Lecture 3: Inference in Tabular-based Graphical Models

Tianpei Xie

Aug. 25th., 2022

Contents

1 Background knowledge

Recall the formulation of Bayesian network and Markov network

- Given directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $(s,t) \neq (t,s)$, the **directed graphical model** factorizes the joint distribution into a set of *factors* $\{p_s(x_s|x_{\pi(s)}) : s \in \mathcal{V}\}$ according to the ancestor relations defined in $\mathcal{G}$

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

  This class of models are also referred as ***Bayesian networks*** [Koller and Friedman, 2009].

- Given undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $(s,t) = (t,s)$, the joint distribution of **Markov random fields** (**Markov network**) *factorize* as

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

  where Z is a constant chosen to ensure that the distribution is normalized. The set $\mathcal{C}$ is often taken to be the *set of all* ***maximal cliques*** *of the graph*, i.e., the set of cliques that are *not* properly contained within any other clique. Note that any representation based on nonmaximal cliques can always be converted to one based on maximal cliques by redefining the compatibility function on a maximal clique to be the *product* over the compatibility functions on the *subsets* of that clique.

- The canonical representation of ***exponential famlity*** of distribution has the following form

$$\begin{aligned} p(x_1, \ldots, x_m) = p(\boldsymbol{x}; \boldsymbol{\eta}) &= \exp\left(\langle \boldsymbol{\eta}\,,\, \boldsymbol{\phi}(\boldsymbol{x}) \rangle - A(\boldsymbol{\eta})\right) h(\boldsymbol{x}) \nu(d\boldsymbol{x}) \\ &= \exp\left(\sum_{\alpha} \eta_\alpha \phi_\alpha(\boldsymbol{x}) - A(\boldsymbol{\eta})\right) \end{aligned} \tag{3}$$

  where ϕ is a feature map and $\boldsymbol{\phi}(\boldsymbol{x})$ defines a set of ***sufficient statistics*** (or ***potential functions***). The normalization factor is defined as

$$A(\boldsymbol{\eta}) := \log \int \exp\left(\langle \boldsymbol{\eta}\,,\, \boldsymbol{\phi}(\boldsymbol{x}) \rangle\right) h(\boldsymbol{x}) \nu(d\boldsymbol{x}) = \log Z(\boldsymbol{\eta})$$

  $A(\boldsymbol{\eta})$ is also referred as ***log-partition function*** or *cumulant function*. The parameters $\boldsymbol{\eta} = (\eta_\alpha)$ are called ***natural parameters*** or *canonical parameters*. The canonical parameter $\{\eta_\alpha\}$ forms a **natural (canonical) parameter space**

$$\Omega = \left\{ \boldsymbol{\eta} \in \mathbb{R}^d : A(\boldsymbol{\eta}) < \infty \right\} \tag{4}$$

- The exponential family is the unique solution of ***maximum entropy estimation*** problem:

$$\min_{q \in \Delta} \quad \mathbb{KL}\left(q \,\|\, p_0\right) \tag{5}$$

$$\text{s.t.} \quad \mathbb{E}_q\left[\phi_\alpha(X)\right] = \mu_\alpha \quad \forall\, \alpha \in \mathcal{I} \tag{6}$$

where $\mathbb{KL}(q \parallel p_0) = \int \log(\frac{q}{p_0}) q dx = \mathbb{E}_q \left[\log \frac{q}{p_0}\right]$ is the relative entropy or the Kullback-Leibler divergence of q w.r.t. p_0.

