pi(s)})$ is equal to the conditional probability of $X_s = x_s$ given that $X_{\pi(s)} = x_{\pi(s)}$). 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 $(\tilde{X}_1,\ldots,\tilde{X}_N)$ from any DAG model is straightforward: assuming the canonical topological ordering, we first sample $\tilde{X}_1 \sim f_1(\cdot)$, and then for $s = 2,\ldots,N$, sample $\tilde{X}_s \sim f_s(\cdot,\, \tilde{X}_{\pi(s)})$. This procedure is well-specified: due to the topological ordering, we are guaranteed that the variable $\tilde{X}_{\pi(s)}$ has been sampled before we move on to sampling $\tilde{X}_s$. Moreover, by construction, the random vector $(\tilde{X}_1, \ldots, \tilde{X}_N)$ is distributed according to the probability distribution (2).

2.1.1 Conditional independence properties for directed graphs

Thus far, we have specified a joint distribution over the random vector $X = (X_1, \ldots, X_N)$ in terms of a particular parent-to-child factorization. We now turn to a different (but ultimately equivalent) characterization in terms of conditional independence. (The reader should recall our standard notation for conditional independence properties from the beginning of Section 2.) Throughout the discussion to follow, we continue to assume that the canonical ordering $\{1, 2, \ldots, N\}$ is topological.

Given any vertex $s \in \mathcal{V} \backslash \{1\}$, our choice of topological ordering implies that the parent set $\pi(s)$ is contained within the set $\{1, 2, \ldots, s-1\}$. Note that for $s = 1$, the parent set must be empty. We then define the set $\nu(s) = \{1, 2, \ldots, s-1\} \backslash \pi(s)$. Our basic conditional independence properties are based on the three disjoint subsets $\{s\}$, $\pi(s)$ and $\nu(s)$.

Definition 3 (Markov property for directed graphical models). The random vector $X = (X_1, \ldots, X_N)$ is Markov with respect to a directed graph if

$$X_s \perp\!\!\!\perp X_{\nu(s)} \mid X_{\pi(s)} \quad \text{for all } s \in \mathcal{V}. \tag{5}$$

We use $\mathcal{F}_{\text{Mar}}(\mathcal{D})$ to denote the set of all distributions that are Markov with respect to $\mathcal{D}$.

Let us illustrate this definition with our running examples.

Example 3 (Conditional independence for directed Markov chain). Recall the directed Markov chain first presented in Example 1. Each node $s \in \{2, \ldots, N\}$ has a unique parent $\pi(s) = s-1$, so that the (non-trivial) basic conditional properties are of the form $X_s \perp\!\!\!\perp (X_1, \ldots, X_{s-2}) \mid X_{s-1}$ for $s \in \{3, \ldots, N\}$. Note that these basic conditional independence statements imply other (non-basic) properties as well. Perhaps the most familiar is the assertion that

$$(X_s, X_{s+1}, \ldots, X_N) \perp\!\!\!\perp (X_1, \ldots, X_{s-2}) \mid X_{s-1} \qquad \text{for all } s \in \{3, \ldots, N\},$$

corresponding the fact that the past and future of a Markov chain are conditionally independent given the present (X_{s-1} in this case). ♣

As a second example, let us now return to the DAG shown in Figure 1(b).

Example 4 (Conditional independence for a more complex DAG). For this DAG, the basic conditional independence assertions (using the canonical ordering) can again be read off from the graph. In particular, the non-trivial relations are $X_3 \perp\!\!\!\perp X_1 \mid X_2$, $X_5 \perp\!\!\!\perp (X_1, X_2) \mid (X_3, X_4)$, as well as

$$X_6 \perp\!\!\!\perp (X_1, X_2, X_3, X_4) \mid X_5, \quad \text{and} \quad X_7 \perp\!\!\!\perp (X_1, X_2, X_3, X_4, X_6) \mid X_5.$$

♣

2.1.2 Equivalence of representations

For any directed graph $\mathcal{D}$, we have now defined two families of probability distributions: the family $\mathcal{F}_{\text{Fac}}(\mathcal{D})$ of all distributions with a factorization of the form (2), and the family $\mathcal{F}_{\text{Mar}}(\mathcal{D})$ of all distributions that satisfy the basic Markov properties (5). It is natural to ask how these two families are related; pleasingly, they are equivalent, as shown in the following result.

Theorem 1 (Equivalence for directed graphical models). *For any directed acyclic graph $\mathcal{D}$, we have $\mathcal{F}_{Fac}(\mathcal{D}) = \mathcal{F}_{Mar}(\mathcal{D})$.*

Proof. The proof of this result is straightforward given our development thus far. We begin with the inclusion $\mathcal{F}_{\text{Mar}}(\mathcal{D}) \subseteq \mathcal{F}_{\text{Fac}}(\mathcal{D})$. From Lemma 1 used in the proof of Proposition 1, for any vertex $t \in \mathcal{V}$, we have

$$\begin{aligned} p(x_1, \ldots, x_t) &= \prod_{s=1}^{t} p(x_s \mid x_{\pi(s)}) \;=\; \prod_{s=1}^{t-1} p(x_s \mid x_{\pi(s)}) \;\; p(x_t \mid x_{\pi(t)}) \\ &= p(x_1, \ldots, x_{t-1}) \;\; p(x_t \mid x_{\pi(t)}). \end{aligned}$$

By the definition of $\nu(t)$, we have $\{1, \ldots, t-1\} = \pi(t) \cup \nu(t)$, and consequently we can write

$$\frac{p(x_{\pi(t)}, x_{\nu(t)}, x_t)}{p(x_{\pi(t)})} = \frac{p(x_{\pi(t)}, x_{\nu(t)})}{p(x_{\pi(t)})} \;\; p(x_t \mid x_{\pi(t)}) \;=\; p(x_{\nu(t)} \mid x_{\pi(t)})\, p(x_t \mid x_{\pi(t)}),$$

which shows that $X_t \perp\!\!\!\perp X_{\nu(t)} \mid X_{\pi(t)}$.

In order to establish the reverse inclusion, suppose that the basic Markov properties hold, where we are still using $\{1, 2, \ldots, N\}$ as our topological ordering. Using the chain rule for probability, we have

$$\begin{aligned} p(x_1, \ldots, x_N) &= p(x_1) \prod_{s=2}^{N} p(x_t \mid x_1, \ldots, x_{t-1}) \\ &\overset{(i)}{=} p(x_1) \prod_{s=2}^{N} p(x_t \mid x_{\pi(t)}), \end{aligned}$$

where equality (i) follows by applying the Markov properties. □

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 $\mathcal{V} = \{1, \ldots, N\}$ is the vertex set (as before), and $\mathcal{E}$ is a collection of undirected edges, meaning that there is no distinction between the edge (s,t) and the edge (t,s). As before, we associate a random variable X_s with each vertex $s \in \mathcal{V}$ of graph, and our interest is in characterizing the joint distribution of the random vector $X = (X_1, \ldots, X_N)$.

As with directed graphical models, there are two different ways in which the probabilistic structure of the random vector X can be linked to the graphical structure: factorization and conditional independence properties. Let us begin our exploration with the former property.

2.2.1 Factorization for undirected models

For undirected graphical models, the factorization properties are specified in terms of cliques of the graph. 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

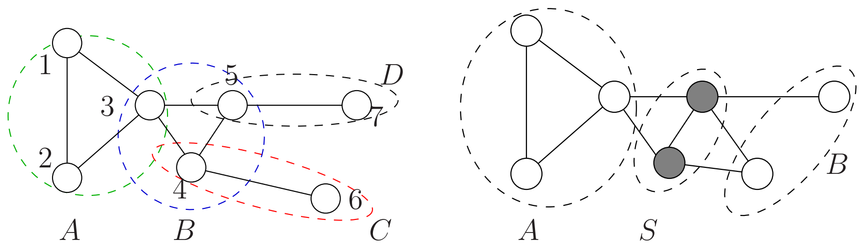

Fig. 2. (a) Illustration of cliques in an undirected graph. Sets $A = \{1,2,3\}$ and $B = \{3,4,5\}$ are 3-cliques, whereas sets $C = \{4,6\}$ and $D = \{5,7\}$ are 2-cliques (that can be identified with edges). (b) Illustration of a vertex cutset: removal of the vertices in S breaks the graph into the separate pieces indexed by subsets A and B.

it is not contained within any other clique. Thus, any singleton set $\{s\}$ is always a clique, but it is not maximal unless s has no neighbors in the graph. See Figure 2(a) for an illustration of some other types of cliques.

Given an undirected graph, we use $\mathfrak{C}$ to denote the set of all its cliques. With each clique $C \in \mathfrak{C}$, we associate a *compatibility function* $\psi_C : \mathcal{X}_C \to \mathbb{R}_+$: it assigns a non-negative number $\psi_C(x_C)$ to each possible configuration $x_C = (x_s, s \in C) \in \mathcal{X}_C$.

The factorization property for an undirected graphical model is stated in terms of these compatibility functions.

Definition 4 (Factorization for undirected graphical models). A probability distribution p factorizes over the graph $\mathcal{G}$ if

$$p(x_1,\ldots,x_N) = \frac{1}{Z}\prod_{C\in\mathfrak{C}}\psi_C(x_C), \tag{6}$$

for some choice of non-negative compatibility function $\psi_C : \mathcal{X}_C \to \mathbb{R}_+$ for each clique $C \in \mathfrak{C}$. We use $\mathcal{F}_{\text{Fac}}(\mathcal{G})$ to denote the set of all distributions that have a factorization of the form (6).

As before, the quantity Z is a constant chosen to ensure that the distribution is appropriately normalized. (In contrast to the directed case, we cannot take $Z = 1$ in general.) Note that there is a great deal of freedom in the factorization (6); in particular, in the way that we have written it, there is no unique choice of the compatibility functions and the normalization constant Z. For instance, if we multiply any compatibility function by some constant $\alpha > 0$ and do the same to Z, we obtain the same distribution.

Let us illustrate these definitions with some examples.

Example 5 (Markov chain as an undirected graphical model). Suppose that we remove the arrows from the edges of the chain graph shown in Figure 1(a); we then obtain an undirected chain. Any distribution that factorizes according to this undirected chain has the form

$$p(x_1,\ldots,x_5) = \frac{1}{Z}\prod_{s=1}^{5}\psi_s(x_s)\,\psi_{12}(x_1,x_2)\,\psi_{12}(x_1,x_2)\,\psi_{12}(x_1,x_2)\,\psi_{12}(x_1,x_2) \tag{7}$$

for some choice of non-negative compatibility functions. Unlike the directed case, in general, these compatibility functions are *not* equal to conditional distributions, nor to marginal distributions. Moreover, we have no guarantee that $Z = 1$ in this undirected representation of a Markov chain. The factorization (7) makes use of both maximal cliques (edges) and non-maximal cliques (singletons). Without loss of generality, we can always restrict to only edge-based factors. However, for applications, it is often convenient, for the purposes of interpretability, to include both types of compatibility functions. ♣

Example 6. As a second example, consider the undirected graph shown in Figure 2(a). Any distribution that respects its clique structure has the factorization

$$p(x_1,\ldots,x_7) = \frac{1}{Z}\,\psi_{123}(x_1,x_2,x_3)\,\psi_{345}(x_3,x_4,x_5)\,\psi_{4.6}(x_4,x_6)\,\psi_{57}(x_5,x_7),$$

for some choice of non-negative compatibility functions. In this case, we have restricted our factorization to the maximal cliques of the graph. ♣

2.2.2 Markov property for undirected models

We now turn to the second way in which graph structure can be related to probabilistic structure, namely via conditional independence properties. In the case of undirected graphs, these conditional independence properties are specified in terms of vertex cutsets of the graph. A *vertex cutset* is a subset S of the vertex set $\mathcal{V}$ such that, when it is removed from the graph, breaks the graph into two or more disconnected subsets of vertices. For instance, as shown in Figure 2(b), removing the subset S of vertices breaks the graph into two disconnected components, as indexed by the vertices in A and B respectively. For an undirected graph, the Markov property is defined by these vertex cutsets:

Definition 5 (Markov property for undirected graphical models). A random vector $(X_1,\ldots,X_N)$ is *Markov with respect to an undirected graph* $\mathcal{G}$ if, for all all vertex cutsets S and associated components A and B, the conditional independence condition $X_A \perp\!\!\!\perp X_B \mid X_C$ holds. We use $\mathcal{F}_{\text{Mar}}(\mathcal{G})$ to denote the set of all distributions that satisfy all such Markov properties defined by $\mathcal{G}$.

For example, for any vertex s, its *neighborhood set*

$$\mathcal{N}(s) := \{t \in \mathcal{V} \mid (s,t) \in \mathcal{E}\} \tag{8}$$

is always a vertex cutset. Consequently, any distribution that is Markov with respect to an undirected graph satisfies the conditional independence relations

$$X_s \perp\!\!\!\perp X_{\mathcal{V}\backslash\{s\}\cup\mathcal{N}(s)} \mid X_{\mathcal{N}(s)}. \tag{9}$$

The set of variables $X_{\mathcal{N}(s)} = (X_t, t \in \mathcal{N}(s)\}$ is often referred to as the *Markov blanket* of X_s, since it is the minimal subset required to render X_s conditionally independent of all other variables in the graph. This particular Markov property plays an important role in pseudolikelihood estimation of parameters [11], as well as related approaches for graphical model selection by neighborhood regression (see the papers [50, 54] for further details).

Let us illustrate with some additional concrete examples.

Example 7 (Conditional independence for a Markov chain). Recall from Example 5 the view of a Markov chain as an undirected graphical model. For this chain graph, the minimal vertex cutsets are singletons, with the non-trivial ones being the subsets $\{s\}$ for $s \in \{2, \ldots, N-1\}$ Each such vertex cutset induces the familiar conditional independence relation

$$\underbrace{(X_1, \ldots, X_{s-1})}_{\text{Past}} \perp\!\!\!\perp \underbrace{(X_{s+1}, \ldots, X_N)}_{\text{Future}} \mid \underbrace{X_s}_{\text{Present}}, \tag{10}$$

corresponding to the fact that the past and future of a Markov chain are conditionally independent given the present. ♣

Example 8 (More Markov properties). Consider the undirected graph shown in Figure 2(b). As previously discussed, the set S is a vertex cutset, and hence it induces the conditional independence property $X_A \perp\!\!\!\perp X_B \mid X_S$. ♣

2.2.3 Hammersley-Clifford equivalence

As in the directed case, we have now specified two ways in which graph structure can be linked to probabilistic structure. This section is devoted to a classical result that establishes a certain equivalence between factorization and Markov properties, meaning the two families of distributions $\mathcal{F}_{\text{Fac}}(\mathcal{G})$ and $\mathcal{F}_{\text{Mar}}(\mathcal{G})$ respectively. For strictly positive distributions p, this equivalence is complete.

