Wainwright and Jordan '08: Chapter 2

Thomas Desautels

Gatsby Unit

26 January 2015

The Main Idea

The key idea is that of factorization: a graphical model consists of a collection of probability distributions that factorize according to the structure of an underlying graph... Our focus in ["Chapter" 2] is the interplay between probabilistic notions such as conditional independence on one hand, and on the other hand, graph-theoretic notions such as cliques and separation.

Some Definitions

- A *graph* $G = (V, E)$ is a collection of *vertices* $V = \{1, 2, \ldots, m\}$ and *edges* $E \subset V \times V$. Edges may be:
  - *undirected*, such that $(s, t) = (t, s)$, or
  - *directed*, where the edge from s to t is denoted $(s \to t)$.
- In a graphical model, $\forall s \in V, \exists$ random variable $X_s \in$ space $\mathcal{X}_s$.
  - Denote a particular element $x_s \in \mathcal{X}_s$, such that $X_s = x_s$ is "X_s takes value x_s."
- For $A \subset V$, $X_A \triangleq (X_s, s \in A)$, $x_A \triangleq (x_s, s \in A)$, and $\otimes_{s \in A} \mathcal{X}_s \triangleq \mathcal{X}_{A_1} \otimes \cdots \otimes \mathcal{X}_{A_{|A|}}$.

More Definitions: Directed Graphical Models

- In a directed graph, $G = (V, E)$:
  - If the edge $(s \to t) \in E$, s is the *parent* of t, which is its *child*.
  - The set of all parents of vertex t is denoted $\pi(t)$.
    - If no such parents exist, $\pi(t) = \emptyset$.
- A *directed cycle* is a sequence of vertices $\{s_1, \ldots, s_k\}$ such that $(s_i \to s_{i+1}) \in E \, \forall i \in \{1, \ldots, k-1\}$ and $(s_k \to s_1) \in E$.
  - If G is directed and does not have any such directed cycles, it is a *directed acyclic graph (DAG)*.
  - A DAG is associated with a partial ordering of the vertices.

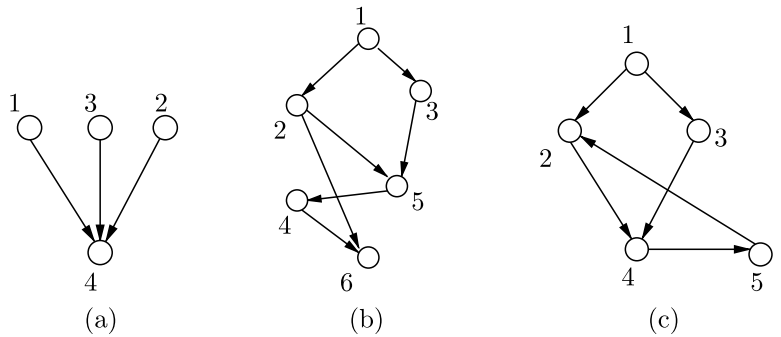


Fig. 2.1 (a) A simple directed graphical model with four variables (X_1, X_2, X_3, X_4). Vertices $\{1,2,3\}$ are all parents of vertex 4, written $\pi(4) = \{1,2,3\}$. (b) A more complicated directed acyclic graph (DAG) that defines a partial order on its vertices. Note that vertex 6 is a child of vertex 2, and vertex 1 is an ancestor of 6. (c) A forbidden directed graph (nonacyclic) that includes the directed cycle $(2 \to 4 \to 5 \to 2)$.