Here $\boldsymbol{\mu} = (\mu_\alpha)_{\alpha \in \mathcal{I}}$ is a set of ***mean parameters***. The space of mean parameters $\mathcal{M}$ is a *convex polytope* spanned by potential functions $\{\phi_\alpha\}$.

$$\mathcal{M} := \left\{ \boldsymbol{\mu} \in \mathbb{R}^d : \exists q \text{ s.t. } \mathbb{E}_q [\phi_\alpha(X)] = \mu_\alpha \quad \forall\, \alpha \in \mathcal{I} \right\} = \operatorname{conv} \{\phi_\alpha(x),\ x \in \mathcal{X},\ \alpha \in \mathcal{I}\} \tag{7}$$

- Note that $A(\boldsymbol{\eta})$ is a convex function and its gradient $\nabla A : \Omega \to \mathcal{M}^\circ$ is a bijection between the natural parameter space Ω and the **interior** of $\mathcal{M}$, $\mathcal{M}^\circ$; $\nabla A(\boldsymbol{\eta}) = \boldsymbol{\mu}$ based on the following equation

$$\frac{\partial A}{\partial \eta_\alpha} = \mathbb{E}_{\boldsymbol{\eta}} [\phi_\alpha(X)] := \int_{\mathcal{X}^m} \phi_\alpha(\boldsymbol{x}) q(\boldsymbol{x}; \boldsymbol{\eta}) d\boldsymbol{x} = \mu_\alpha \tag{8}$$

- Moreover $A(\boldsymbol{\eta})$ has a variational form

$$A(\boldsymbol{\eta}) = \sup_{\boldsymbol{\mu} \in \mathcal{M}} \{\langle \boldsymbol{\eta}\,,\, \boldsymbol{\mu} \rangle - A^*(\boldsymbol{\mu})\} \tag{9}$$

where $A^*(\boldsymbol{\mu})$ is the conjugate dual function of A and it is defined as

$$A^*(\boldsymbol{\mu}) := \sup_{\boldsymbol{\eta} \in \Omega} \{\langle \boldsymbol{\mu}\,,\, \boldsymbol{\eta} \rangle - A(\boldsymbol{\eta})\} \tag{10}$$

It is shown that $A^*(\boldsymbol{\mu}) = -H(q_{\boldsymbol{\eta}(\boldsymbol{\mu})})$ for $\boldsymbol{\mu} \in \mathcal{M}^\circ$ which is the negative entropy. $A^*(\boldsymbol{\mu})$ is also the optimal value for the **maximum likelihood estimation** problem on p. The exponential family can be reparameterized according to its mean parameters $\boldsymbol{\mu}$ via backward mapping $(\nabla A)^{-1} : \mathcal{M}^\circ \to \Omega$, called **mean parameterization**.

- We can formulate the **KL divergence** between two distributions in exponential family Ω using its primal and dual form
  - **Primal-form**: given $\boldsymbol{\eta}_1, \boldsymbol{\eta}_2 \in \Omega$

$$\begin{aligned} \mathbb{KL}\left(p_{\boldsymbol{\eta}_1} \parallel p_{\boldsymbol{\eta}_2}\right) \equiv \mathbb{KL}(\boldsymbol{\eta}_1 \parallel \boldsymbol{\eta}_2) &= A(\boldsymbol{\eta}_2) - A(\boldsymbol{\eta}_1) - \langle \boldsymbol{\mu}_1\,,\, \boldsymbol{\eta}_2 - \boldsymbol{\eta}_1 \rangle \\ &\equiv A(\boldsymbol{\eta}_2) - A(\boldsymbol{\eta}_1) - \langle \nabla A(\boldsymbol{\eta}_1)\,,\, \boldsymbol{\eta}_2 - \boldsymbol{\eta}_1 \rangle \end{aligned} \tag{11}$$

  - **Primal-dual form**: given $\boldsymbol{\mu}_1 \in \mathcal{M}, \boldsymbol{\eta}_2 \in \Omega$

$$\mathbb{KL}(\boldsymbol{\mu}_1 \parallel \boldsymbol{\eta}_2) = A(\boldsymbol{\eta}_2) + A^*(\boldsymbol{\mu}_1) - \langle \boldsymbol{\mu}_1\,,\, \boldsymbol{\eta}_2 \rangle \tag{12}$$

