

Graphical Models, Exponential Families, and Variational Inference: Chapters 1 & 2

Martin J. Wainwright and Michael I. Jordan (2008)

Lecture Notes

Sweta Karlekar

2024

Wainwright–Jordan: Chapters 1 & 2

Objective

Given a distribution represented as a graphical model, we are concerned with computing *marginal probabilities and modes*. We know this as the task of “probabilistic inference.”

Overview

- Factorizing graphical models
 - Directed
 - Undirected
 - Factor graphs
- Inference objectives
- Exact inference algorithms
 - Sum-product algorithm
 - Junction-tree algorithm
- Intuition for approximate inference
- Examples

Goal of this section:

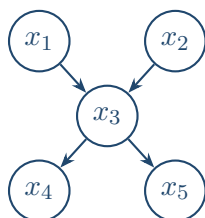
- a) Build background for graphical models.
- b) Provide an explanation of exact inference methods.
- c) Motivate why we want approximate inference methods.

Factorizing graphs

Objective

Represent each graphical model as a *product of factors*, with each factor associated with some small subgraph.

Directed graphs



In a directed graph, we know the *parents* of each node. The joint distribution is the product of the local conditional distributions:

$$p(x_1, \dots, x_m) = \prod_{s \in V} p_s(x_s \mid x_{\pi(s)}).$$

V : vertex set of the graph. $\pi(s)$: parents of s .
 $p_s(x_s \mid x_{\pi(s)})$ is a nonnegative conditional probability mass function, normalized over x_s :

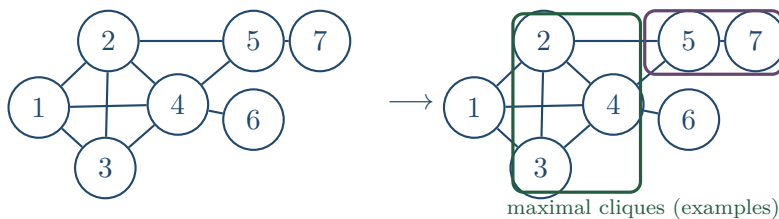
$$\sum_{x_s} p_s(x_s \mid x_{\pi(s)}) = 1 \quad \text{for each choice of parents } x_{\pi(s)}.$$

Undirected graphs (Markov random fields; MRFs)

For an undirected graph we do not have “parents,” so instead we use *cliques*.

What is a clique?

A clique is a fully connected (complete) subgraph, potentially as small as one vertex.



Analogously, each clique C has an associated compatibility / potential function $\psi_C(x_C)$, defined over the variables in that clique.

Undirected factorization

$$p(x_1, \dots, x_m) = \frac{1}{Z} \prod_{C \in \mathcal{C}} \psi_C(x_C)$$

$\mathcal{C}$: cliques used in the factorization; Z : normalizing factor.

The potentials ψ_C need not have an obvious or direct relationship to marginal or conditional distributions over the graph.

Unlike directed factorization, where factors correspond to conditional probabilities for child–parent tuples.

Factor graphs

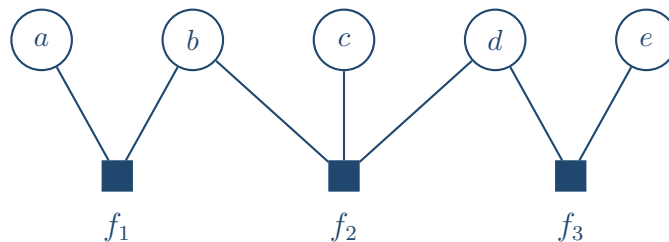
An alternative graphical representation that emphasizes the *factorization* of the distribution.

Definition

Bipartite graph with two kinds of nodes—*variables* and *factors*—where an edge between a variable and factor exists iff that variable occurs in the factor.

For example, let

$$f(a, b, c, d, e) = f_1(a, b) f_2(b, c, d) f_3(d, e).$$



We'll return to factor graphs when performing inference!

Inference

Start with the basics:

$$p(A | B) = \frac{p(B | A) p(A)}{p(B)}$$

Inference focuses on solving one or more of the following:

- a) **Likelihood of the observed data** $\Rightarrow p(B | A)$.

Given observed data B and a hypothesis or model parameters A , how likely is it to observe the data under those parameters?

- b) **Marginal distribution** of some subset of nodes $\Rightarrow p(B)$.

Integrating / summing out variables in the evidence part of Bayes' rule.