Theorem 2 (Hammersley-Clifford). *If the distribution p of random vector $X = (X_1, \ldots, X_N)$ factorizes over a graph $\mathcal{G}$, then X is Markov with respect to $\mathcal{G}$. Conversely, if a random vector X is Markov with respect to $\mathcal{G}$* and $p(x) > 0$ *for all $x \in \mathcal{X}_{[N]}$, then p factorizes over the graph.*

Proof. We begin by proving that $\mathcal{F}_{\text{Fac}}(\mathcal{G}) \subseteq \mathcal{F}_{\text{Mar}}(\mathcal{G})$. Suppose that the factorization (6) 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 expressions 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

$$\begin{aligned} \frac{p(x_A, x_S, x_B)}{p(x_S)} &= \frac{\frac{1}{Z} \Psi_A(x_A, x_S) \Psi_S(x_S) \Psi_B(x_B, x_S)}{\frac{Z_A(x_S)\, Z_B(x_S)}{Z} \Psi_S(x_S)} \\ &= \frac{\Psi_A(x_A, x_S) \Psi_B(x_B, x_S)}{Z_A(x_S)\, Z_B(x_S)}. \end{aligned} \tag{11}$$

Similar calculations yield the relations

$$\frac{p(x_A, x_S)}{p(x_S)} = \frac{\frac{Z_B(x_S)}{Z} \Psi_A(x_A, x_S)\, \Psi_S(x_S)}{\frac{Z_A(x_S) Z_B(x_S)}{Z} \Psi_S(x_S)} = \frac{\Psi_A(x_A, x_S)}{Z_A(x_S)}, \quad \text{and} \tag{12a}$$

$$\frac{p(x_B, x_S)}{p(x_S)} = \frac{\frac{Z_A(x_S)}{Z} \Psi_B(x_B, x_S)\, \Psi_S(x_S)}{\frac{Z_A(x_S) Z_B(x_S)}{Z} \Psi_S(x_S)} = \frac{\Psi_B(x_B, x_S)}{Z_B(x_S)}. \tag{12b}$$

Combining equation (11) with equations (12a) and (12b) 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 opposite inclusion—namely, that the Markov property for a strictly positive distribution implies the factorization property—we require a version of the *inclusion-exclusion formula*. Given a set $[N] = \{1, 2, \ldots, N\}$, let $\mathcal{P}([N])$ be its power set, meaning the set of all subsets of $[N]$. With this notation, the following inclusion-exclusion formula is classical:

Lemma 2 (Inclusion-exclusion). *For any two real-valued functions Ψ and Φ defined on the power set $\mathcal{P}([N])$, the following statements are equivalent:*

$$\Phi(A) = \sum_{B \subseteq A} (-1)^{|A \backslash B|} \Psi(B) \quad \text{for all } A \in \mathcal{P}([N]). \tag{13a}$$

$$\Psi(A) = \sum_{B \subseteq A} \Phi(B) \quad \text{for all } A \in \mathcal{P}([N]). \tag{13b}$$

Proofs of this result can be found in standard texts in combinatorics (e.g., [47]). Returning to the main thread, let $y \in \mathcal{X}_{[N]}$ be some fixed element, and for each subset $A \in \mathcal{P}([N])$, define the function $\phi : \mathcal{X}_{[N]} \to \mathbb{R}$ via

$$\phi_A(x) := \sum_{B \subseteq A} (-1)^{|A \backslash B|} \log \frac{p(x_B, y_{B^c})}{p(y)}. \tag{14}$$

(Note that taking logarithms is meaningful since we have assumed $p(x) > 0$ for all $x \in \mathcal{X}_{[N]}$). 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 $A = [N]$ yields

$$p(x) = p(y) \exp\{ \sum_{B \in \mathcal{P}([N])]} \phi_B(x) \}. \tag{15}$$

From the definition (14), 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

$$\phi_A(x) = \sum_{B \subseteq A \backslash \{t\}} (-1)^{|A-B|} \log \frac{p(x_t \mid x_B, y_{B^c \backslash \{t\}})}{p(y_t \mid x_B, y_{B^c \backslash \{t\}})}. \tag{16}$$

To establish this claim, we write

$$\begin{aligned}
\phi_A(x) &= \sum_{\substack{B \subset A \\ B \not\ni t}} (-1)^{|A \backslash B|} \log \frac{p(x_B, y_{B^c})}{p(y)} + \sum_{\substack{B \subset A \\ B \ni t}} (-1)^{|A \backslash B|} \log \frac{p(x_B, y_{B^c})}{p(y)} \\
&= \sum_{B \subseteq A \backslash \{t\}} (-1)^{|A \backslash B|} \left\{ \log \frac{p(x_B, y_{B^c})}{p(y)} - \log \frac{p(x_{B \cup t}, y_{B^c \backslash \{t\}})}{p(y)} \right\} \\
&= \sum_{B \subseteq A \backslash \{t\}} (-1)^{|A \backslash B|} \log \frac{p(x_B, y_{B^c})}{p(x_{B \cup t}, y_{B^c \backslash \{t\}})}
\end{aligned} \tag{17}$$

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

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

Substituting into equation (17) yields the claim (16).

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 (16), we can write ϕ_A as a sum of four terms (i.e., $\phi_A = \sum_{i=1}^{4} T_i$), where

$$\begin{aligned}
T_1(x) &= \sum_{B \subseteq A \backslash \{s,t\}} (-1)^{|A \backslash B|} \log p(y_t \,|\, x_B, y_{B^c \backslash \{t\}}), \\
T_2(x) &= \sum_{B \subseteq A \backslash \{s,t\}} (-1)^{|A \backslash (B \cup \{t\})|} \log p(x_t \,|\, x_B, y_{B^c \backslash \{t\}}) \\
T_3(x) &= \sum_{B \subseteq A \backslash \{s,t\}} (-1)^{|A \backslash (B \cup \{s\})|} \log p(y_t \,|\, x_{B \cup \{s\}}, y_{B^c \backslash \{s,t\}}), \quad \text{and} \\
T_4(x) &= \sum_{B \subseteq A \backslash \{s,t\}} (-1)^{|A \backslash (B \cup \{s,t\})|} \log p(x_t \,|\, x_{B \cup \{s\}}, y_{B^c \backslash \{s,t\}}).
\end{aligned}$$

Combining these separate terms, we obtain

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

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

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

which completes the proof. □

2.2.4 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 [44, 48].

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 (s,a) of nodes $s \in \mathcal{V}$ such that the fact $a \in \mathcal{N}(s)$, or equivalently such that $a \in \mathcal{N}(s)$. See Figure 3(b) for an illustration.

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. This explicitness can be useful in resolving certain ambiguities that arise with the stan-

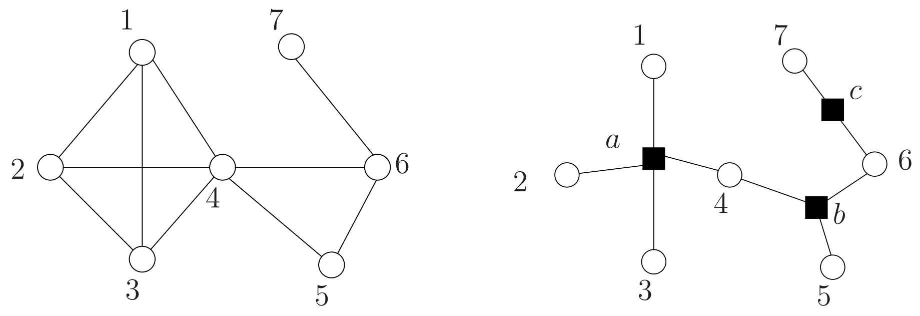

Fig. 3. Illustration of undirected graphical models and factor graphs. (a) An undirected graph on $N = 7$ vertices, with maximal cliques $\{1,2,3,4\}$, $\{4,5,6\}$ and $\{6,7\}$. (b) Equivalent representation of the undirected graph in (a) as a factor graph, assuming that we define compatibility functions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

dard formalism of undirected graphs. As one illustration, consider the three-vertex cycle shown in Figure 4(a). Two possible factorizations that are consistent with this graph structure are the *pairwise-only factorization*

$$p(x_1,x_2,x_3) = \frac{1}{Z}\psi_{12}(x_1,x_2)\,\psi_{23}(x_2,x_3)\,\psi_{13}(x_1,x_3), \tag{18}$$

which involves only the non-maximal pairwise cliques, versus the *generic triplet factorization* $p(x_1,x_2,x_3) = \frac{1}{Z'}\psi_{123}(x_1,x_2,x_3)$. The undirected graph in panel (a) does not distinguish between these two possibilities. In contrast, the factor graph representations of these two models are distinct, as shown in panels (b) and (c) respectively. This distinction between the pairwise and triplet factorization is im-

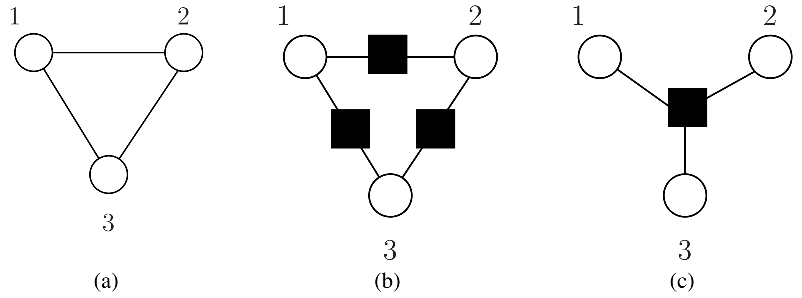

Fig. 4. (a) A three-vertex cycle graph that can be used to represent any distribution over three random variables. (b) Factor graph representation of the pairwise model (19). (c) Factor graph representation of the triplet interaction model.

portant in many applications, for instance when testing for the presence of ternary interactions between different factors underlying a particular disease.

3 Exact algorithms for marginals, likelihoods and modes

In applications of graphical models, one is typically interested in solving one of a core set of computational problems. One example is the problem of *likelihood computation*, which arises in parameter estimation and hypothesis testing with graphical models. 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 a configuration $\widehat{x} \in \mathcal{X}_{[N]}$ of highest probability—that is, $\widehat{x} \in \max_{x \in \mathcal{X}_{[N]}} p(x)$. In typical applications, some subset of the variables are observed, so these computations are often performed for conditional distributions, with the observed values fixed. It is straightforward to incorporate observed values into the algorithms that we discuss in this section.

At first sight, these problems—-likelihood computation, marginalization and mode computation—might seem quite simple and indeed, for small graphical models (e.g., $N \approx 10$—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 $(X_1, \ldots, X_N) \in \mathcal{X}_{[N]}$ such that, for each node $s \in \mathcal{V}$, the random variable X_s takes values in the state space $\mathcal{X}_s = \{0, 1, \ldots, m-1\}$. 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 m^{N-1} elements, it is clear that a brute force approach will rapidly become intractable. Given a graph with $N = 100$ vertices (a relatively small problem) and binary variables ($m = 2$), 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, then moving onto more sophisticated message-passing algorithms, and culminating in our derivation of the junction tree algorithm.

3.1 Elimination algorithm

We begin with an exact but relatively naive method known as the elimination algorithm. It operates on an undirected graphical model equipped with a factorization of the form

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

As input to the elimination algorithm, we provide it with a *target vertex* $T \in \mathcal{V}$ (or more generally a target subset of vertices), an *elimination ordering* $\mathcal{I}$, meaning an ordering of the vertex set $\mathcal{V}$ such that the target T appears first; as well as the collection of compatibility functions $\{\psi_C, C \in \mathfrak{C}\}$ defining the factorization (6). As output, the elimination algorithm returns the marginal distribution $p(x_T)$ over the variables x_T at the target vertices.

Elimination algorithm for marginalization

1. Initialization:
   (a) Initialize active set of compatibility functions $\mathcal{A} = \{\psi_C, C \in \mathfrak{C}\}$.
   (b) Initialize elimination ordering $\mathcal{I}$ (with the target vertex T appearing first).
2. Elimination: while card($\mathcal{I}$) > card(T):
   (a) Activate last vertex s on the current elimination order:
   (b) Form product

$$\phi(x_s, x_{\mathcal{N}(s)}) = \prod_{\psi_C \text{ involving } x_s} \psi_C(x_C),$$

   where $\mathcal{N}(s)$ indexes all variables that share active factors with s.
   (c) Compute partial sum:

$$\widetilde{\phi}_{\mathcal{N}(s)}(x_{\mathcal{N}(s)}) = \sum_{x_s} \phi(x_s, x_{\mathcal{N}(s)}),$$

and remove all $\{\psi_C\}$ that involve x_s from active list.
(d) Add $\widetilde{\phi}_{N(s)}$ to active list, and remove s from index set $\mathcal{I}$.
3. Termination: when $\mathcal{I} = \{T\}$, form the product $\phi_T(x_T)$ of all remaining functions in $\mathcal{A}$, and then compute

$$Z = \sum_{x_T} \phi_T(x_T) \quad \text{and} \quad p(x_T) = \frac{\phi_T(x_T)}{Z}.$$

We note that in addition to returning the marginal distribution $p(x_T)$, the algorithm also returns the normalization constant Z. The elimination algorithm is best understood in application to a particular undirected graph.

Example 9 (Elimination in action). Consider the 7-vertex undirected graph shown in Figure 2(a). As discussed in Example 6, it induces factorizations of the form

$$p(x_1,\dots,x_7) \propto \psi_{123}(x_1,x_2,x_3)\, \psi_{345}(x_3,x_4,x_5)\, \psi_{46}(x_4,x_6)\, \psi_{57}(x_5,x_7),$$

Suppose that our goal is to compute the marginal distribution $p(x_5)$. In order to do so, we initialize the elimination algorithm with $T = 5$, the elimination ordering $\mathcal{I} = \{5,4,2,3,1,6,7\}$, and the active set $\mathcal{A} = \{\psi_{123},\psi_{345},\psi_{46},\psi_{57}\}$.