  - **Dual-form**: given $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2 \in \mathcal{M}$

$$\begin{aligned} \mathbb{KL}(\boldsymbol{\mu}_1 \parallel \boldsymbol{\mu}_2) &= A^*(\boldsymbol{\mu}_1) - A^*(\boldsymbol{\mu}_2) - \langle \boldsymbol{\eta}_2\,,\, \boldsymbol{\mu}_1 - \boldsymbol{\mu}_2 \rangle \\ &\equiv A^*(\boldsymbol{\mu}_1) - A^*(\boldsymbol{\mu}_2) - \langle \nabla A^*(\boldsymbol{\mu}_2)\,,\, \boldsymbol{\mu}_1 - \boldsymbol{\mu}_2 \rangle \end{aligned} \tag{13}$$

2 Inference in graphical models

Given a probability distribution p defined by a graphical model, our focus will be solving one or more of the following computational inference problems:

1. *Computing the likelihood of* ***observed data****.* $p(\boldsymbol{y};\boldsymbol{\eta}) = \sum_{\boldsymbol{x}} p((\boldsymbol{y};\boldsymbol{x},\boldsymbol{\eta})$
2. *Computing the* ***marginal distribution*** $p(\boldsymbol{x}_A)$ *over a particular subset* $A \subset \mathcal{V}$ *of nodes.*
3. *Computing the* ***conditional distribution*** $p(\boldsymbol{x}_A|\boldsymbol{x}_B)$*, for* ***disjoint*** *subsets* A *and* B*, where* $A \cup B$ *is in general a proper subset of* $\mathcal{V}$*.* $p(\boldsymbol{x}_A|\boldsymbol{x}_B) = \frac{p(\boldsymbol{x}_A,\boldsymbol{x}_B)}{p(\boldsymbol{x}_B)}$
4. *Computing a* ***mode*** *of the density (i.e., an element* $\boldsymbol{x}$ *in the set* $\operatorname{argmax}_{\boldsymbol{x}\in\mathcal{X}^m} p(\boldsymbol{x})$.

The problem $1-3$ are similar in that it all requires to compute the marginal distribution $p(\boldsymbol{x}_A)$, which involves **summation** or **integration** depending on the factor form. In contrast, the problem of computing modes stated in 4 is fundamentally different, since it entails **maximization** rather than integration.

A **fundamental challenge** in graphical model is the *high-dimensionality* of domain, which prohibits a simple integration and summation over the entile space. Therefore, inference on graphical model usually takes advantage of the **recursive structures** within in the graph topology $\mathcal{G}$. By utilizing recursive structure of graph, efficient algorithms such as ***dynamic programming*** can be implemented.

3 Inference for tabular-based graphical models

The tabular-based graphical models store information *locally* via factors. In order to perform marginalization over the whole graph $\mathcal{G}$, adjacent factors need to ***communicate*** with each other in order to make sure that they **agree on** the marginal distribution information on their **shared common variables**, i.e. ***local consistency***. This communication is usually performed **in parallel**. Whether or not the ***global consistency*** can be reached via local consistency depends on the ***graph topolgy***. This affects the exactness of inference.

For tree-based models, we can prove that the ***local consistency*** *is equal to* ***global consistency***. This means that **exact inference** can be achieved via belief propagation at convergence. For general graphs, we transforms it into ***junction trees*** and perform belief propagtion on junction trees.