- c) **Conditional distributions or posteriors** $\Rightarrow p(A | B)$.

Given some observed data, find the posterior distribution of the variables in A .

- d) **Mode of the density** $\Rightarrow$ MAP estimate.

Find the most probable state, or the mode of the posterior distribution.

We tend to focus most effort on computing marginals; computing modes requires only a slight change.

Exact inference

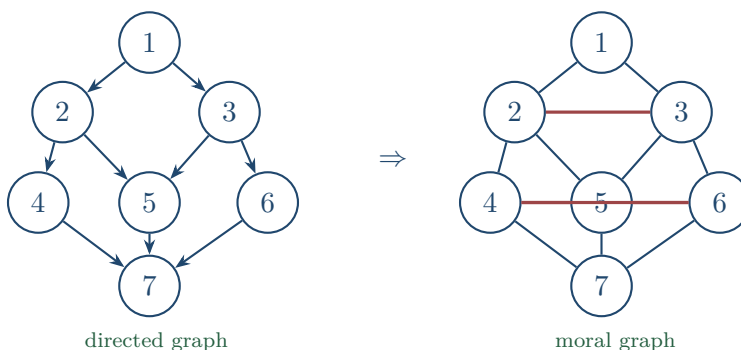
Naive approach

Compute the marginal at a single node by summing the joint over every possible value of the remaining variables.

For inference, it is convenient to convert directed graphs to undirected graphs, since both have factorized expressions for the joint probability. Then we can use the undirected representation for inference.

Directed $\rightarrow$ undirected: moralization

- Connect all parents of each child directly.
- Convert all directed edges into undirected edges.



Why do we do this? The factors in a directed graph involve child–parent tuples; the parents are not necessarily connected. In a moral graph, the factors are defined on cliques.

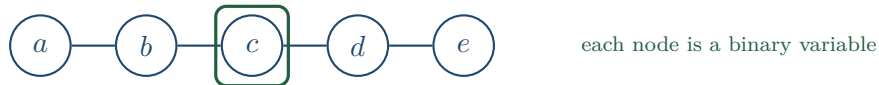
An undirected factorization is the more general representation here.

The exact-inference algorithms in these notes assume the needed transformation to an undirected graph has already been performed.

Back to inference: why sum-product?

The naive approach becomes exponentially expensive as the number of nodes grows.

For five binary variables arranged in a chain:



$$p(c) = \sum_{a,b,d,e} p(a,b,c,d,e), \quad \underbrace{2^4 = 16}_{\text{possible assignments}}.$$

Each of a, b, d, e can take 2 values; the computation grows exponentially.

How can we make this better?

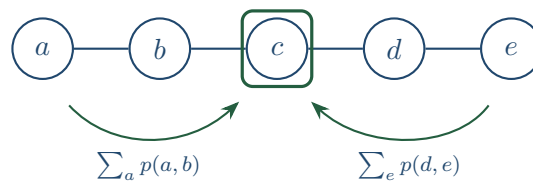
Dynamic programming

Break the computation into parts, and store intermediate results so they do not need to be recomputed.

Sum-product algorithm

Intuition

Input a Markov random field; start at the leaves and recursively compute the marginals, passing partial results closer to the root and normalizing at the end.

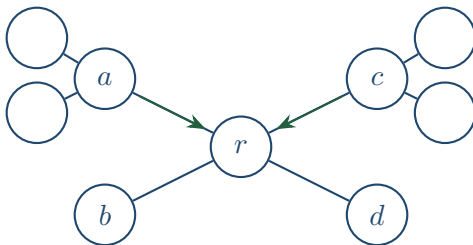


Start at the leaves and work toward the root. Instead of enumerating every assignment simultaneously, process them in chunks.

Why does message passing work on trees?

Sum-product assumes information can move through the graph in a way that each node receives information from its neighbors exactly once.

In a tree, any node is reachable from another node through *one unique path*.