Eliminating x_7**:** At the first step, $s = 7$ becomes the active vertex. Since ψ_{57} is the only function involving x_7, we form $\phi(x_5,x_7) = \psi_{57}(x_5,x_7)$, and then compute the partial sum $\widetilde{\phi}_5(x_5) = \sum_{x_7} \psi_{57}(x_5,x_7)$. We then form the new active set $\mathcal{A} = \{\psi_{123},\psi_{345},\psi_{46},\widetilde{\phi}_5\}$.
Eliminating x_6**:** This phase is very similar. We compute the partial sum $\widetilde{\phi}_4(x_4) = \sum_{x_6} \psi_{46}(x_4,x_6)$, and end up with the new active set $\mathcal{A} = \{\psi_{123},\psi_{345},\widetilde{\phi}_5,\widetilde{\phi}_4\}$.
Eliminating x_1**:** We compute $\widetilde{\phi}_{23}(x_2,x_3) = \sum_{x_1} \psi_{123}(x_1,x_2,x_3)$, and conclude with the new active set $\mathcal{A} = \{\widetilde{\phi}_{23},\psi_{345},\widetilde{\phi}_5,\widetilde{\phi}_4\}$.
Eliminating x_3**:** In this phase, we first form the product $\phi_{2345}(x_2,x_3,x_4,x_5) = \widetilde{\phi}_{23}(x_2,x_3)\psi_{345}(x_3,x_4,x_5)$, and then compute the partial sum

$$\widetilde{\phi}_{245}(x_2,x_4,x_5) = \sum_{x_2} \phi_{2345}(x_2,x_3,x_4,x_5).$$

We end up with the active set $\mathcal{A} = \{\widetilde{\phi}_{245},\widetilde{\phi}_5,\widetilde{\phi}_4\}$.
Eliminating x_2**:** The output of this phase is the active set $\mathcal{A} = \{\widetilde{\phi}_{45},\widetilde{\phi}_5,\widetilde{\phi}_4\}$, where $\widetilde{\phi}_{45}$ is obtained by summing out x_2 from $\widetilde{\phi}_{245}$.
Eliminating x_4**:** We form the product $\phi_{45}(x_4,x_5) = \widetilde{\phi}_4(x_4)\widetilde{\phi}_{45}(x_4,x_5)$, and then the partial sum $\widehat{\phi}(x_5) = \sum_{x_4} \phi_{45}(x_4,x_5)$. We conclude with the active set $\mathcal{A} = \{\widetilde{\phi}_5,\widehat{\phi}_5\}$.

After these steps, the index set $\mathcal{I}$ has been reduced to the target vertex $T = 5$. In the final step, we form the product $\phi_5(x_5) = \widetilde{\phi}_5(x_5)\widehat{\phi}_5(x_5)$, and then re-normalize it to sum to one. ♣

A few easy extensions of the elimination algorithm are important to note. Although we have described it in application to an undirected graphical model, a simple pre-processing step allows it to be applied to any directed graphical model as well. More precisely, any directed graphical model can be converted to an undirected graphical model via the following steps:

- remove directions from all the edges
- "moralize" the graph by adding undirected edges between all parents of a given vertex—that is, in detail

$$\text{for all } s \in \mathcal{V}, \text{ and for all } t,u \in \pi(s), \text{ add the edge } (t,u).$$

- the added edges yield cliques $C = (s, \pi(s))$; on each such clique, define the compatibility function $\psi_C(x_C) = p(x_s \,|\, x_{\pi(s)})$.

Applying this procedure yields an undirected graphical model to which the elimination algorithm can be applied.

A second extension concerns the computation of conditional distributions. In applications, it is frequently the case that some subset of the variables are observed, and our goal is to compute a conditional distribution of X_T given these observed variables. More precisely, suppose that we wish to compute a conditional probability of the form $p(x_T \mid \bar{x}_O)$ where $O \subset \mathcal{V}$ are indices of the observed values $X_O = \bar{x}_O$. In order to do so, it suffices to compute a marginal of the form $p(x_T, \bar{x}_O)$. This marginal can be computed by adding binary indicator functions to the active set given as input to the elimination algorithm. In particular, for each $s \in O$, let $\mathbb{I}_{s;\bar{x}_s}(x_s)$ be a zero-one indicator for the event $\{x_s = \bar{x}_s\}$. We then provide the elimination algorithm with the initial active set $\mathcal{A} = \{\psi_C, C \in \mathfrak{C}\} \cup \{\mathbb{I}_{s,\bar{x}_s}, s \in O\}$, so that it computes marginals for the modified distribution

$$q(x_1, \ldots, x_N) \propto \prod_{C \in \mathfrak{C}} \psi_C(x_C) \prod_{s \in O} \mathbb{I}_{s\bar{x}_s}(x_s),$$

in which $q(x) = 0$ for all configurations $x_O \neq \bar{x}_O$. The output of the elimination algorithm can also be used to compute the likelihood $p(\bar{x}_O)$ of the observed data.

3.1.1 Graph-theoretic versus analytical elimination

Up to now, we have described the elimination algorithm in an analytical form. It can also be viewed from a graph-theoretic perspective, namely as an algorithm that eliminates vertices, but adds edges as it does so. In particular, during the elimination step, the graph-theoretic version removes the active vertex s from the current elimination order (as well as all edges involving s), but then adds an edge for each pair of vertices (t,u) that were connected to s in the current elimination graph. These adjacent vertices define the neighborhood set $\mathcal{N}(s)$ used in Step 2(c) of the algorithm. This dual perspective—the analytical and graph-theoretic—is best illustrated by considering another concrete example.

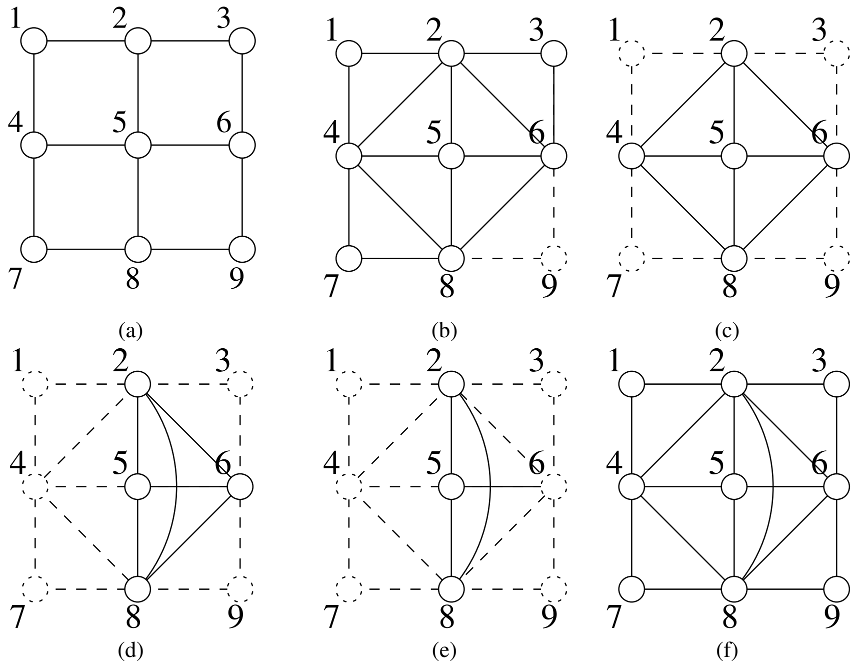

Fig. 5. (a) A 3×3 grid graph in the plane. (b) Result of eliminating vertex 9. Dotted vertices/edges indicate quantities that have been removed during elimination. (c) Remaining graph after having eliminated vertices $\{1,3,7,9\}$. (d) Resulting graph after eliminating vertex 4. (e) Elimination of vertex 6. (f) Final reconstituted graph: original graph augmented with all edges added during the elimination process.

Example 10 (Graph-theoretic versus analytical elimination). Consider the 3×3 grid-structured graph shown in panel (a) of Figure 5. Given a probability distribution that is Markov with respect to this grid, suppose that we want to compute the marginal distribution $p(x_5)$, so that the target vertex is $T = 5$. In order to do so, we initialize the elimination algorithm using the ordering $\mathcal{I} = \{5,2,8,6,4,1,3,7,9\}$. Panel (b) shows the graph obtained after eliminating vertex 9, now shown in dotted lines to indicate that it has been removed. At the graph-theoretic level, we have removed edges $(8,9)$ and $(6,9)$, and added edge $(6,9)$ At the analytical level, we have performed the summation over x_9 to compute

$$\widetilde{\phi}(x_6,x_8) = \sum_{x_9} \psi_{69}(x_6,x_9)\psi_{89}(x_8,x_9),$$

and added it to the active list of compatibility functions. Notice how the new function $\widetilde{\phi}_{68}$ depends only on the variables (x_6,x_8), which are associated with the edge $(6,8)$ added at the graph-theoretic level. Panel (c) shows the result after having eliminated vertices $\{1,3,7,9\}$, in reverse order. Removal of the three vertices (in addition to 9) has a symmetric effect on each corner of the graph. In panel (d), we see the effect of eliminating vertex 4; in this case, we remove edges $(4,5)$, $(4,2)$, and $(4,8)$, and then add edge $(2,8)$. At the analytical stage, this step will introduce a new compatibility function of the form $\widetilde{\phi}_{258}(x_2,x_5,x_8)$. Panel (e) shows the result of removing vertex 6; no additional edges are added at this stage. Finally, panel (f) shows the *reconstituted graph* $\widetilde{\mathcal{G}}$, meaning the original graph plus all the edges that were added during the elimination process. Note that the added edges have increased the size of the maximal clique(s) to four; for instance, the set $\{2,5,6,8\}$ is a maximal clique in the reconstituted graph. This reconstituted graph plays an important role in our discussion of junction tree theory to follow. ♣

3.1.2 Complexity of elimination

What is the computational complexity of the elimination algorithm? Let us do some rough calculations in the discrete case, when $\mathcal{X}_s = \{0,1,\ldots,m-1\}$ for all vertices $s \in \mathcal{V}$. Clearly, it involves at least a linear cost in the number of vertices N. The other component is the cost of computing the partial sums $\widetilde{\phi}$ at each step. Supposing that we are eliminating s, let $\mathcal{N}(s)$ be the current set of its neighbors in the elimination graph. For instance, in Figure 5, when we eliminate vertex $s = 4$, its current neighborhood set is $\mathcal{N}(4) = \{2,5,8\}$, and the function $\widetilde{\phi}_{258}$ at this stage depends on (x_2,x_5,x_8). Computing $\widetilde{\phi}_{258}$ incurs a complexity of $\mathcal{O}(m^4)$, since we need to perform a summation (over x_4) involving m terms for each of the m^3 possible instantiations of (x_2,x_5,x_8).

In general, then, the worst-case cost over all elimination steps will scale as $\mathcal{O}(m^{|\mathcal{N}(s)|+1})$, where $|\mathcal{N}(s)|$ is the neighborhood size in the elimination graph when eliminating s. It can be seen that the worst-case neighborhood size (plus one) is equivalent to the size of the largest clique in the reconstituted graph. For instance, in Figure 5(f), the largest cliques in the reconstituted graph have size 4, consistent with the $\mathcal{O}(m^4)$ calculation from above. Putting together the pieces, we obtain a conservative upper bound on the cost of the elimination algorithm—namely, $\mathcal{O}(m^c N)$, where c is the size of the largest clique in the reconstituted graph.

A related point is that for any given target vertex T, there are many possible instantiations of the elimination algorithm that can be used to compute the marginal $p(x_T)$. Remember that our only requirement is that T appear first in the ordering, so that there are actually $(N-1)!$ possible orderings of the remaining vertices. Which ordering should be chosen? Based on our calculations above, in order to minimize the computational cost, one desirable goal would be to minimize the size of the largest clique in the reconstituted graph. In general, finding an optimal elimination ordering of this type is computationally challenging; however, there exist a variety of heuristics for choosing good orderings, as we discuss in Section 4.

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

$$p(x_1, \ldots, x_N) = \frac{1}{Z} \prod_{s \in \mathcal{V}} \psi_s(x_s) \prod_{(s,t) \in \mathcal{E}} \psi_{st}(x_s, x_t), \tag{19}$$

where $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ is a given tree, and $\{\psi_s, s \in \mathcal{V}\}$ and $\{\psi_{st}, (s,t) \in \mathcal{E}\}$ are compatibility functions. For a tree, the elimination algorithm takes a very simple form—assuming that the appropriate ordering is followed. Recall that a vertex 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 $\mathcal{I}$ such that:

(a) The last vertex in $\mathcal{I}$ (first to be eliminated) is a leaf; and
(b) For each successive stage of the elimination algorithm, the last vertex in $\mathcal{I}$ remains a leaf of the reduced graph.

This claim follows because any tree always has at least one leaf vertex [17], so that the algorithm can be initialized in Step 1. Moreover, in removing a leaf vertex, the elimination algorithm introduces no additional edges to the graph, so that it remains a tree at each stage. Thus, we may apply the argument recursively, again choosing a leaf vertex to continue the process. We refer to such an ordering as a *leaf-exposing ordering*.

In computational terms, when eliminating leaf vertex s from the graph, the algorithm marginalizes over x_s, and then passes the result to its unique parent $\pi(s)$. The result $\widetilde{\phi}_{\pi(s)}$ of this intermediate computation is a function of $x_{\pi(s)}$; following standard notation used in presenting the sum-product algorithm, we can represent it by a "message" of the form $M_{s \to \pi(s)}$, where the subscript reflects the direction in which the message is passed. Upon termination, the elimination algorithm will return the marginal distribution $p(x_T)$ at the target vertex $T \in \mathcal{V}$; more specifically, this marginal is specified in terms of the local compatibility function ψ_T and the incoming messages as follows:

$$p(x_T) \propto \psi_T(x_T) \prod_{s \in \mathcal{N}(r)} M_{s \to T}(x_T).$$

Assuming that each variable takes at most m values (i.e., $|\mathcal{X}_s| \le m$ for all $s \in \mathcal{V}$), the overall complexity of running the elimination algorithm for a given root vertex scales as $O(m^2 N)$, since computing each message amounts to summing m numbers a total of m times, and there are $O(N)$ rounds in the elimination order. Alternatively, since the leaf-exposing ordering adds no new edges to the tree, the reconstituted graph is *equivalent* to the original tree. The maximal cliques in any tree are the edges, hence of size two, so that the complexity estimate $O(m^2 N)$ follows from our discussion of the elimination algorithm.

3.2.1 Sum-product algorithm

In principle, by running the tree-based elimination algorithm N times—once for each vertex acting as the target in the elimination ordering—we could compute all singleton marginals in time $O(m^2 N^2)$. 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 $O(m^2 N)$. In fact, as an added bonus, without any added complexity, it also computes *all pairwise marginal distributions* over variables (x_s, x_t) such that $(s,t) \in \mathcal{E}$.