Directed Graphical Models

For a DAG, for each vertex s and its parent set $\pi(s)$, let there exist $p_s(x_s|x_{\pi(s)}) \geq 0$ (a conditional distribution over X_s), where $\sum_{x_s \in \mathcal{X}_s} p(x_s|x_{\pi(s)}) = 1$.
A *directed graphical model* is a collection of such distributions $\forall s \in V$ such that the joint distribution over $X_1, \ldots, X_m$ (where $|V| = m$) factorizes as

$$p(x_1, \ldots, x_m) = \prod_{s \in V} p_s(x_s|x_{\pi(s)}). \tag{1}$$

Undirected Graphical Models

Here, instead of factorizing over sets of (single child, all parents), the distribution will factorize over cliques.
A *clique* C within graph G is a fully connected subset of V (i.e., $s, t \in C \implies (s, t) \in E$)
For every clique C, let there exist a positive, scalar *compatibility function* $\psi_C : (\otimes_{s \in C} \mathcal{X}_s) \to \mathbb{R}^+$
Then the joint over all $X_s, s \in V$ is:

$$p(x_1, \ldots, x_m) = \frac{1}{Z} \prod_{C \in \mathcal{C}} \psi(x_C). \tag{2}$$

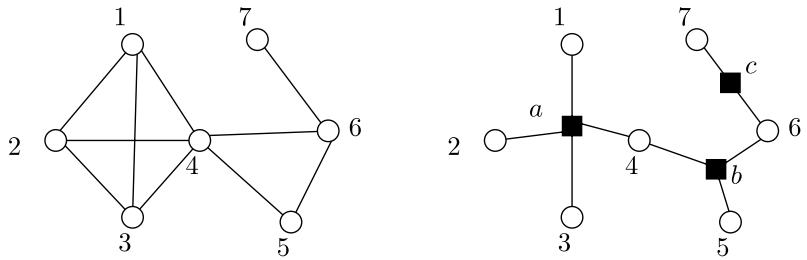
While $\mathcal{C}$ is often the set of *maximal cliques*, it can be any set of cliques which covers the graph (redundancy is permitted).

Factor Graphs

Factor graphs are a way of visualizing the factorization relationships in graphical models.

- Factors either capture the set of parent/child neighborhoods (directed graphs) or cliques (undirected).
- From $G = (V, E)$, create bipartite graph $G' = (V, F, E')$,
  - Two node types; factors (F) and random variables (V),
  - New edge set E', where E' connects vertex s to a factor a if and only if x_s is part of factor indicated by a.

Given a graph G, because the clique choice $\mathcal{C}$ is not unique, the resulting factor graph G' is not unique.



Conditional Independence I

Graph structure implies conditional independencies.

- Undirected graphical model:
  - Let the set of observed variables $V' \subseteq V$ to be a *cutset*; if the graph G' resulting from the deletion of vertices V' from G (along with their associated edges) is not connected, the variables in the disconnected parts are conditionally independent, given V'.
- Directed graphical models
  - More complicated (conditional *dependence* of parents is *induced* by observing the child), but conditional independence is also encoded in directed graphical models.

Conditional Independence II

W&J make the following points:

- For any graph structure, the implicit conditional independence assertions are consistent, and are satisfied by the set of probability distributions precisely equal to those which can be written as our directed and undirected forms of the joint distribution.
- The two characterizations are thus equivalent: factorization (algebraic) is linked with separation/reachability (graph-theoretic).
- Thus algorithmic problems where conditional independence may be necessary can be solved by appealing to the graph structure and algorithms which deal with it effectively.

Inference Problems

W&J give the following problems we might want to solve:

1. Computing the likelihood of observed data
2. Computing the marginal distribution over a particular subset $A \subset V$ of nodes.
3. Computing the conditional distribution $p(x_A|x_B)$ for disjoint subsets of variables A and B, and $A \cup B$ may be a proper subset of V
4. Computing a mode of the joint density: $\hat{x} \in \arg\max_{x \in \otimes \mathcal{X}_s} p(x)$.

The first three boil down to needing to compute the marginal distribution over x_A. The fourth is fundamentally different; it requires maximization, rather than integrating or summing out unobserved variables.

Algorithmic benefits

Naïve marginalization:

- Evaluate the joint for (exponentially many) instantiations of the marginalized variables.
- Sum over these.