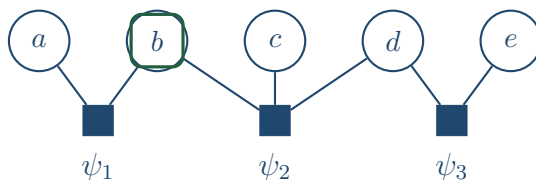


build the calculation from subtrees toward the root

Intermediate results are called *messages*; hence the name *message-passing algorithm*. **Let's walk through an example.**

Suppose

$$p(a, b, c, d, e) \propto \psi_1(a, b) \psi_2(b, c, d) \psi_3(d, e).$$



We want to marginalize out all variables except b :

$$\begin{aligned} p(b) &= \sum_{a, c, d, e} p(a, b, c, d, e) \\ &\propto \sum_a \sum_c \sum_d \sum_e \psi_1(a, b) \psi_2(b, c, d) \psi_3(d, e) \\ &= \underbrace{\left[\sum_a \psi_1(a, b) \right]}_{\text{left subtree}} \underbrace{\left[\sum_{c, d} \psi_2(b, c, d) \underbrace{\sum_e \psi_3(d, e)}_{\text{start here}} \right]}_{\text{right subtree}}. \end{aligned}$$

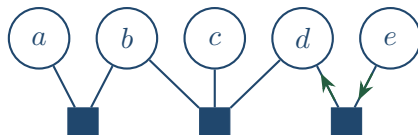
Use the factorized joint distribution and break the sums into smaller computations.

Computing the messages

Messages are always functions of the recipient.

First, pass information from e to d :

$$N_{\psi_3 \rightarrow d}(d) = \sum_e \psi_3(d, e).$$

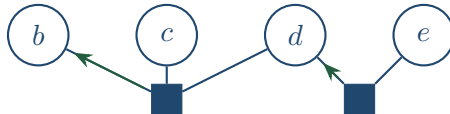


information from the d, e end of the graph

Now include $\psi_2(b, c, d)$ and the message received from ψ_3 :

$$N_{\psi_2 \rightarrow b}(b) = \sum_{c,d} \psi_2(b, c, d) N_{\psi_3 \rightarrow d}(d).$$

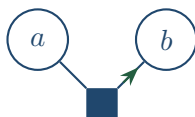
After summing out c and d , the message is a function of b .



combine the right-subtree information and pass it to b

The other end supplies a message to b :

$$N_{\psi_1 \rightarrow b}(b) = \sum_a \psi_1(a, b).$$



left-subtree message to b

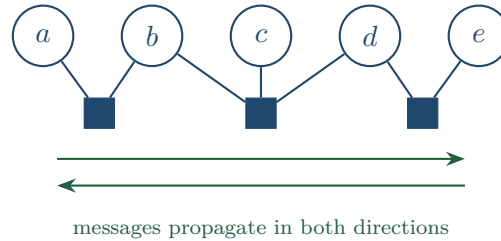
Multiply the incoming messages and normalize:

$$p(b) = \frac{1}{Z_b} N_{\psi_1 \rightarrow b}(b) N_{\psi_2 \rightarrow b}(b).$$

A backward pass can calculate marginals for the other nodes too—this is where the computational benefit really appears.

Sum-product: general rules

To find the marginals of all nodes, messages must also travel in the other direction.



At variable nodes

Take the *product* of all incoming messages, excluding the message arriving from the edge to which we are sending a message.

$$N_{t \rightarrow s}(x_t) = \prod_{u \in \mathcal{N}(t) \setminus \{s\}} N_{u \rightarrow t}(x_t), \quad \boxed{p(x_t) \propto \prod_{s \in \mathcal{N}(t)} N_{s \rightarrow t}(x_t).}$$

At factor nodes

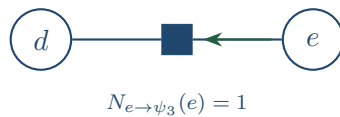
Take the product of the local potential and the incoming messages from all other adjacent variables; *sum out* all variables except the outgoing recipient.

$$\boxed{N_{s \rightarrow t}(x_t) = \sum_{\mathbf{x}_{\mathcal{N}(s) \setminus \{t\}}} \psi_s(\mathbf{x}_{\mathcal{N}(s)}) \prod_{v \in \mathcal{N}(s) \setminus \{t\}} N_{v \rightarrow s}(x_v).}$$

For a MAP estimate, replace the sum with a maximum.

Initialize missing or leaf messages to 1.

For example, from the leaf e toward d :



$$N_{\psi_3 \rightarrow d}(d) = \sum_e \psi_3(d, e) N_{e \rightarrow \psi_3}(e) = \sum_e \psi_3(d, e).$$

Observed variables and the limits of trees

If a variable is observed, substitute its observed value into all attached factors, then remove that variable from the factor graph.



Junction trees

Sum-product works on trees (including chains, a subtype of trees).