4 Exact marginal inference on tree-structure model

4.1 Sum-product belief propagation

For tree-based graphical model, we can simplify the representation (2) as

$$p(x_1,\ldots,x_m;\mathcal{T}) = \frac{1}{Z}\prod_{s\in\mathcal{V}}\psi_s(x_s)\prod_{(s,t)\in\mathcal{E}_{\mathcal{T}}}\psi_{s,t}(x_s,x_t), \tag{14}$$

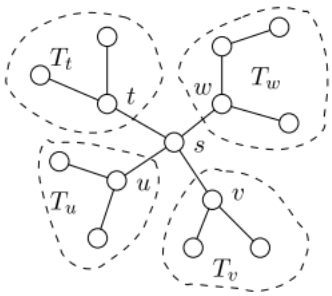

Fig. 2.10 Decomposition of a tree, rooted at node s, into subtrees. Each neighbor (e.g., u) of node s is the root of a subtree (e.g., T_u). Subtrees T_u and T_v, for $u \neq v$, are disconnected when node s is removed from the graph.

Figure 1: The decomposition of trees by subtrees rooted at neighborhood.

In **marginalization** task, the target is to find marginal distribution

$$\mu_s(x_s) := \sum_{\boldsymbol{x}_{-s}} p(x_s, \boldsymbol{x}_{-s}) \tag{15}$$

where $\boldsymbol{x}_{-s} = [x_k]_{k \neq s}$. Since (14) is a pruduct of all factors, the marginalization task is the **sum of products** among all factors over $\boldsymbol{x}_{-s}$.

As stated above, a tree $\mathcal{T} = (\mathcal{V}, \mathcal{E}_{\mathcal{T}})$ has clear a **recursive structure**. Specifically, define the neighborhood of node s as

$$\mathcal{N}(s) = \{t \in \mathcal{V} : (s,t) \in \mathcal{E}_{\mathcal{T}}\}$$

For each $u \in \mathcal{N}(s)$, let $\mathcal{T}_u = (\mathcal{V}_u, \mathcal{E}_u)$ be the subgraph formed by the set of nodes (and edges joining them) that *can be reached from* u by **paths** that ***do not pass*** through node s. An **important property** for tree is that the subgraph $\mathcal{T}$ is a tree and for any $u \neq t, \forall u, t \in \mathcal{N}(s)$, $\mathcal{T}_u \cap \mathcal{T}_t = \emptyset$. This means that we can **decompose** the tree $\mathcal{T}_s$ rooted at s, by **removing** s from the tree. It then form a collection or non-overlapping subtrees $\{\mathcal{T}_t, t \in \mathcal{N}(s)\}$. Figure 2 shows the tree decomposition.

In particular, we introduce the ***Sum-Product algorithm***, a form of ***nonserial dynamic programming*** [Bertele and Brioschi, 1973], which generalizes the usual serial form of deterministic dynamic programming [Bertsekas, 2012] to arbitrary tree-structured graphs. The essential idea is divide and conque: dividing the problem of marginalization over **all variables** into the problem of marginalization over **local variables within each factor**; and "**passing**" this marginalized probability as "***message***" to other nodes in the neigborhood. See that

$$\begin{aligned}
\mu_s(x_s) &= p(x_s) \sum_{\boldsymbol{x}_{-s}} p(\boldsymbol{x}_{-s} | x_s) \\
&= p(x_s) \sum_{\boldsymbol{x}_{-s}} \prod_{t \in \mathcal{N}(s)} p_t(\boldsymbol{x}_{\mathcal{V}_t} | x_s; \mathcal{T}_t) \quad (\text{since } x_{\mathcal{V}_u} \perp\!\!\!\perp x_{\mathcal{V}_v} | x_s \, \forall u, v \in \mathcal{N}(s)) \\
&= p(x_s) \prod_{t \in \mathcal{N}(s)} \left[\sum_{\boldsymbol{x}_{\mathcal{V}_t}} p_t(\boldsymbol{x}_{\mathcal{V}_t} | x_s; \mathcal{T}_t) \right] \quad (\text{ distribution law })
\end{aligned}$$