Infeasible in all but the smallest problems.
By taking advantage of the graph structure, exact (in tree-structured graphs, the rest of this chapter) or approximate inference algorithms can solve these problems much more efficiently.

Hierarchical Bayesian Models

Naturally encoded in terms of conditional (i.e., hierarchical) probability distributions. GPs are a great example:

- Hyperprior $p(\theta)$ over the Gaussian process hyperparameters θ.
- θ (and input locations x) specify a prior over the regression function f, such that $f|\theta \sim \mathcal{GP}(\theta)$ and for any x in the domain, $f(x) \sim \mathcal{N}(\mu(x,\theta), \sigma(x,\theta))$.
- Observations are often taken to be noisy, such that y corresponding to x is distributed as $y|f(x) \sim N(f(x), \sigma_n)$

The joint over θ, $f(x_1), \ldots, f(x_n)$, and corresponding y_i is

$$p(y_1, \ldots, y_n, f(x_1), \ldots, f(x_n), \theta) = p(\theta)p(f(x_1), \ldots, f(x_n)|\theta) \prod_{i=1}^{n} p(y_i|f(x_i)) \tag{3}$$

Constraint Satisfaction and Combinatorial Optimization

Problem: find a configuration of logical variables X_s such that all of a collection of logical *clauses* about them are satisfied (i.e., true).

- Clauses can be viewed $\{0, 1\}$ valued compatibility functions in the undirected form.
- If there exists a clause involving a set of variables, they form a clique C in the related graph, and the set of cliques $\mathcal{C}$ is formed of the set of clause-implicit cliques.
- The overall objective function (which is either 0 or 1) can be written as the product over $\mathcal{C}$ of the individual clauses, just as in the typical undirected formalism.

You can sort of think of normal (non-zero-one) compatibility functions as “soft CSP.”

Exact Marginalization

Joint is a product of factors: exact marginalization $\Longleftrightarrow$ choosing an order of integrating or summing out the variables and ensuring that the computation is correct.
How do we correctly and efficiently organize this computation?
For each $s \in V$, could marginalize out all $m - 1$ other variables and compute these m marginals separately; however, intermediate computations are in common, and should be shared.

Message Passing Algorithms

Message passing algorithms are designed to share these intermediate computations.

Directed graphical models: convert to undirected models first.

- This requires *moralization*; for all parents $t, u \in \pi(s)$, add edge (t, u) to the graph and convert directed edges $(t \to s)$ and $(u \to s)$ to undirected edges, such that $\{s, \pi(s)\}$ is a clique.
  - This loses some information about the model.

Sum-product algorithm

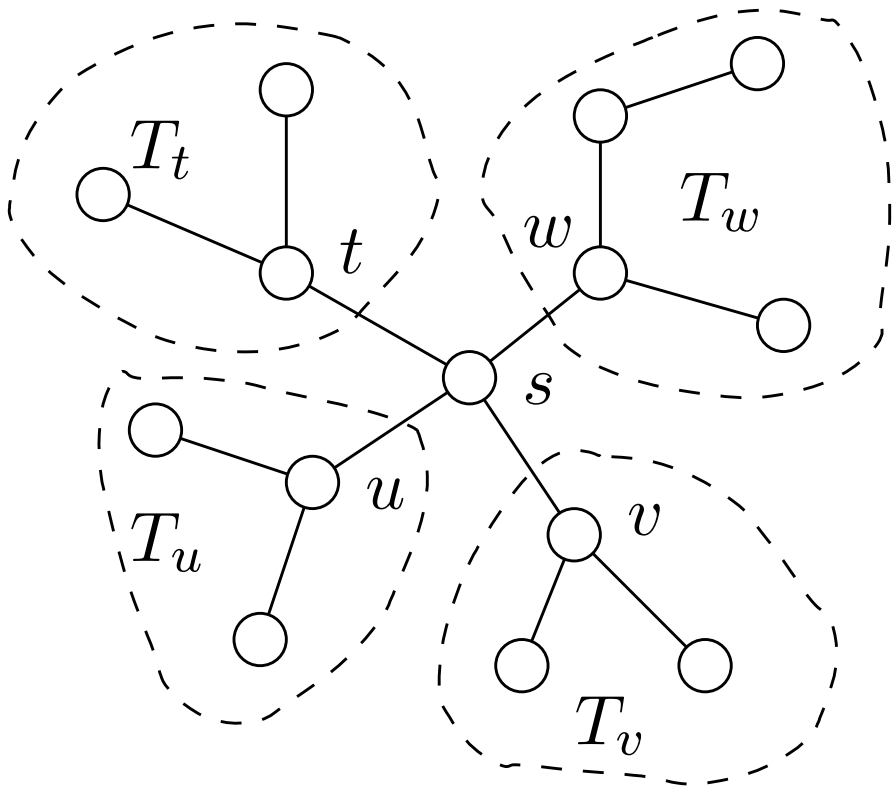
For now, assume the undirected graph is a *tree* $T = (V, E(T))$, (no cycles).