It relies on message passing along a *unique path* between nodes. In a general graph with cycles, messages can encounter repeated information. **What do we do for graphs that are not trees? Junction trees!**

Basic intuition

Work with cliques as nodes in a new tree representation:

1. Start with an MRF.
2. Adjust the graph by “triangulation.”
3. Find a junction tree from the triangulated graph.
4. Run a tree-inference algorithm on the junction tree.

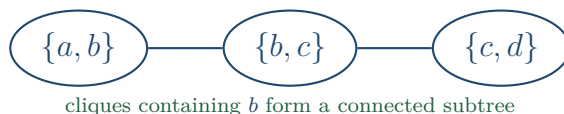
We will create a junction tree, then discuss sum-product on it.

Creating a junction tree

Definition

A tree of cliques that exhibits the *running-intersection property*:

$$C_i \cap C_j \subseteq C_k \quad \text{for every clique } C_k \text{ along the path } C_i \leftrightarrow C_j.$$

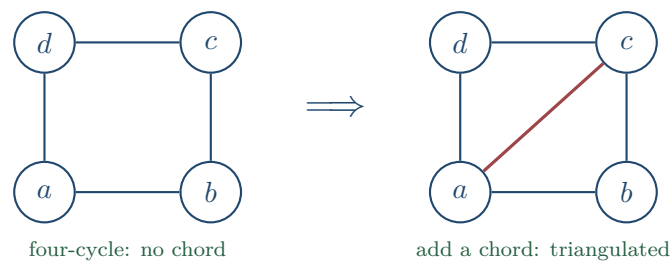


Triangulation and junction-tree construction

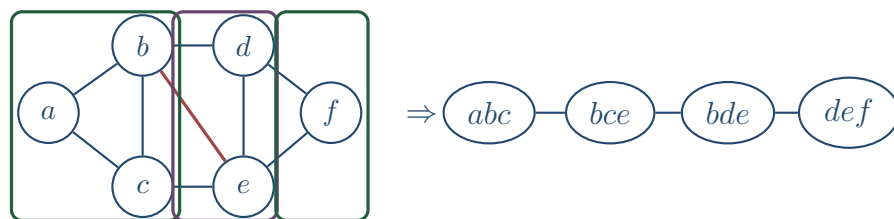
How can we ensure our clique trees have the running-intersection property?

Triangulation

Every cycle of length 4 or more has a *chord* (an edge joining nonconsecutive vertices).



For a larger graph, add chords where needed, then take maximal cliques to form the junction-tree nodes.



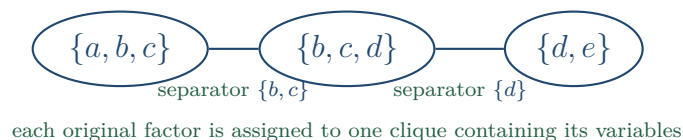
Why doesn't the transformation change the probabilistic model?

- Added edges make dependencies explicit in the computational representation.
- The original factors can still be represented; the added edges need not change the joint distribution.

Technical clarification to the handwritten note: extra edges can remove graph-implied conditional independencies, even when the joint distribution being represented is unchanged.

Junction-tree potentials

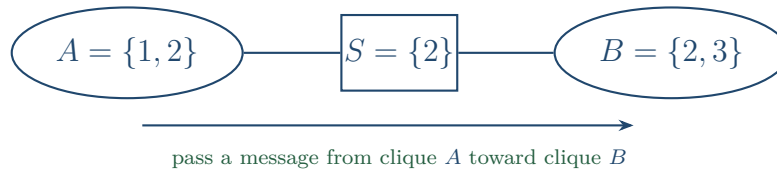
The nodes are cliques rather than individual variables. The original potential functions are assigned to cliques containing all of their variables.



Inference on a junction tree

Inference can be expressed using potential functions on both cliques (nodes) and separators.

$\phi_C(\cdot)$ clique potential, $\phi_S(\cdot)$ separator potential.



Take the current clique potential of A and sum out variables not in the separator:

$$\tilde{\phi}_S(s) = \sum_{x_{A \setminus S}} \phi_A(x_A).$$

$\tilde{\phi}_S(s)$ is the marginal information from clique A arriving at separator S .

Update the neighboring clique potential:

$$\tilde{\phi}_B(x_B) = \underbrace{\frac{\tilde{\phi}_S(x_S)}{\phi_S(x_S)}}_{\text{new / old separator message}} \phi_B(x_B).$$

Product of incoming information and the internal potential function; analogous to messages in factor graphs.