For each edge $(s,t) \in \mathcal{E}$, the sum-product algorithm maintains two messages—namely, $M_{s\to t}$ and $M_{t\to s}$—corresponding to two possible directions of the edge.[1] With this notation, the sum-product message-passing algorithm for discrete random variables takes the following form:

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

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

$$M^0_{t\to s}(x_s) = 1 \quad \text{for all } x_s \in \mathcal{X}_s \quad \text{and} \quad M^0_{s\to t}(x_t) = 1 \quad \text{for all } x_t \in \mathcal{X}_t.$$

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

$$M^k_{t\to s}(x_s) \leftarrow \alpha \sum_{x'_t} \Big\{ \psi_{st}(x_s, x'_t)\, \psi_t(x'_t) \prod_{u \in \mathcal{N}(t)/s} M^{k-1}_{u\to t}(x'_t) \Big\}, \tag{20}$$

   where $\alpha > 0$ chosen such that $\sum_{x_s} M^k_{s\to s}(x_s) = 1$.
   (ii) Upon convergence, compute marginal distributions:

$$p(x_s) = \alpha_s\, \psi_s(x_s) \prod_{t \in \mathcal{N}(s)} M_{t\to s}(x_s), \quad \text{and} \tag{21a}$$

$$p_{st}(x_s, x_t) = \alpha_{st}\, \psi_{st}(x_s, x_t) \prod_{u \in \mathcal{N}(s)\backslash\{t\}} M_{u\to s}(x_s) \prod_{u \in \mathcal{N}(t)\backslash\{s\}} M_{u\to t}(x_t), \tag{21b}$$

[1] Recall that the message $M_{s\to t}$ is a vector of $|\mathcal{X}_t|$ numbers, one for each value $x_s \in \mathcal{X}_t$.

where $\alpha_s > 0$ and $\alpha_{st} > 0$ are normalization constants (to ensure that the marginals sum to one).

The initialization given in Step 1 is the standard "uniform" one; it can be seen that the final output of the algorithm will be same for any initialization of the messages that has strictly positive components. As with our previous discussion of the elimination algorithm, we have stated a form of message-passing updates suitable for discrete random variables, involving finite summations. For continuous random variables, we simply need to replace the summations in the message update (20) with integrals. For instance, in the case of a Gauss-Markov chain, we obtain the Kalman filter as a special case of the sum-product updates.

There are various ways in which the order of message-passing can be scheduled. The simplest to describe, albeit not the most efficient, is the *flooding schedule*, in which every message is updated during every round of the algorithm. Since each edge is associated with two messages—one for each direction—this protocol involves updating $2|\mathcal{E}|$ messages *per iteration*. The flooding schedule is especially well-suited to parallel implementations of the sum-product algorithm, such as on a digital chip or field programmable gate array, in which a single processing unit can be devoted to each vertex of the graph.

In terms of minimizing the total number of messages that are passed, a more efficient scheduling protocol is based on the following rule: any given vertex s updates the message $M_{s\to t}$ to its neighbor $t \in \mathcal{N}(s)$ only after it has received messages from all its other neighbors $u \in \mathcal{N}(s)\backslash\{t\}$. In order to see that this protocol will both start and terminate, note that any tree has at least one leaf vertex. All leaf vertices will pass a message to their single neighbor at round 1. As in the elimination algorithm, we can then remove these vertices from further consideration, yielding a sub-tree of the original tree, which also has at least one leaf node. Again, we see that a recursive leaf-stripping argument will guarantee termination of the algorithm. Upon termination, a total of $2|\mathcal{E}|$ messages have been passed *over all iterations*, as opposed to per iteration in the flooding schedule.

Regardless of the particular message-update schedule used, the following result states the convergence and correctness guarantees associated with the sum-product algorithm on a tree:

Proposition 2. *For any tree $\mathcal{T}$ with diameter $d(\mathcal{T})$, the sum-product updates have a unique fixed point M^*. The algorithm converges to it after at most $d(\mathcal{T})$ iterations, and the marginals obtained by equations* (21a) *and* (21b) *are exact.*

Proof. We proceed via induction on the number of vertices. For $N = 1$, the claim is trivial. Now assume that the claim holds for all trees with at most $N-1$ vertices, and let us show that it also holds for any tree with N vertices. It is an elementary fact of graph theory [17] that any tree has at least one leaf node. By re-indexing as necessary, we may assume that node N is a leaf, and its unique parent is node 1. Moreover, by appropriately choosing the leaf vertex, we may assume that the diameter of $\mathcal{T}$ is achieved by a path ending at node N. By definition of the sum-product updates, for all iterations $k = 1, 2, \ldots$, the message sent from N to 1 is given by

$$M^*_{N \to 1}(x_1) = \alpha \sum_{x_N} \psi_N(x_N) \psi_{N1}(x_N, x_1).$$

(It never changes since vertex N has only one neighbor.) Given this fixed message, we may consider the probability distribution defined on variables $(x_1, \ldots, x_{N-1})$ given by

$$p(x_1, \ldots, x_{N-1}) \propto M^*_{N \to 1}(x_1) \, [\prod_{s=1}^{N-1} \psi_s(x_s)] \prod_{(s,t) \in \mathcal{E} \backslash \{(1,N)\}} \psi_{st}(x_s, x_t).$$

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

By the induction hypothesis, the sum-product algorithm applied to $\mathcal{T}'$ will converge after at most $d(\mathcal{T}')$ iterations to a fixed point $M^* = \{M^*_{s \to t}, M^*_{t \to s} \mid (s,t) \in \mathcal{E}'\}$, and this fixed point will yield the correct marginals at all vertices $s \in \mathcal{V}'$ and edges $(s,t) \in \mathcal{E}'$. As previously discussed, the message from N to 1 remains fixed for all iterations after the first, and the message from 1 to N will be fixed once the iterations on the sub-tree $\mathcal{T}'$ have converged. Taking into account the extra iteration for node 1 to N, we conclude that sum-product on $\mathcal{T}$ converges in at most $d(\mathcal{T})$ iterations as claimed.

It remains to show that the marginal at node N can be computed by forming the product $p(x_N) \propto \psi_N(x_N) M^*_{1 \to N}(x_N)$, where

$$M^*_{1 \to N}(x_N) := \alpha \sum_{x_1} \psi_1(x_1) \, \psi_{1N}(x_1, x_N) \prod_{u \in \mathcal{N}(1) \backslash \{N\}} M^*_{u \to 1}(x_1).$$

By elementary probability theory, we have $p(x_N) = \sum_{x_1} p(x_N \mid x_1) p(x_1)$. Since N is a leaf node with unique parent 1, we have the conditional independence relation $X_N \perp\!\!\!\perp X_{\mathcal{V} \backslash \{1\}} \mid X_1$, and hence, using the form of the factorization (19),

$$p(x_N \mid x_1) = \frac{\psi_N(x_N) \psi_{1N}(x_1, x_N)}{\sum_{x_N} \psi_N(x_N) \psi_{1N}(x_1, x_N)} \propto \frac{\psi_N(x_N) \psi_{1N}(x_1, x_N)}{M^*_{N \to 1}(x_1)}. \qquad (22)$$

By the induction hypothesis, the sum-product algorithm, when applied to the distribution (22), returns the marginal distribution

$$p(x_1) \propto [M^*_{N\to 1}(x_1)\psi_1(x_1)] \prod_{t\in\mathcal{N}(1)\backslash\{N\}} M^*_{t\to 1}(x_1).$$

Combining the pieces yields

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

as required. A similar argument yields that equation (21b) yields the correct form of the pairwise marginal for the edge $(1,N) \in \mathcal{E}$; 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. A tree factor graph is simply a factor graph without cycles; see Figure 3(b) for an example. We consider a distribution with a factorization of the form

$$p(x_1,\ldots,x_N) = \frac{1}{Z}\prod_{s\in\mathcal{V}}\psi_s(x_s)\prod_{a\in\mathcal{F}}\psi_a(x_a), \tag{23}$$

where $\mathcal{V}$ and $\mathcal{F}$ are sets of vertices and factor nodes, respectively.

Before proceeding, it is worth noting that for the case of discrete random variables considered here, the resulting algorithm is not more general than sum-product for pairwise interactions. Indeed, in any distribution over discrete random variables with a tree-structured factor graph—regardless of the order of interactions that it involves—can be converted to an equivalent tree with pairwise interactions. We refer the reader to Appendix E.3 in the monograph [67] for further details on this procedure. Nonetheless, it is often convenient to apply the non-pairwise form of the sum-product algorithm directly to a factor tree, as opposed to going through this conversion.

We use $M_{s\to a}$ to denote the message from vertex s to the factor node $a \in \mathcal{N}(s)$, and similarly $M_{a\to s}$ to denote the message from factor node a to the vertex $s \in \mathcal{N}(a)$. Both messages (in either direction) are functions of x_s, where $M_{a\to s}(x_s)$ (respectively $M_{s\to a}(x_s)$) denotes the value taken for a given $x_s \in \mathcal{X}_s$. We let $x_a = \{x_s, s \in \mathcal{N}(a)\}$ denote the sub-vector 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:

$$M_{s\to a}(x_s) \leftarrow \alpha\, \psi_s(x_s) \prod_{b\in \mathcal{N}(s)\backslash\{a\}} M_{b\to s}(x_s), \quad \text{and} \tag{24a}$$

$$M_{a\to s}(x_s) \leftarrow \alpha \sum_{x_t,\, t\in\mathcal{N}(a)\backslash\{s\}} \Big[\psi_a(x_a) \prod_{t\in\mathcal{N}(a)\backslash\{s\}} M_{t\to a}(x_t)\Big]. \tag{24b}$$

Here the quantity α 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_s and over the variables x_a are given, respectively, by the quantities

$$p(x_s) = \alpha\, \psi_s(x_s) \prod_{a\in\mathcal{N}(s)} M_{a\to s}(x_s), \quad \text{and} \tag{25a}$$

$$p(x_a) = \alpha\, \psi_a(x_a) \prod_{t\in\mathcal{N}(a)} M_{t\to a}(x_t). \tag{25b}$$

We leave it as an exercise for the reader to verify that the message-passing updates (24) will converge after a finite number of iterations (related to the diameter of the factor tree), and when equation (25) 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.

The problem of mode computation amounts to finding a vector

$$x^* \in \arg\max_{x\in\mathcal{X}_{[N]}} p(x_1,\ldots,x_N),$$

where we recall our shorthand notation $\mathcal{X}_{[N]} = \mathcal{X}_1\times\mathcal{X}_2\times\cdots\times\mathcal{X}_N$. 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 $\mathcal{T} = (\mathcal{V},\mathcal{E})$. In particular, for each vertex $s\in\mathcal{V}$, we define the *singleton max-marginal*

$$\nu_s(x_s) := \max_{\{x'\in\mathcal{X}_{[N]}\,|\,x'_s = x_s\}} p(x'_1,\ldots,x'_N), \tag{26}$$

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

$$\nu_{ij}(x_i, x_j) := \max_{\{x' \in \mathcal{X}_{[N]} \,|\, (x'_i, x'_j) = (x_i, x_j)\}} p(x'_1, \ldots, x'_N), \tag{27}$$

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 $s \in \mathcal{V}$, the maximum over $\nu_s(x_s)$ is achieved at a single element, meaning that

$$\arg\max_{x_s \in \mathcal{X}_s} \nu_s(x_s) = x_s^* \quad \text{for all } s \in \mathcal{V}. \tag{28}$$

In this case, it is straightforward to compute the mode, as summarized in the following:

Lemma 3. *Consider a distribution p whose singleton max-marginals satisfy the unique maximizer condition* (28). *Then the vector $x^* = (x_1^*, \ldots, x_N^*) \in \mathcal{X}_{[N]}$ is the unique maximizer of $p(x_1, \ldots, x_N)$.*

Proof. The proof of this claim is straightforward. Note that for any vertex $s \in \mathcal{V}$, by the unique maximizer condition and the definition of the max-marginals, we have

$$\nu_s(x_s^*) \; = \; \max_{x_s \in \mathcal{X}_s} \nu_s(x_s) \; = \; \max_{x \in \mathcal{X}_{[N]}} p(x_1, \ldots, x_N).$$

Now for any other configuration $\tilde{x} \neq x^*$, there must be some index t such that $x_t^* \neq \tilde{x}_t$. For this index, we have

$$\max_{x \in \mathcal{X}_{[N]}} p(x_1, \ldots, x_N) \; = \; \nu_t(x_t^*) \; \overset{(i)}{>} \; \nu_t(\tilde{x}_t) \; \overset{(ii)}{\geq} \; p(\tilde{x}),$$

where inequality (i) follows from the unique maximizer condition, and inequality (ii) follows by definition of the max-marginal. We have thus established that $p(\tilde{x}) < \max_{x \in \mathcal{X}_{[N]}} p(x)$ for all $\tilde{x} \neq x^*$, so that x^* uniquely achieves the maximum. □

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* $x_s^* \in \arg\max_{x_s \in \mathcal{X}_i} \nu_i(x_s)$, as illustrated by the following toy example.

Example 11 (Failure of unique maximizer). Consider the distribution over a pair of binary variables given by

$$p(x_1, x_2) = \begin{cases} 0.45 & \text{if } x_1 \neq x_2, \text{ and} \\ 0.05 & \text{otherwise.} \end{cases} \tag{29}$$

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 (27). In order to do so, we consider a back-tracking procedure that samples some configuration $x^* \in \arg\max_{x \in \mathcal{X}_{[N]}} p(x)$. Let us assume that the tree is rooted at node 1, and that $\mathcal{V} = \{1, 2, \ldots, N\}$ is a topological ordering (i.e., such that $\pi(s) < s$ for all $s \in \{2, \ldots, N\}$). 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 $s = 2, \ldots, N$, choose a configuration x_s^* at vertex s such that

$$x_s^* \in \arg\max_{x_s \in \mathcal{X}_s} \nu_{s,\pi(s)}(x_s, x_{\pi(s)}^*).$$

Note that this is a recursive procedure, in that the choice of the configuration x_s^* depends on the previous choice of $x_{\pi(s)}^*$. Our choice of topological ordering ensures that $x_{\pi(s)}^*$ is always fixed before we reach vertex s. We establish that this back-tracking procedure is guaranteed to output a vector $x^* \in \arg\max_{x \in \mathcal{X}_{[N]}} p(x)$ 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*:

$$M_{t \to s}(x_s) \leftarrow \alpha \max_{x_t} \Big\{ \psi_{st}(x_s, x_t) \psi_t(x_t) \prod_{u \in \mathcal{N}(t)/s} M_{u \to t}(x_t) \Big\}, \tag{30}$$

where $\alpha > 0$ chosen such that $\max_{x_s} M_{t \to s}(x_s) = 1$. When the algorithm converges to a fixed point M^*, we can compute the max-marginals via equations (21a) and (21b).