Define the **value function** that encodes the *local information* to **be passed** from subtree $\mathcal{T}_t \to s$ as a ***message***

$$M^*_{t\to s}(x_s) := M^*_{t,s}(x_s) = \sum_{\boldsymbol{x}_{\mathcal{V}_t}} \psi_{s,t}(x_s, x_t) p(\boldsymbol{x}_{\mathcal{V}_t}; \mathcal{T}_t) \tag{16}$$

where $p(\boldsymbol{x}_{\mathcal{V}_t}; \mathcal{T}_t)$ is the joint distribution of subtree $\mathcal{T}_t$ following the same factorization as (14)

$$p(\boldsymbol{x}_{\mathcal{V}_t}; \mathcal{T}_t) = \frac{1}{Z} \prod_{u \in \mathcal{V}_t} \psi_u(x_u) \prod_{(u,v) \in \mathcal{E}_{\mathcal{T}_t}} \psi_{u,v}(x_u, x_v).$$

Given all messages $\{M^*_{t,s}(x_s)\}_{t \in \mathcal{N}(s)}$ from neighboring subtrees $\{\mathcal{T}_t, t \in \mathcal{N}(s)\}$, the ***marginal distribution*** can be seen as a **value function at node** s as well:

$$\mu_s(x_s) = \psi_s(x_s) \prod_{t \in \mathcal{N}(s)} M^*_{t,s}(x_s). \tag{17}$$

For subtree $\mathcal{T}_t$, the computation of message $M^*_{t,s}(x_s)$ is again a sub-problem of marginalization. We can see that $(\mu_s(x_s), (M^*_{t,s}(x_s))_{t \in \mathcal{N}(s)})$ are all value functions which store local results for the problem.

Applying the decomposition twice and combing (16) and (17), we can obtain the ***Bellman equation*** w.r.t. the **message function** $M^*_{t,s}(x_s)$

$$\begin{aligned} M^*_{t,s}(x_s) &= \sum_{\boldsymbol{x}_{\mathcal{V}_t}} \psi_{s,t}(x_s, x_t) p(\boldsymbol{x}_{\mathcal{V}_t}; \mathcal{T}_t) \\ &= \kappa \sum_{x_t \in \mathcal{X}} \left\{ \psi_{s,t}(x_s, x_t) \psi_t(x_t) \prod_{u \in \mathcal{N}(t) - \{s\}} M^*_{u,t}(x_t) \right\}, \forall t \in \mathcal{N}(s), s \in \mathcal{V} \end{aligned} \tag{18}$$

where $\kappa > 0$ again denotes a *normalization constant*.

The **Sum-Product** algorithm is described as

1. At each iteration, each node t **passes a message** $M^*_{t,u}(x_u)$ to **each of its neighbors** $u \in \mathcal{N}(t)$. $M^*_{t,u}(x_u)$ is a function of the possible states $x_u \in \mathcal{X}_u$ (i.e., a vector of length $|\mathcal{X}_u|$ for discrete random variables).

   This operation is done **in parallel** for all t. There are total of $2|\mathcal{E}|$ messages in transmission.

2. Then the mssages from $t \to s$ are updated based on messages $u \to t$ from neighboring nodes $u \in \mathcal{N}(t) - \{s\}$

$$M^*_{t,s}(x_s) \leftarrow \kappa \sum_{x_t \in \mathcal{X}} \left\{ \psi_{s,t}(x_s, x_t) \psi_t(x_t) \prod_{u \in \mathcal{N}(t) - \{s\}} M^*_{u,t}(x_t) \right\},$$

3. Repeat the above steps until converge

The algorithm will **converge to fixed-point solution** $M^* := \left\{ M^*_{t,s}, M^*_{s,t}, (s,t) \in \mathcal{E}_{\mathcal{T}} \right\}$ for tree-structured graphs after a finite number of iterations. This solution is the solution of Bellman

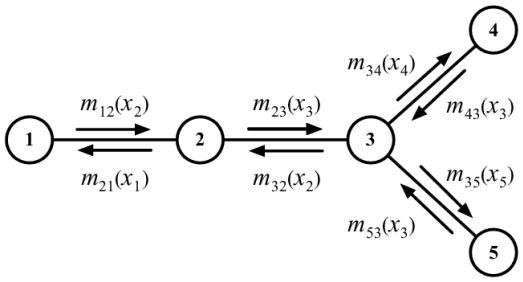

Figure 2: The sum-product message passing, i.e. belief propagation.

equation (18). Since the fixed point $M_{t,s}^*$ specifies the solution to all of the subproblems, the marginal μ_s at every node $s \in \mathcal{V}$ can be computed easily via equation (17).