We have seen how to pass messages in directed and undirected graphical models. What about factorization in junction trees?

Factorizing a junction tree

Recall the factorization of an MRF: a normalized product of clique potentials.

$$p(x) \propto \prod_{C \in \mathcal{C}} \psi_C(x_C).$$

Simply multiplying the calibrated clique potentials of a junction tree would double count variables appearing in more than one clique.

Instead divide by the potentials of the separator sets.

Junction-tree factorization and validation

Using the clique and separator potentials:

$$p(x) \propto \frac{\prod_{C \in \mathcal{C}} \phi_C(x_C)}{\prod_{S \in \mathcal{S}} [\phi_S(x_S)]^{d(S)-1}}.$$

Here $d(S)$ is the number of maximal cliques adjacent to a particular separator set S .



2 is counted twice in the cliques

Now that we understand message passing, how do we validate the resulting marginal probabilities?

Validation

Suppose we have functions τ_C and τ_S on the clique and separator sets. When are they valid marginals of a distribution?

1. Normalization condition

For each separator set, the probabilities must sum to 1:

$$\forall S \in \mathcal{S}, \quad \sum_{x_S} \tau_S(x_S) = 1.$$

2. Marginalization condition

Each clique marginal, after summing out variables not in the separator, must agree with the distribution on that separator:

$$\forall C \in \mathcal{C}, \quad S \subseteq C : \quad \sum_{x_{C \setminus S}} \tau_C(x_C) = \tau_S(x_S).$$

$$p(B) = \sum_A p(A, B) = \sum_C p(B, C)$$

Computational complexity

For each variable in a clique, we must consider its possible states and sum over them; the work grows exponentially with clique size.

The size of the largest clique, minus one, is the *treewidth*:

$$w = \max_{C \in \mathcal{C}} |C| - 1.$$

Large treewidth can make junction-tree inference infeasible.

Junction trees work especially well for trees and chains, where the necessary cliques are small.
But what do we do about larger graphs?

Approximate inference

Variational methods for computing approximations to marginal distributions and likelihoods.

Loopy sum-product

Apply sum-product to graphs containing cycles. Initialize messages and update them repeatedly around the graph, in a manner similar to ordinary sum-product. Repeat until convergence or a maximum number of iterations is reached.

Works best when the graph is sparse or its loops are large.

Naive mean field

Similar in spirit to a Gibbs sampler, but replace sampled variable values with expectations. **To explain this, contrast it with a Gibbs sampler.**

Gibbs sampling vs. naive mean field

Gibbs sampler

1. Initialize variables.
2. Pick a variable to update.
3. Randomly sample from its conditional distribution, e.g.

$$X_i \sim p(X_i \mid X_{-i}^{\text{current}}).$$

4. Repeat for the other variables until convergence.
 - Does not assume independence.
 - Eventually samples from the joint distribution (under appropriate conditions).
 - Can be more accurate.

Even when the variables are not truly independent, the mean-field assumption greatly simplifies the mathematics.

The updated mean of a variable may be viewed as a “message” arriving at the variable vertex.

Examples

Hierarchical Bayesian models

These can be represented as graphical models and obtain the benefits of:

- MCMC and variational inference.
- Easy visualization of model dependencies.

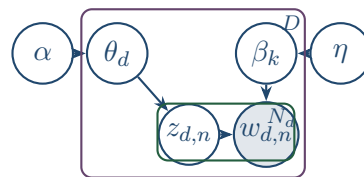
Hidden Markov models (HMMs)

Used in bioinformatics and NLP:

- Forward-backward algorithm: sum-product to compute marginals.
- Viterbi algorithm: replace sum with max to obtain MAP estimates.

Naive mean field

1. Initialize the means of variables.
2. Pick a variable to update.
3. Update its mean deterministically, using corresponding parameters / means of neighbors.
4. Repeat until convergence.
 - Assumes mutual independence in the approximation.
 - More computationally efficient.



schematic plate representation

LDA as a graphical model

$$p(\beta, \theta, z, w) = \prod_{k=1}^K p(\beta_k \mid \eta) \prod_{d=1}^D \left[p(\theta_d \mid \alpha) \cdot \prod_{n=1}^{N_d} p(z_{dn} \mid \theta_d) p(w_{dn} \mid \beta_{z_{dn}}) \right]$$

A hierarchical Bayesian model can be treated as a graphical model for inference.