Let us summarize the properties of the max-product algorithm on any undirected tree:

Proposition 3. *For any tree $\mathcal{T}$ with diameter $d(\mathcal{T})$, the max-product updates converge to their unique fixed point M^* after at most $d(\mathcal{T})$ iterations. Moreover, equations* (21a) *and* (21b) *can be used to compute the max-marginals, and the back-tracking procedure yields an element $x^* \in \arg\max_{x \in \mathcal{X}_{[N]}} p(x)$.*

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 $x^* \in \mathcal{X}_{[N]}$ 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 $\mathcal{V} = \{1, 2, \dots, N\}$ 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:

$$J(x_1, \dots, x_N; \nu) = \nu_1(x_1) \prod_{s=2}^{N} \frac{\nu_{s\pi(s)}(x_s, x_{\pi(s)})}{\nu_{\pi(s)}(x_{\pi(s)})}. \tag{31}$$

In order for division to be defined in all cases, we take $0/0 = 0$.

We first claim that there exists a constant $\alpha > 0$ such that $J(x; \nu) = \alpha\, p(x)$ for all $x \in \mathcal{X}_{[N]}$, which implies that $\arg\max_{x \in \mathcal{X}_{[N]}} J(x; \nu) = \arg\max_{x \in \mathcal{X}_{[N]}} p(x)$. By virtue of this property, we say that ν defines a *reparameterization* of the original distribution. (See the papers [65, 66] for further details on this property and its utility in analyzing the sum and max-product algorithms.)

To establish the reparameterization property, we first note that the cost function J can also be written in the symmetric form

$$J(x; \nu) = \prod_{s=1}^{N} \nu_s(x_s) \prod_{(s,t) \in \mathcal{E}} \frac{\nu_{st}(x_s, x_t)}{\nu_s(x_s)\nu_t(x_t)}. \tag{32}$$

We then recall that the max-marginals are specified by the message fixed point M^* via equations (21a) and (21b). Substituting these relations into the symmetric form (32) of J yields

$$\begin{aligned} J(x; \nu) &\propto \Big[\prod_{s \in \mathcal{V}} \psi_s(x_s) \prod_{u \in \mathcal{N}(s)} M^*_{u \to s}(x_s)\Big]\Big[\prod_{(s,t) \in \mathcal{E}} \frac{\psi_{st}(x_s, x_t)}{M^*_{s \to t}(x_t) M^*_{t \to s}(x_s)}\Big] \\ &= \prod_{s \in \mathcal{V}} \psi_s(x_s) \prod_{(s,t) \in \mathcal{E}} \psi_{st}(x_s, x_t) \\ &\propto p(x), \end{aligned}$$

as claimed.

Consequently, it suffices to show that any $x^* \in \mathcal{X}_{[N]}$ returned by the back-tracking procedure is an element of $\arg\max_{x \in \mathcal{X}_{[N]}} J(x;\nu)$. We claim that for each vertex $s \in \mathcal{V}$ such that $\pi(s) \neq \emptyset$,

$$\frac{\nu_{s\pi(s)}(x_s^*, x_{\pi(s)}^*)}{\nu_{\pi(s)}(x_{\pi(s)}^*)} \geq \frac{\nu_{s\pi(s)}(x_s, x_{\pi(s)})}{\nu_{\pi(s)}(x_{\pi(s)})} \quad \text{for all } (x_s, x_{\pi(s)}) \in \mathcal{X}_s \times \mathcal{X}_{\pi(s)}. \tag{33}$$

By definition of the max-marginals and our choice of x^*, the left-hand side is equal to one. On the other hand, the definition of the max-marginals implies that $\nu_{\pi(s)}(x_{\pi(s)}) \geq \nu_{s\pi(s)}(x_s, x_{\pi(s)})$, 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 (33) and the definition (31), implies that $J(x^*;\nu) \geq J(x;\nu)$ for all $x \in \mathcal{X}_{[N]}$, showing that x^* is an element of the set $\arg\max_{x \in \mathcal{X}_{[N]}} J(x;\nu)$, 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, 28, 57, 63] 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 vertices 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 with vertex set $\mathcal{V} = \{1,2,3,4,5\}$ shown in Figure 6(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 vertices 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).

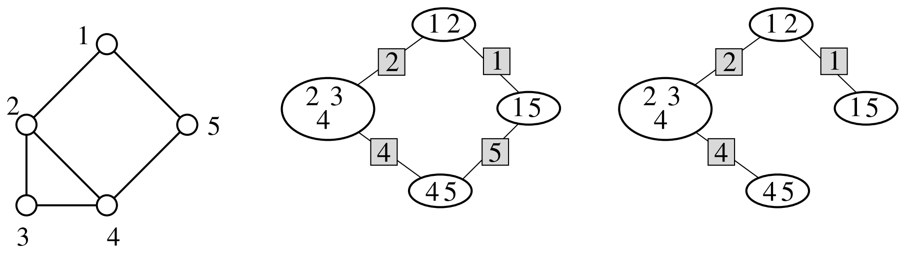

Fig. 6. (a) A graph with cycles on five vertices. (b) Associated clique graph in which circular vertices are maximal cliques of $\mathcal{G}$; square gray boxes sitting on the edges represent separator sets, corresponding to the intersections between adjacent cliques. (c) One clique tree extracted from the clique graph. It can be verified that this clique tree fails to satisfy the running intersection property.

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

$$p(x_1,\ldots x_5) \propto \psi_{234}(x_2,x_3,x_4)\psi_{12}(x_1,x_2)\psi_{15}(x_1,x_5)\psi_{45}(x_4,x_5). \tag{34}$$

We can 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 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. In our particular example, we introduce extra variables x'_s for $s \in \{1,2,4,5\}$, since these variables appear twice in the clique graph. We then form a distribution $q(x,x') = q(x_1,x_2,x_3,x_4,x_5 x'_1,x'_2,x'_4,x'_5)$ that factorizes as

$$q(x, x') \propto \psi_{234}(x_2, x_3, x_4)\psi_{12}(x_1, x_2')\psi_{15}(x_1', x_5)\psi_{45}(x_4', x_5) \times \prod_{s \in \mathcal{V} \backslash \{3\}} \mathbb{I}[x_s = x_s'], \tag{35}$$

where $\mathbb{I}[x_s = x_s']$ is a $\{0,1\}$-valued indicator function for the event $\{x_s = x_s'\}$. By construction, for any $s \in \{1,2,3,4,5\}$, if we were to compute the distribution $q(x_s)$ in the expanded model (35), it would be equal to the marginal $p(x_s)$ from the original model (34). Although the expanded model (35) involves additional copies x_s', the indicator functions that have been added enforce the needed equalities to maintain model consistency.

However, the clique graph in panel (b) and the distribution (35) are not still *not* 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 (34)? Panel (c) of Figure 6 shows one clique tree, obtained by dropping the edge labeled with separator set $\{5\}$ that connects the vertices 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

$$r(x, x') \propto \psi_{234}(x_2, x_3, x_4)\psi_{12}(x_1, x_2')\psi_{15}(x_1', x_5)\psi_{45}(x_4', x_5) \times \prod_{s \in \mathcal{V} \backslash \{3,5\}} \mathbb{I}[x_s = x_s'], \tag{36}$$

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 (34). This problem occurs because—in sharp contrast to the clique graph factorization (35)—the distribution (36) might assign non-zero probability to some configuration for which $x_5 \neq x_5'$. Of course, the clique tree shown in Figure 6(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 $x_s = x_s' = x_s'' = \ldots$, for all copies of a given variable indexed by some $s \in \mathcal{V}$. (Although variables only appeared twice in the example of Figure 6, the clique graphs obtained from more complicated graphs could have any number of copies.) Note that there are copies x_s and x_s' of a given variable x_s if and only if s 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 vertices, and what we require is that the equivalence $\{x_s = x_s'\}$ be propagated along this path. In graph-theoretic terms, the required property can be formalized as follows:

Definition 6. A clique tree has the *running intersection property* if for any two clique vertices C_1 and C_2, all vertices 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 6 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.

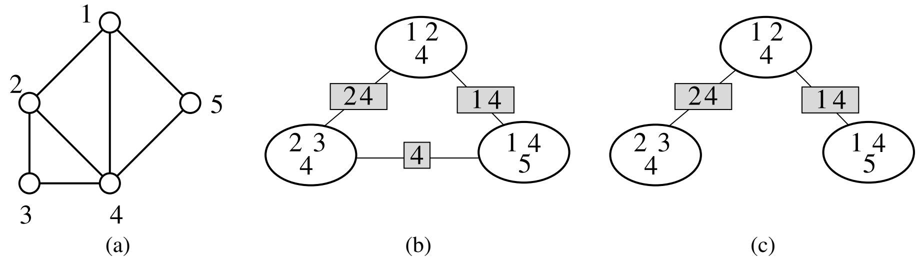

Fig. 7. (a) A modified version of the graph from Figure 6(a), containing the extra edge $(1,4)$. The modified graph has three maximal cliques, each of size three. (b) The associated clique graph has one node for each of three maximal cliques, with gray boxes representing the separator sets. (c) A clique tree extracted from the clique graph; it can be verified that this clique tree satisfies the running intersection property.

Figure 7(a) shows a modified version of the graph from Figure 6(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 6(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 6(a)) for which it was impossible to obtain a junction tree, and a second (Figure 7(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*.

A cycle in a graph is a sequence of vertices $(s_1, s_2, \dots s_\ell, s_1)$ such that $(s_\ell, s_1) \in \mathcal{E}$ and $(s_i s_{i+1}) \in \mathcal{E}$ for all $i = 1, \dots, \ell - 1$. 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 6(a) is chordless, whereas in contrast, the cycle $(1, 2, 3, 4, 5, 1)$ contains the chord $(2, 4)$.

Definition 7. A graph is *triangulated* if it contains no chordless cycles of length greater than three.

Thus, the graph in Figure 6(a) is not triangulated (due to the chordless cycle $(1, 2, 4, 5, 1)$), whereas it can be verified that the graph in Figure 7(a) is triangulated. We now state the fundamental connection between triangulation and junction trees:

Theorem 3. *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 A. The practical significance of Theorem 3 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 4. *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 4 as an exercise for the reader. This fact establishes that there is a connection between the elimination algorithm and the property of triangulation.

Example 12 (Triangulation and junction tree). To illustrate the triangulation procedure, consider the 3×3 grid shown in Figure 8(a). If we run the vertex elimination algorithm using the ordering $\mathcal{I} = \{1, 3, 7, 9, 4, 6, 2, 8\}$, 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 vertices 2 and 8 were not present. Without this edge, the 4-cycle

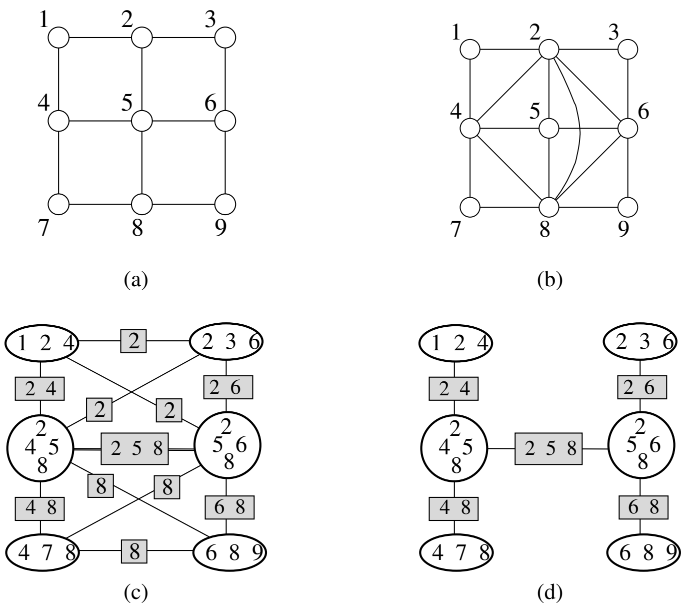

Fig. 8. Illustration of junction tree construction. (a) Original graph is a 3×3 grid. (b) Triangulated version of original graph. Note the two 4-cliques in the middle. (c) Corresponding clique graph for triangulated graph in (b), with maximal cliques depicted within ellipses, and separator sets within rectangles. (d) A junction tree extracted from the clique graph in (c). It is a clique tree that satisfies the running intersection property.

$\{2,4,8,6,2\}$ would lack a chord). Note that $\widetilde{\mathcal{G}}$ has six maximal cliques: two 4-cliques 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 that this clique tree satisfies the running intersection property, and so is a junction tree. ♣

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 vertices 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 vertices. Let us consider an arbitrary spanning tree T of the weighted clique graph, say with vertices $\{C_1, \ldots, C_M\}$ 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 $w(\mathcal{T}) = \sum_{j=1}^{M-1} \text{card}(S_j)$. 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

$$w(T) = \sum_{j=1}^{M-1} \sum_{t \in \mathcal{V}} \mathbb{I}[t \in S_j] \; = \; \sum_{t \in \mathcal{V}} \sum_{j=1}^{M-1} \mathbb{I}[t \in S_j]. \tag{37}$$

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

$$\sum_{j=1}^{M-1} \mathbb{I}[t \in S_j] \leq \sum_{i=1}^{M} \mathbb{I}[t \in C_i] - 1, \tag{38}$$

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 vertices 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 (38) 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 (38) into the earlier inequality (37), 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. □

Putting together the pieces of our development, we are led to the following algorithm:

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., using Kruskal's algorithm, or Prim's algorithm).
3. Given the junction tree, define appropriate compatibility functions on its vertices and edges to reproduce the original distribution, and run the sum-product algorithm.

There are various techniques for finding *some* triangulation of a graph; as noted in Lemma 4, 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 8. The *treewidth* $\kappa(\mathcal{G})$ of a undirected graph $\mathcal{G}$ 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 7(a) has treewidth two, and the grid in Figure 8 has treewidth three.

For any fixed treewidth $\kappa = \kappa(\mathcal{G})$, there are efficient algorithms—meaning polynomial-time in the graph size—to test whether the graph has treewidth κ, and if so, to determine an associated junction tree (e.g., [62, 16]). However, the complexity of these algorithms grows exponentially in the treewidth κ, so they are feasible only for graphs of bounded treewidth. For a general graph, the problem of finding an optimal triangulation is NP-hard [68, 6]. Despite this negative result, there are a variety of heuristic algorithms for obtaining "good" triangulations of a graph. Some of most widely studied are the minimum degree and the minimum fill heuristics [36, 4]. The minimum degree heuristic is based on following the elimination ordering obtained by choosing a minimum degree vertex (in the partially reduced graph) at each step. The minimum fill heuristic operates similarly, but instead is based on choosing a vertex s so as to minimize the total number of edges required to make the neighborhood $\mathcal{N}(s)$ a clique, a quantity known as the "fill".