Specifically, for tree-based graphical model, the sum-product iteration is done via **forward-backward procedue**, i.e. starting **from root** passing messages down to **leaves requesting** marginal probability information; and then **from leaf** passing message up to **root filling** in marginal probability information. The sum-product algorithm is also known as ***belief propagation algorithm*** [Wainwright et al., 2008].

4.2 Factor graph message passing

In ***factor-graph*** formulation, the definitions of **variable node** and **factor node** are introduced. The joint probability is factorized into product of factors.

$$p(x_1, \ldots, x_m) = \frac{1}{Z} \prod_{a \in \mathcal{C}} \psi_a(\boldsymbol{x}_a), \tag{19}$$

The message in (16) is split into **two messages**.

1. **Messages from a variable node t to a factor node a**:

$$M_{t \to a}(x_t) = \prod_{a' \in \mathcal{N}(t) - \{s\}} M_{a' \to t}(x_t) \tag{20}$$

   Note that in factor graph, the neighborhood of a variable node $\mathcal{N}(t)$ only contains factor nodes, thus a' is a factor node and $M_{a' \to t}$ is a message from factor to variable.

2. **Messages from a factor node a to a variable node t**: It is defined to be the product of the factor with messages from all other nodes, marginalized over all variables except the one associated with t

$$M_{a \to t}(x_t) = \sum_{\boldsymbol{x}_{a.-t}} \left\{ \psi_a(\boldsymbol{x}_{a,-t}, x_t) \prod_{w \in \mathcal{N}(a) - \{t\}} M_{w \to s}(x_w) \right\} \tag{21}$$

   Here $\mathcal{N}(a)$ only contains variable node so $M_{w \to s}$ is the message from variable node w to factor s. Also $\boldsymbol{x}_{a,-t} = (x_w, w \in \mathcal{N}(a) - \{t\})$ denote all input of factor a except for variable x_t.

Note that for tree, each (edge) factor only contains two nodes, and each node only connects to two (edge) factors. So these two messages are combined.

The marginal distribution of node s, $\mu_s(x_s)$

$$\mu_s(x_s) \propto \prod_{a \in \mathcal{N}(s)} M_{a \to s}(x_s)$$

and the marginal distribution of $\boldsymbol{x}_a$ belongs to factor a

$$\mu_a(\boldsymbol{x}_a) \propto \psi_a(\boldsymbol{x}_a) \prod_{t \in \mathcal{N}(a)} M_{t \to a}(x_t)$$

4.3 Bellman equation in log-space

The Bellman equation in (18) with tabular factor ψ is for all $x_s \in \mathcal{X}, t \in \mathcal{N}(s), s \in \mathcal{V}$. It is a system of $2|\mathcal{E}| \times |\mathcal{X}|$ linear equations. For tree structure, $|\mathcal{V}| = |\mathcal{E}| + 1$. Let $|\mathcal{V}| = d$, $|\mathcal{X}| = r$, then $\boldsymbol{m} = [M^*_{t,s}(x_s)]_{\forall (s,t) \in \mathcal{E}, x_s \in \mathcal{X}} \in \mathbb{R}^{2(d-1)r}$. We can replace the Bellman equation with two steps

$$H_{s,t}(x_s, x_t) = \psi_{s,t}(x_s, x_t)\psi_t(x_t) \prod_{u \in \mathcal{N}(t) - \{s\}} M^*_{u,t}(x_t)$$