- In a tree, the set of maximal cliques is the set of edges.
- Choose $\mathcal{C} = E(T) \cup V$, and thus:

$$p(x_1, \ldots x_m) = \frac{1}{Z} \prod_{s \in V} \psi_s(x_s) \prod_{(s,t) \in E(T)} \psi_{st}(x_s, x_t). \tag{4}$$

Sum-product algorithm II

The most useful feature of tree-structured graphs is that using a single node as a cutset produces subgraphs which are disjoint, and also trees: recursive methods of divide and conquer.



Sum-product algorithm III

For each sub-tree $T_t = (V_t, E_t)$, write the product

$$p(x_{V_t}; T_t) \propto \prod_{u \in V_t} \psi_u(x_u) \prod_{u,v \in E_t} \psi_{uv}(x_u, x_v) \tag{5}$$

The joint and marginal can then be written as

$$p(x_1, \ldots, x_m) = \frac{1}{Z} \psi_s(x_s) \prod_{t \in N(s)} \left[\psi_{st}(x_s, x_t) \prod_{u \in V_t} \psi_u(x_u) \prod_{u,v \in E_t} \psi_{uv}(x_u, x_v) \right] \tag{6}$$

$$\mu(x_s) = \kappa \psi_s(x_s) \prod_{t \in N(s)} M^*_{ts}(x_s) \tag{7}$$

$$M^*_{ts}(x_s) \triangleq \sum_{x'_{V_t}} \psi_{st}(x_s, x'_t) p(x'_{V_t}; T_t) \tag{8}$$

Computing $M^*_{ts}(x_s)$ is a (smaller) sub-tree summation problem.

Sum-product algorithm IV

How can we find the M^* distributions? Iteratively, computing updated versions of all messages at each iteration. We can write the recursion in terms of the Ms (messages):

$$M_{ts}(x_s) \leftarrow \kappa \sum_{x'_t} \left\{ \psi_{st}(x_s, x'_t)\psi_t(x_t) \prod_{u \in N(t)/s} M_{ut}(x'_t) \right\} \tag{9}$$

For trees, this turns out to

- Have a unique fixed point at the M^* values (up to normalization)
- converge in finitely many iterations.

Junction Tree Algorithm

How can we deal with graphs that aren't trees (i.e., have cycles)? Turn them into trees by lumping cliques into "super-nodes" and then dealing with the *clique tree*, the tree-structured graph of such nodes. Problems:

- Building such a graph
- Ensuring consistency in the resulting algorithm
  - In building the graph, we may need to duplicate variables, if they fall inside multiple cliques; the local copies of these variables cannot be allowed to have different values if we want to compute correct marginals.

Junction Tree Algorithm II

A clique tree which satisfies the *running intersection property* will have the appropriate consistency results we need, and is called a *junction tree*.