5 Basics of graph estimation

Up to this point, we have considered various types of "forward problems", by which we mean that the graphical model—both the graph and the functions in its factorization—are known quantities. In this section, we turn our attention to various classes of "inverse problems", in which either the functions and/or the structure of the graph are unknown. Given a black box that generates samples from the distributions, we consider two possible problems:

- assuming that the clique structure is known, how to estimate the functions in the factorization?
- given no *a priori* information, how to estimate the clique structure and the functions?

The former problem is one of parameter estimation, since it involves estimating the parameters that specify either the conditional probabilities (for a directed graphical model) or the compatibility functions (for an undirected graphical model). In contrast, the latter problem is a form of model selection, since we need to select the correct graph from competing families of graphs. In the machine learning community, the latter problem is also known as structure learning. We begin our discussion with the former problem, as it is simpler both conceptually and computationally.

5.1 Parameter estimation for directed graphs

The problem of parameter estimation is particularly simple for directed graphical models, in which case, as long as the data is fully observed, the solution can be obtained in closed form.

More concretely, for a known directed graph $\mathcal{D} = (\mathcal{V}, \overrightarrow{\mathcal{E}})$, consider the set of all distributions that factorize in the form

$$p(x_1, \ldots, x_N) = \prod_{s \in \mathcal{V}} p(x_s \,|\, x_{\pi(s)}),$$

where $\pi(s) \subseteq \mathcal{V}$ is the parent set of vertex s. Suppose we observe a collection of n samples, each of the form $X_{i\bullet} = (X_{i1}, \ldots, X_{iN}) \in \mathcal{X}_{[N]}$. Given the collection of samples $X_{1\bullet}^{n\bullet} := \{X_{1\bullet}, X_{2\bullet}, \ldots, X_{n\bullet}\}$, our goal is to estimate the conditional distributions at each vertex $s \in \mathcal{V}$. In these notes, we focus exclusively on the case of discrete variables, in particular with $\mathcal{X}_s = \{0, 1, \ldots, m-1\}$ for all vertices $s \in \mathcal{V}$. (The extension to discrete variables with differing numbers of states per vertex is straightforward.)

By definition, each conditional distribution can be written as a ratio of two marginal distributions—namely, as

$$p(x_s \,|\, x_{\pi(s)}) = \frac{p(x_s, x_{\pi(s)})}{p(x_{\pi(s)})}.$$

A natural estimate of each marginal distribution are the *empirical marginal distributions*—namely, the quantities

$$\widehat{p}(x_s) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}(X_{i\,s} = x_s), \quad \text{and} \quad \widehat{p}(x_s, x_{\pi(s)}) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}(X_{i\,s} = x_s, X_{i\,\pi(s)} = x_{\pi(s)}),$$

where $\mathbb{I}(X_{i\,s} = x_s)$ is a zero-one valued indicator function for the event that $\{X_{i\,s} = x_s\}$, with the other indicator similarly defined. In words, the empirical marginal $\widehat{p}(x_s)$ is simply the relative fraction of times that $X_s = x_s$ in the data.

Given these estimates of the marginal distributions, a natural "plug-in" estimate of the conditional distribution is given by

$$\widehat{p}(x_s \mid x_{\pi(s)}) = \frac{\widehat{p}(x_s, x_{\pi(s)})}{\widehat{p}(x_{\pi(s)})}. \tag{39}$$

(To be clear, we take $0/0$ to be equal to 0, since these correspond to configurations that never appear in the data set.)

5.2 *Parameter estimation for undirected graphs*

For undirected graphical models, the problem of parameter estimation is not so simple in general. In particular, given an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, consider the set of all distributions that factorize in the form

$$p(x_1, \ldots, x_N; \psi) = \frac{1}{Z} \prod_{C \in \mathfrak{C}} \psi_C(x_C), \tag{40}$$

where $\mathfrak{C}$ is the set of all maximal cliques of the graph. In this setting, our goal is to estimate the unknown compatibility functions $\psi_C : \mathcal{X}_C \to (0, \infty)$ associated with each clique of the graph. A challenge here is that, at least in general, the compatibility functions are not directly related to conditional or marginal distributions of the graph. The exception to this rule is given by undirected trees (and more generally, junction trees), in which case there is always a choice of compatibility functions that is related to the marginal and/or conditional distributions.

In these notes, we limit our discussion to the method of *maximum likelihood* for estimating the compatibility functions. It is based on choosing the compatibility functions $\psi = \{\psi_C, C \in \mathfrak{C}\}$ so as to maximize the (rescaled) log likelihood of the data, namely the quantity

$$\ell(\psi; X_{1\bullet}, \ldots, X_{n\bullet}) = \frac{1}{n}\sum_{i=1}^{n} \log p(X_{i\bullet}; \psi). \tag{41}$$

We use $\widehat{\psi} = \{\widehat{\psi}_C, C \in \mathfrak{C}\}$ to denote the *maximum likelihood estimate*, namely a choice of compatibility functions that maximize the log likelihood. A remark about parameterization before proceeding: since we are considering discrete random variables, the estimate $\widehat{\psi}$ can be viewed as a vector of real numbers. In particular, the estimate $\widehat{\psi}_C$ can be represented as a sub-vector of $m^{|C|}$ real numbers, one for each possible configuration $J = \{J_s, s \in C\} \in \mathcal{X}_C$ that the variables x_C may assume. By concatenating all of these sub-vectors, we obtain a vector $\widehat{\psi} \in \mathbb{R}^D$, where $D = \sum_{C \in \mathfrak{C}} m^{|C|}$. Consequently, the rescaled log likelihood is a function from $\mathbb{R}^D$ to $\mathbb{R}$.

5.2.1 Maximum likelihood for undirected trees

Let us begin by considering the maximum likelihood problem for an undirected tree $\mathcal{T} = (\mathcal{V}, \mathcal{E})$, say with a factorization of the form

$$p(x_1, \ldots, x_N; \psi) = \frac{1}{Z}\prod_{s \in \mathcal{V}} \psi_s(x_s) \prod_{(s,t) \in \mathcal{E}} \psi_{st}(x_s, x_t). \tag{42}$$

For any tree, we claim that the maximum likelihood estimate $\widehat{\psi}$ can be expressed in terms of the empirical marginals at the vertices and edges of the tree—namely, the quantities

$$\widehat{p}(x_s = j) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}_{s;j}(X_{is}), \quad \text{and} \quad \widehat{p}((x_s, x_t) = (j,k)) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}_{st;jk}(X_{is}, X_{it}), \tag{43}$$

where $\mathbb{I}_{s;j}(x_s)$ is a zero-one indicator for the event that $\{x_s = j\}$, with $\mathbb{I}_{st;jk}(x_s, x_t)$ similarly defined. With these empirical marginals computed from the data, the maximum likelihood estimates of the compatibility functions are given by

$$\widehat{\psi}_s(x_s) = \widehat{p}(x_s), \quad \text{and} \quad \widehat{\psi}_{st}(x_s, x_t) = \frac{\widehat{p}(x_s, x_t)}{\widehat{p}(x_s)\widehat{p}(x_t)}. \tag{44}$$

(As before, we take $0/0 = 0$ for concreteness.) We prove in the following section, as a corollary of a more general result, that the relations (44) do in fact define a global maximizer of the log likelihood (41) for a tree-structured graph.

5.2.2 Maximum likelihood on general undirected graphs

In the case of graphs with cycles, the MLE no longer has a convenient closed-form expression. In order to appreciate the challenges, it is convenient to discuss the maximum likelihood objective using an alternative exponential parameterization. Let us do so for a graph with general clique set $\mathfrak{C}$, understanding that the undirected tree is a special case. For a given clique C and configuration $J = \{J_s, s \in C\} \in \mathcal{X}_C$, we define the function

$$\mathbb{I}_{C;J}(x_C) = \begin{cases} 1 & \text{if } x_C = J \\ 0 & \text{otherwise,} \end{cases} \tag{45}$$

corresponding to a binary-valued indicator for the event that $\{x_C = J\}$. Note that these functions are generalizations of the vertex-based functions $\mathbb{I}_{s;j}$ and edge-based functions $\mathbb{I}_{st;jk}$ discussed previously.

We use $\mathbb{I}_C = \{\mathbb{I}_{C;J}, J \in \mathcal{X}_C\}$ to denote the vector all of $m^{|C|}$ such indicator functions associated with clique C. With this notation, any function $\psi_C : \mathcal{X}_C \to (0, \infty)$ can be parameterized in the log-linear form

$$\log \psi_C(x_C) = \langle \theta_C, \mathbb{I}_C(x_C) \rangle := \sum_{J \in \mathcal{X}_C} \theta_{C;J} \mathbb{I}_{C;J}(x_C)$$

for some real vector $\theta_C = \{\theta_{C;J},\ J \in \mathcal{X}_C\}$ in $m^{|C|}$ dimensions. If we adopt this parameterization for each clique in the graph, then the overall factorization can be re-written in the form

$$p(x_1, \ldots, x_N; \theta) = \exp\Big\{ \sum_{C \in \mathfrak{C}} \langle \theta_C, \mathbb{I}_C(x_C) \rangle - \Phi(\theta) \Big\} \tag{46}$$

where $\theta = \{\theta_C, C \in \mathfrak{C}\}$ is a real-valued vector of parameters to be estimated, and

$$\Phi(\theta) := \log \sum_{x \in \mathcal{X}_{[N]}} \exp\Big\{ \sum_{C \in \mathfrak{C}} \langle \theta_C, \mathbb{I}_C(x_C) \rangle \Big\} \tag{47}$$

is the log normalization constant. The representation (46) shows that an undirected graphical model can be associated with an *exponential family*, namely one with sufficient statistics $\{\mathbb{I}_C(\cdot), C \in \mathfrak{C}\}$ and natural parameter $\theta = \{\theta_C, C \in \mathfrak{C}\} \in \mathbb{R}^D$. This connection is useful, since there is a rich and classical statistical literature on the properties of exponential families (e.g., [30, 19, 67]). We make use of some of these properties here. Moreover, an attractive property of the maximum likelihood estimate is its invariance to reparameterizations: namely, if we can compute the MLE $\widehat{\theta}$ in our new, exponential parameterization, then $\widehat{\psi} \equiv \exp(\widehat{\theta})$ will define the MLE in the original parameterization. We use this fact freely in the discussion to follow.

One convenient property of the exponential parameterization is that the log likelihood $\ell(\theta; X_{1\bullet}, \ldots, X_{n\bullet}) := \frac{1}{n} \sum_{i=1}^{n} \log p(X_{i\bullet}; \theta)$ can be written in terms of the empirical marginals

$$\widehat{\mu}_{C;J} = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}_{C;J}(X_{iC}), \tag{48}$$

at each clique. Note that $\widehat{\mu}_{C;J} = \widehat{p}(x_C = J)$ corresponds to the average fraction of times that $\{X_C = J\}$ in the data set. Let us define the vector $\widehat{\mu}_C = \{\widehat{\mu}_{C;J}, J \in \mathcal{X}_C\} \in m^{|C|}$, as well as the inner product $\langle \theta_C, \widehat{\mu}_C \rangle = \sum_{J \in \mathcal{X}_C} \theta_{C;J}\, \widehat{\mu}_{C;J}$. With this notation, the maximum likelihood estimate $\widehat{\theta}$ takes the form

$$\widehat{\theta} = \arg\max_{\theta \in \mathbb{R}^D} \Big\{ \sum_{C \in \mathfrak{C}} \langle \theta_C, \widehat{\mu}_C \rangle - \Phi(\theta) \Big\}. \tag{49}$$

In order to understand the structure of this optimization problem, we require some useful properties of the function Φ:

Lemma 5. *The function Φ is convex and infinitely differentiable on $\mathbb{R}^D$, and in particular, we have*

$$\frac{\partial \Phi}{\partial \theta_{C;J}}(\theta) = \underbrace{\sum_{x \in \mathcal{X}} p(x;\theta)\, \mathbb{I}_{C;J}(x_C)}_{p(x_C = J;\theta)}$$

This result shows that taking the partial derivatives of Φ with respect to the parameters over clique C yields the marginal distribution over that clique. The proof of these properties are relatively straightforward, so we leave them as an exercise for the reader.

Lemma 5 allows us to characterize the structure of the maximum likelihood estimate (49). First, it shows that the rescaled log likelihood ℓ is a concave function of θ, and hence has no local optima that can trap an iterative algorithm. Secondly, it can be used to characterize the stationary conditions that characterize a global optimum; more precisely, by taking derivatives of the rescaled log likelihood and setting them to zero, we find that the MLE $\widehat{\theta}$ satisfies the stationary conditions

$$p(x_C = J; \widehat{\theta}) = \widehat{\mu}_{C;J} \quad \text{for all } C \in \mathfrak{C}, \text{ and } J \in \mathcal{X}_C. \tag{50}$$

In words, the maximum likelihood estimate $\widehat{\theta}$ is such that the marginals computed under the distribution $p(\cdot; \widehat{\theta})$ are equal to the empirical marginals determined by the data. This type of moment-matching condition characterizes the MLE in any exponential family; see the monograph [67] for further details.

Let us now return to our previously open question: for a tree-structured graph $\mathcal{T}$ with the factorization (42), why does equation (44) specify the maximum likelihood estimate? We simply need to verify that the moment-matching conditions are satisfied. Given the suggested form of $\widehat{\psi}$ from equation (44), we have

$$p(x_1,\dots,x_N;\widehat{\psi}) = \frac{1}{Z}\prod_{s\in\mathcal{V}}\widehat{p}(x_s)\prod_{(s,t)\in\mathcal{E}}\frac{\widehat{p}(x_s,x_t)}{\widehat{p}(x_s)\widehat{p}(x_t)}. \tag{51}$$

Our claim is that in this factorization, we must necessarily have $Z = 1$ and moreover

$$p(x_s;\widehat{\psi}) = \widehat{p}(x_s) \quad \text{for all } s\in\mathcal{V}, \text{ and} \quad p(x_s,x_t;\widehat{\psi}) = \widehat{p}(x_s,x_t) \quad \text{for all } (s,t)\in\mathcal{E}. \tag{52}$$

In order to verify this claim, we first observe that any undirected tree can be converted to an equivalent directed form as follows: first, designate one vertex, say r, as the root, and then orient all edges away from the root toward the leaves. See Figure 9 for an illustration of this procedure.

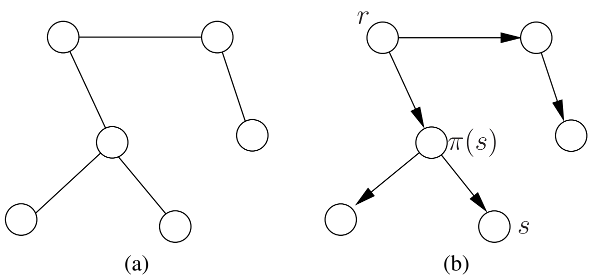

Fig. 9. (a) Undirected tree. (b) Conversion of undirected tree to a directed tree rooted at vertex r. Each vertex $s\in\mathcal{V}\backslash\{r\}$ now has a unique parent $\pi(s)$. Any given parent may have multiple children.

After this conversion, each vertex $s\in\mathcal{V}\backslash\{r\}$ has a unique parent $\pi(s)$, and we can associate the product $\widehat{\psi}_s(x_s)\,\widehat{\psi}_{s\pi(s)}(x_s,x_{\pi(s)})$ with the directed edge $(\pi(s)\to s)$ and the term $\widehat{\psi}_r(x_r)$ with the root vertex. In this way, the factorization (51) can be written the directed form

$$p(x_1,\dots,x_N;\widehat{\psi}) = \frac{1}{Z}\widehat{\psi}_r(x_r)\prod_{s\in\mathcal{V}\backslash\{r\}}\widehat{\psi}_s(x_s)\,\widehat{\psi}_{s\pi(s)}(x_s,x_{\pi(s)}) \;=\; \frac{1}{Z}\underbrace{\widehat{p}(x_r)}_{f_r(x_r)}\prod_{s\in\mathcal{V}\backslash\{r\}}\underbrace{\frac{\widehat{p}(x_s,x_{\pi(s)})}{\widehat{p}(x_{\pi(s)})}}_{f_s(x_s,x_{\pi(s)})}.$$

Note that this is now a factorization of the general directed form (2), with functions $\{f_s, s\in\mathcal{V}\}$ that satisfy the normalization condition (1). Consequently, an application of Proposition 1 implies that

$$p(x_r;\widehat{\psi}) = \widehat{p}(x_r), \quad \text{and} \quad \frac{p(x_s,x_{\pi(s)};\widehat{\psi})}{p(x_{\pi(s)};\widehat{\psi})} = \frac{\widehat{p}(x_s,x_{\pi(s)})}{\widehat{p}(x_{\pi(s)})} \quad \text{for all } \pi(s)\to s.$$

A recursive argument, moving from the root r out to the leaves, then shows that the marginal-matching condition (52) holds. We leave the details of this final step as an exercise for the reader. (Start with the case of a Markov chain to see the general idea.)

5.2.3 Iterative proportional scaling

Thus far, we have seen that in the special case of a tree-structured graph, it is straightforward to find a set of compatibility functions $\widehat{\psi}$ (or equivalently, a vector of parameters $\widehat{\theta}$) that satisfy the moment-matching conditions (50). The same basic idea can be extended to any triangulated graph, by first building a junction tree, and then following the analogous procedure on the junction tree.

In contrast, for a general undirected graph (with cycles, not triangulated), the MLE problem no longer has a closed-form solution, and needs to be solved by an iterative algorithm. Since the moment-matching conditions (50) characterize the global optimum of the convex program (49) that defines the MLE, a variety of standard algorithms from convex programming [9, 18] could be applied. Here we describe one particular algorithm, attractive due to the simplicity of its updates, known as *iterative proportional scaling*, or IPS for short. As we will see, it amounts to a block co-ordinate ascent of the objective function. The IPS algorithm takes as input the empirical marginals $\{\widehat{\mu}_C, C \in \mathfrak{C}\}$, computed from the data set as in equation (48), as well as a particular ordering of clique set.

[**Iterative proportional scaling (IPS)**]

1. Initialize estimate $\theta'_C = 0$ for all cliques $C \in \mathfrak{C}$.
2. Cycling over cliques in the given order:
   a. Activate clique A, and given current parameter estimate θ', compute marginal probabilities

$$\mu'_{A;J} = p(x_A = J; \theta') \quad \text{for all } J \in \mathcal{X}_A, \tag{53}$$

   and form the vector $\mu'_A = \{\mu'_{A;J}, J \in \mathcal{X}_A\}$.

   b. Update the parameter estimate $\theta' \mapsto \theta$ via

$$\theta_C = \begin{cases} \theta'_C & \text{for all } C \in \mathfrak{C} \backslash \{A\} \\ \theta'_A + \log \frac{\widehat{\mu}_A}{\mu'_A} & \text{for } C = A. \end{cases}$$

3. Iterate until convergence.

Observe that Step 2(a) of the algorithm requires computation of the marginal distribution $p(x_A; \theta')$ over the currently active clique A. Doing so requires running the elimination algorithm, or building a junction tree on which to run a message-passing algorithm. This step can be computationally expensive if the graph treewidth is high, in which context approximate methods for marginalization become important (see the monograph [67] for more details). The update in Step 2(b) fixes the parameter estimate on all cliques *except* for the active one A; for this clique, the update can be equivalently written in the form

$$e^{\theta_{A;J}} = e^{\theta'_{A;J}} \frac{\widehat{\mu}_{A;J}}{\mu'_{A;J}} \quad \text{for all } J \in \mathcal{X}_A.$$

This form of the update clarifies the origins of the name "proportional scaling", since the parameters are being adjusted by the relative proportions of the empirical marginal $\widehat{\mu}_{A;J}$ and the current model marginal $\mu'_{A;J}$.

It is clear from the IPS updates that any fixed point must satisfy the moment-matching conditions (50), and hence be a global optimum of the log likelihood. Less clear, however, is whether convergence is guaranteed (though it is assumed in the statement of Step 3). Among other properties, the following result shows that the IPS algorithm is indeed guaranteed to converge to the MLE:

Proposition 5 (IPS properties). *The IPS algorithm has the following properties:*

(a) We have $\Phi(\theta') = \Phi(\theta)$ over all updates $\theta' \mapsto \theta$ of the algorithm.
(b) After updating clique A, the estimate θ satisfies

$$p(x_A = J; \theta) = \widehat{\mu}_{A;J} \quad \textit{for all } J \in \mathcal{X}_A. \tag{54}$$

(c) The algorithm iterates converge to an MLE solution $\widehat{\theta}$.

Proof. (a), (b) By definition of the update, for each $J \in \mathcal{X}_A$, we have

$$\begin{aligned}
p(x_A = J; \theta) &= \sum_{x \in \mathcal{X}_{[N]}} p(x; \theta) \mathbb{I}_{A;J}(x_A) \\
&= \sum_{x \in \mathcal{X}_{[N]}} p(x; \theta') \frac{e^{\Phi(\theta') - \Phi(\theta)} \widehat{\mu}_{A;J}}{\mu'_{A;J}} \mathbb{I}_{A;J}(x_A) \\
&= \Big\{ \frac{\sum_{x \in \mathcal{X}_{[N]}} p(x; \theta') \mathbb{I}_{A;J}(x_A)}{\mu'_{A;J}} \Big\} \widehat{\mu}_{A;J} e^{\Phi(\theta') - \Phi(\theta)} \\
&= \widehat{\mu}_{A;J} e^{\Phi(\theta') - \Phi(\theta)}.
\end{aligned}$$

Summing both sides over $J \in \mathcal{X}_A$ yields that $\Phi(\theta') = \Phi(\theta)$ as claimed in part (a), and we then see that $p(x_A = J; \theta) = \widehat{\mu}_{A;J}$ as claimed in part (b).

(c) Note that the rescaled log likelihood can be block partitioned in the form $\ell(\theta'_C, C \in \mathfrak{C})$. With θ' fixed in all co-ordinates except those indexed by A, let us consider the function

$$\theta_A \mapsto \ell(\theta_A, \theta'_C, C \in \mathfrak{C} \backslash A).$$

It is a concave function of θ_A. Moreover, by combining Lemma 5 with the result of part (b), we see that, after updating clique A, the IPS algorithm has maximized this function over θ_A. Thus, the IPS algorithm is equivalent to performing block co-ordinate ascent of the log-likelihood function. By known results on the convergence of such procedures [9], it follows that IPS converges to a global optimum. □

5.3 Tree selection and the Chow-Liu algorithm

In our discussion thus far, we have assumed that the graph structure is known, and that only the compatibility functions need to be estimated. In this section, we consider an instance of the more general problem in which neither is known. As previously mentioned, this problem is known as model selection for graphical models, or graph structure learning. We limit ourselves to the problem of selecting tree-structured graphical models. See the bibliographic section for discussion of other methods suitable for more general forms of graph structure learning.

Given our data set $\{X_{1\bullet}, \ldots, X_{n\bullet}\}$, suppose that our goal is to choose the "best-fitting" spanning tree graphical model. More formally, let $\mathbb{T}$ be the set of all spanning trees on N vertices. For a given tree $\mathcal{T} \in \mathbb{T}$ and a given distribution $p(x;\theta(\mathcal{T}))$ that is Markov with respect to $\mathcal{T}$, we measure its fit in terms of its rescaled log likelihood $\ell(\theta(\mathcal{T})) = \frac{1}{n}\sum_{i=1}^{n} \log p(X_{i\bullet};\theta(\mathcal{T}))$.

In principle, the solution to this problem is straightforward: for each tree $\mathcal{T} \in \mathbb{T}$, we compute the maximum likelihood estimate

$$\widehat{\theta}(\mathcal{T}) = \arg\max_{\theta(\mathcal{T})} \ell(\theta(\mathcal{T})),$$

thereby obtaining a list $\{\ell(\widehat{\theta}(\mathcal{T}), \mathcal{T} \in \mathbb{T}\}$ of maximized likelihoods for every tree $\mathcal{T}$ in the set $\mathbb{T}$ of all spanning trees. We then choose the tree $\widehat{\mathcal{T}}$ with the highest maximized likelihood, that is $\widehat{\mathcal{T}} = \arg\max_{\mathcal{T}\in\mathbb{T}} \ell(\widehat{\theta}(\mathcal{T}))$. However, a graph with N vertices has an enormous number of possible spanning trees—N^{N-2} to be precise [20]—so that a brute force approach is infeasible. Fortunately, as previously shown (44), the MLE for a tree-structured graph has a very special structure, which can be exploited to obtain an efficient algorithm for finding the best-fitting tree.

[**Chow-Liu algorithm for maximum likelihood tree**]

1. Compute the empirical marginals (43) associated with all vertices $s \in \mathcal{V}$ and all distinct vertex pairs (s,t).
2. For each distinct pair of vertices (s,t), compute the empirical mutual information

$$\widehat{I}_{st} = \sum_{(x_s,x_t)\in\mathcal{X}_s\times\mathcal{X}_t} \widehat{p}(x_s,x_t)\log\frac{\widehat{p}(x_s,x_t)}{\widehat{p}(x_s)\,\widehat{p}(x_t)}. \tag{55}$$

3. Using $\{\widehat{I}_{st}\}$ as edge weights, compute a maximum weight spanning tree

$$\widehat{\mathcal{T}} = \arg\max_{\mathcal{T}\in\mathbb{T}} \sum_{(s,t)\in\mathcal{E}(\mathcal{T})} \widehat{I}_{st}. \tag{56}$$

4. Return the maximum likelihood estimate $\widehat{\theta}(\mathcal{T})$ on the tree from the previous step:

$$\widehat{\theta}_s(x_s;\widehat{\mathcal{T}}) = \log\widehat{p}_s(x_s) \quad \text{for all } s\in\mathcal{V}, \text{ and}$$
$$\widehat{\theta}_{st}(x_s,x_t;\widehat{\mathcal{T}}) = \log\frac{\widehat{p}(x_s,x_t)}{\widehat{p}(x_s)\widehat{p}(x_t)} \quad \text{for all } (s,t)\in\mathcal{E}(\widehat{\mathcal{T}}).$$

Note that Step 3 requires solving a maximum weight spanning tree problem. As mentioned previously during our treatment of the junction tree algorithm, there are various efficient methods for this problem, including Kruskal's algorithm and Prim's algorithm. See the bibliographic section for further discussion.

We claim that the Chow-Liu algorithm is guaranteed to return a maximum likelihood tree—namely, one satisfying

$$\widehat{\mathcal{T}} \in \arg\max_{\mathcal{T}\in\mathbb{T}} \ell(\widehat{\theta}(\mathcal{T})) \;=\; \arg\max_{\mathcal{T}\in\mathbb{T}} \Big\{\max_{\theta(\mathcal{T})} \ell(\theta(\mathcal{T}))\Big\}.$$

The proof of this claim is straightforward given our previous discussion. For any fixed tree $\mathcal{T}$, the maximum likelihood solution has the form (44), whence

$$\begin{aligned}
\ell(\widehat{\theta}(\mathcal{T})) &= \frac{1}{n}\sum_{i=1}^{n} \log p(X_{i\bullet};\widehat{\theta}(\mathcal{T})) \\
&= \frac{1}{n}\sum_{i=1}^{n}\Big\{\sum_{s\in\mathcal{V}}\log\widehat{p}(X_{is}) + \sum_{(s,t)\in\mathcal{E}(\mathcal{T})}\log\frac{\widehat{p}(X_{is},X_{it})}{\widehat{p}(X_{is})\,\widehat{p}(X_{it})}\Big\} \\
&= \sum_{s\in\mathcal{V}}\widehat{p}(x_s)\log\widehat{p}(x_s) + \sum_{(s,t)\in\mathcal{E}(\mathcal{T})}\widehat{p}(x_s,x_t)\log\frac{\widehat{p}(x_s,x_t)}{\widehat{p}(x_s)\,\widehat{p}(x_t)} \\
&= C + \sum_{(s,t)\in\mathcal{E}(\mathcal{T})}\widehat{I}_{st},
\end{aligned}$$

where C denotes a constant that is independent of the tree $\mathcal{T}$. Consequently, the maximum weight spanning tree obtained in Step 3 corresponds to the maximum likelihood tree.

Intuitively, one might think that an analogous procedure could be extended to junction trees, requiring solving a maximum weight spanning hypertree problem as opposed to a maximum weight spanning tree problem. Unfortunately, unlike the case of spanning trees, computing a maximum weight hypertree is computationally expensive, so that in practice, exact methods are limited to ordinary spanning trees, as in the Chow-Liu algorithm.

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

6 Bibliographic details and remarks

Graphical models were developed independently in a number of different communities. The notion of a Gibbs distribution appeared initially in the statistical physics literature, a notable example being the Ising model [40] that was introduced to model ferromagnetism. Other communities in which graphical models were developed include information theory and coding theory [34, 61], contingency tables and discrete multivariate analysis [38, 13], image processing [35, 12], and artificial intelligence [52].

The Hammersley-Clifford theorem derives its name from a manuscript that was never published [39]. Besag [10] and Grimmett [37] were the first to publish proofs of the result; in these notes, we have adapted the latter proof. See Clifford [22] for some discussion of the history of the result. Lauritzen [45] provides discussion of how the Markov-factorization equivalence can break down when the strict positivity condition fails to hold.

Forms of the sum-product and max-product algorithms were independently developed in many different communities, including the work of Kalman in control theory [42], work on LDPC codes [34] and Viterbi decoding [64, 32] in error-control coding, the alpha-beta or forward-backward algorithms in signal processing [53], Pearl's work [52] in artificial intelligence, and work on non-serial dynamic programming [8]. Dating back to the 1960s, a number of authors (e.g., [51, 56]) studied the relation between graph triangulation and the vertex elimination algorithm, and pointed out the connection to the complexity of Gaussian elimination algorithms for solving linear systems of equations. (In the statistical setting, this problem can be cast as an inference problem for a multivariate Gaussian.) At the end of the 1980s, several groups of researchers independently recognized that the significance of tree decompositions and triangulation extended beyond solving linear equations to more general classes of graph-theoretic problems [46, 15]. Within the statistics and artificial intelligence communities, the junction tree framework was developed by Lauritzen and Spielgelhalter [46]. The paper [14] contains an extensive description of treewidth and related notions of graph complexity. There are various algorithms for solving the maximum weight spanning tree problem, including Kruskal's algorithm and Prim's algorithm [23].

Early work on iterative proportional fitting and scaling algorithms include the papers [60, 31, 27]. These algorithms are actually special cases of more general Bregman-based projection algorithms [29, 26, 24, 25]; see also the work of Amari [2, 3] on the information geometry of exponential families. The Chow-Liu algorithm for finding the maximum likelihood tree appeared in the paper [21]. Karger and Srebro [43, 59] studied generalizations of this problem to junction trees of higher treewidth, establishing the hardness of the problem (in a complexity-theoretic sense) but also proposing an approximation algorithm. There is a very large literature on methods for graphical model selection, local forms of ℓ_1-regularized regression (e.g., [50, 54]), ℓ_1-regularization and global likelihoods (e.g., [33, 7, 55, 49]), as well as multiple testing and related approaches (e.g., [58, 41, 5]).

Appendix

A: Triangulation and equivalent graph-theoretic properties

In this Appendix, we prove Theorem 3 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 9. 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 $A \cup B \cup S$ 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 10. 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 4. *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 prove the sequence of implications $(T) \Rightarrow (D) \Rightarrow (R) \Rightarrow (J) \Rightarrow (T)$.

(T) ⇒ (D): We proceed via induction on the graph size N. The claim is trivial for $N = 1$, so let us assume it for all graphs with N vertices, and prove that it also holds for any graph $\mathcal{G}$ with $N+1$ vertices. If $\mathcal{G}$ is complete, then it is certainly decomposable. Moreover, if $\mathcal{G}$ has more than one connected component, each of which is complete, then it is also decomposable. 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 vertices 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 vertices 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 we need to show that S is complete. If $|S| = 1$, the claim is trivial. Otherwise, for any two distinct vertices $s, t \in \mathcal{S}$, 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 claim relies on the minimality of S: if there did not exist a path from a to s, then vertex s could be removed from $\mathcal{S}$. Similar reasoning applies to establish a path from a to t, and also the paths involving B.)

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 vertices 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 must be triangulated, since otherwise $\mathcal{G}$ would not be triangulated, and they have cardinality strictly smaller than $N+1$, so the result follows by induction.

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

Lemma 6. *Every decomposable graph with at least two vertices has at least two simplicial vertices. If the graph is not complete, these vertices can be chosen to be non-adjacent.*

Proof. Proof by induction on graph size N. Trivial for $N = 2$. Consider a decomposable graph with $N+1$ vertices. If the graph is complete, all vertices 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 vertices 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 vertices 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 vertices 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 N. Trivial for $N = 1$. Let s be a simplicial vertex, and consider subgraph $\mathcal{G}'$ obtained by removing s. By induction, $\mathcal{G}'$ has a junction tree $\mathcal{T}'$, which we will extend to a junction tree $\mathcal{T}$ for $\mathcal{G}$. Let C' be a maximal clique in $\mathcal{T}'$ 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 $\mathcal{T}$, 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 $\mathcal{T}'$, with an edge to C'. Since s is in no other clique of $\mathcal{T}$ and $C \backslash \{s\}$ is a subset of C', the tree $\mathcal{T}'$ is a junction tree for $\mathcal{G}$¿

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

Step 1: If $C \subseteq C'$, then $\mathcal{T}'$ 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 $\mathcal{T}'$ (using junction tree property). (I.e., say $R \cap D \neq 0$ for some clique node D in $\mathcal{T}'$. Then there exists $s \in C \cap D$, but $s \notin C'$, with violates running intersection.) Follows that $\mathcal{T}'$ 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 Transactions on Information Theory **46**, 325–343 (2000)
2. Amari, S.: Differential geometry of curved exponential families — curvatures and information loss. Annals of Statistics **10**(2), 357–385 (1982)
3. Amari, S.: Differential-geometrical methods in statistics. Springer-Verlag, New York (1985)
4. Amestoy, P., Davis, T.A., Duff, I.S.: An approximate minimum degree ordering algorithm. SIAM Journal on Matrix Analysis and Applications **17**, 886–905 (1996)
5. Anandkumar, A., Tan, V.Y.F., Huang, F., Willsky, A.S.: High-dimensional structure learning of Ising models: Local separation criterion. Annals of Staitstics **40**(3), 1346–1375 (2012)
6. Arnborg, S.: Complexity of finding embeddings in a k-tree. SIAM Jour. Alg. Disc. Math **3**(2), 277–284 (1987)
7. Banerjee, O., Ghaoui, L.E., d'Aspremont, A.: Model selection through sparse maximum likelihood estimation for multivariate Gaussian or binary data. Jour. Mach. Lear. Res. **9**, 485–516 (2008)
8. Bertele, U., Brioschi, F.: Nonserial dynamic programming. Academic Press, New York (1972)
9. Bertsekas, D.: Nonlinear programming. Athena Scientific, Belmont, MA (1995)
10. Besag, J.: Spatial interaction and the statistical analysis of lattice systems. J. Roy. Stat. Soc. Series B **36**, 192–236 (1974)
11. Besag, J.: Efficiency of pseudolikelihood estimation for simple Gaussian fields. Biometrika **64**(3), 616–618 (1977)
12. Besag, J.: On the statistical analysis of dirty pictures. Journal of the Royal Statistical Society, Series B **48**(3), 259–279 (1986)
13. Bishop, Y.M., Fienberg, S.E., Holland, P.W.: Discrete multivariate analysis: Theory and practice. MIT Press, Boston, MA (1975)
14. Bodlaender, H.: A tourist guide through treewidth. Acta Cybernetica **11**, 1–21 (1993)
15. Bodlaender, H.L.: Dynamic programming on graphs with bounded treewidth. In: Automata, Languages and Programming, vol. 317, pp. 105–118. Springer-Verlag (1988)
16. Bodlaender, H.L.: A linear-time algorithm for finding tree decompositions of small treewidth. SIAM Journal of Computing **25**, 1305–1317 (1996)
17. Bollobás, B.: Graph theory: an introductory course. Springer-Verlag, New York (1979)
18. Boyd, S., Vandenberghe, L.: Convex optimization. Cambridge University Press, Cambridge, UK (2004)
19. Brown, L.D.: Fundamentals of statistical exponential families. Institute of Mathematical Statistics, Hayward, CA (1986)
20. Cayley, A.: A theorem on trees. Quart. J. Math **23**, 376–378 (1889)
21. Chow, C.K., Liu, C.N.: Approximating discrete probability distributions with dependence trees. IEEE Trans. Info. Theory **IT-14**, 462–467 (1968)

22. Clifford, P.: Markov random fields in statistics. In: G. Grimmett, D.J.A. Welsh (eds.) Disorder in physical systems. Oxford Science Publications (1990)
23. Cormen, T.H., Leiserson, C.E., Rivest, R.L.: Introduction to Algorithms. MIT Press, Cambridge, MA (1990)
24. Csiszár, I.: Sanov property, generalized I-projection and a conditional limit theorem. Annals of Probability **12**(3), 768–793 (1984)
25. Csiszár, I.: A geometric interpretation of Darroch and Ratcliff's generalized iterative scaling. Annals of Statistics **17**(3), 1409–1413 (1989)
26. Csiszar, I.: I-divergence geometry of probability distributions and minimization problems. Annals of Probability, **3**(1), 146–158 (1975)
27. Darroch, J.N., Ratcliff, D.: Generalized iterative scaling for log-linear models. Annals of Mathematical Statistics **43**, 1470–1480 (1972)
28. Dawid, A.P.: Applications of a general propagation algorithm for probabilistic expert systems. Statistics and Computing **2**, 25–36 (1992)
29. Dykstra, R.L.: An iterative procedure for obtaining I-projections onto the intersection of convex sets. Annals of Probability **13**(3), 975–984 (1985)
30. Efron, B.: The geometry of exponential families. Annals of Statistics **6**, 362–376 (1978)
31. Fienberg, S.: An iterative procedure for estimation in contingency tables. Annals of Mathematical Statistics **41**(3), 907–917 (1970)
32. Forney Jr., G.D.: The Viterbi algorithm. Proc. IEEE **61**, 268–277 (1973)
33. Friedman, J., Hastie, T., Tibshirani, R.: Sparse inverse covariance estimation with the graphical lasso. Biostatistics **9**, 432–441 (2008)
34. Gallager, R.G.: Low-density parity check codes. MIT Press, Cambridge, MA (1963)
35. Geman, S., Geman, D.: Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. IEEE Trans. PAMI **6**, 721–741 (1984)
36. George, A., Liu, J.W.H.: The evolution of the minimum degree ordering algorithm. SIAM Review **31**(1), 1–19 (1989)
37. Grimmett, G.R.: A theorem about random fields. Bulletin of the London Mathematical Society **5**, 81–84 (1973)
38. Haberman, S.J.: The analysis of frequency data. University of Chicago Press, Chicago, IL (1974)
39. Hammersley, J.M., Clifford, P.: Markov fields on finite graphs and lattices (1971). Unpublished
40. Ising, E.: Beitrag zur theorie der ferromagnetismus. Zeitschrift für Physik **31**(1), 253–258 (1925)
41. Kalisch, M., Bühlmann, P.: Estimating high-dimensional directed acyclic graphs with the PC algorithm. Journal of Machine Learning Research **8**, 613–636 (2007)
42. Kalman, R.: A new approach to linear filtering and prediction problems. The American Society of Mechanical Engineers: Basic Engineering, series D **82**, 35–45 (1960)
43. Karger, D., Srebro, N.: Learning Markov networks: maximum bounded tree-width graphs. In: Symposium on Discrete Algorithms, pp. 392–401 (2001)
44. Kschischang, F., Frey, B., Loeliger, H.A.: Factor graphs and the sum-product algorithm. IEEE Trans. Info. Theory **47**(2), 498–519 (2001)
45. Lauritzen, S.L.: Graphical Models. Oxford University Press, Oxford (1996)
46. Lauritzen, S.L., Spiegelhalter, D.J.: Local computations with probabilities on graphical structures and their application to expert systems (with discussion). Journal of the Royal Statistical Society B **50**, 155–224 (1988)
47. van Lint, J.H., Wilson, R.M.: A course in combinatorics. Cambridge University Press, Cambridge (1992)
48. Loeliger, H.A.: An introduction to factor graphs. IEEE Signal Processing Magazine **21**, 28–41 (2004)
49. Meinshausen, N.: A note on the Lasso for graphical Gaussian model selection. Statistics and Probability Letters **78**(7), 880–884 (2008)
50. Meinshausen, N., Bühlmann, P.: High-dimensional graphs and variable selection with the Lasso. Annals of Statistics **34**, 1436–1462 (2006)

51. Parter, S.V.: The use of linear graphs in Gaussian elimination. SIAM Review **3**, 119–130 (1961)
52. Pearl, J.: Probabilistic reasoning in intelligent systems. Morgan Kaufman, San Mateo (1988)
53. Rabiner, L.R.: A tutorial on hidden Markov models and selected applications in speech recognition. Proceedings of the IEEE **77**(2), 257–285 (1989)
54. Ravikumar, P., Wainwright, M.J., Lafferty, J.: High-dimensional Ising model selection using ℓ_1-regularized logistic regression. Annals of Statistics **38**(3), 1287–1319 (2010)
55. Ravikumar, P., Wainwright, M.J., Raskutti, G., Yu, B.: High-dimensional covariance estimation by minimizing ℓ_1-penalized log-determinant divergence. Electronic Journal of Statistics **5**, 935–980 (2011)
56. Rose, D.J.: Triangulated graphs and the elimination process. Journal of Mathematical Analysis and Applications **32**, 597–609 (1970)
57. Shafer, G.R., Shenoy, P.P.: Probability propagation. Annals of Mathematics and Artificial Intelligence **2**, 327–352 (1990)
58. Spirtes, P., Glymour, C., Scheines, R.: Causation, prediction and search. MIT Press (2000)
59. Srebro, N.: Maximum likelihood Markov networks: an algorithmic approach. Master's thesis, MIT (2000)
60. Stephan, F.F.: Iterative method fo adjusting sample frequency tables when expected margins are known. Annals of Mathematical Statistics **13**, 166–178 (1942)
61. Tanner, R.M.: A recursive approach to low complexity codes. IEEE Trans. Info. Theory **IT-27**, 533–547 (1980)
62. 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)
63. Verdú, S., Poor, H.V.: Abstract dynamic programming models under commutativity conditions. SIAM J. Control and Optimization **25**(4), 990–1006 (1987)
64. Viterbi, A.: Error bounds for convolutional codes and an asymptotically optimal decoding algorithm. IEEE Trans. Info. Theory **IT-13**, 260–269 (1967)
65. Wainwright, M.J., Jaakkola, T.S., Willsky, A.S.: Tree-based reparameterization framework for analysis of sum-product and related algorithms. IEEE Trans. Info. Theory **49**(5), 1120–1146 (2003)
66. Wainwright, M.J., Jaakkola, T.S., Willsky, A.S.: Tree consistency and bounds on the max-product algorithm and its generalizations. Statistics and Computing **14**, 143–166 (2004)
67. 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)
68. Yannakakis, M.: Computing the minimum fill-in is NP-complete. SIAM Journal on Algebraic and Discrete Methods **2**(1), 77–79 (1981)