$$M^*_{t,s}(x_s) = \sum_{x_t} H_{s,t}(x_s, x_t)$$

and then introduce the logarithm $h_{s,t}(x_s, x_t) = \log H_{s,t}(x_s, x_t)$ and $m_{s,t}(x_s) = \log M^*_{t,s}(x_s)$. We have Bellman equation in log-space

$$h_{s,t}(x_s, x_t) = \log \psi_{s,t}(x_s, x_t) + \log \psi_t(x_t) + \sum_{u \in \mathcal{N}(t) - \{s\}} m_{u,t}(x_t) \tag{22}$$

$$m^*_{t,s}(x_s) = \log \left(\sum_{x_t \in \mathcal{X}} \exp(h_{s,t}(x_s, x_t)) \right) := \text{soft-max}_{x_t} \left(h_{s,t}(x_s, x_t) \right) \tag{23}$$

$$\Rightarrow m^*_{t,s}(x_s) = \text{soft-max}_{x_t \in \mathcal{X}} \left[\log \psi_{s,t}(x_s, x_t) + \log \psi_t(x_t) + \sum_{u \in \mathcal{N}(t) - \{s\}} m_{u,t}(x_t) \right] \tag{24}$$

Here the log-sum-exp function is a **smooth version of maximum** $\log(\sum_t e^{h_t}) = \text{soft-max}_t(h)$.

5 Exact mode inference on tree-structure model

The problem of finding a **mode** of a tree-structured distribution is formulated as

$$\max_{\boldsymbol{x} \in \mathcal{X}^m} \frac{1}{Z} \prod_{s \in \mathcal{V}} \psi_s(x_s) \prod_{(s,t) \in \mathcal{E}_\mathcal{T}} \psi_{s,t}(x_s, x_t) \tag{25}$$

Similar to Sum-Product algorithm, we have **Max-Product algorithm** where maximization replaces the summation. Max-Product algorithm is essentially a ***generalized Viterbi algorithm***. We consider the problem to compute the ***max-marginal***

$$\nu_s(x_s) := \max_{\boldsymbol{x}_{-s}} p(x_s, \boldsymbol{x}_{-s}) \tag{26}$$

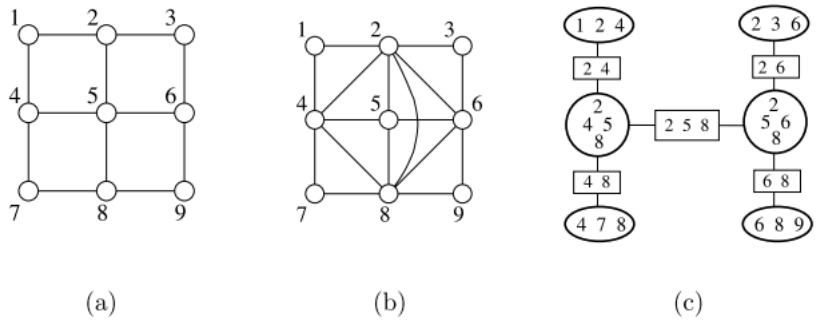

Fig. 2.11 Illustration of junction tree construction. (a) Original graph is a 3×3 grid. (b) Triangulated version of original graph. Note the two 4-cliques in the middle. (c) Corresponding junction tree for triangulated graph in (b), with maximal cliques depicted within ellipses, and separator sets within rectangles.

Figure 3: Triangulation and junction tree. [Wainwright et al., 2008]

$$
\begin{aligned}
&= p(x_s) \max_{\boldsymbol{x}_{-s}} p(\boldsymbol{x}_{-s}|x_s) \\
&= p(x_s) \max_{\boldsymbol{x}_{-s}} \prod_{t \in \mathcal{N}(s)} p_t(\boldsymbol{x}_{\mathcal{V}_t}|x_s; \mathcal{T}_t) \quad \text{(by tree decomposition)} \\
&= p(x_s) \prod_{t \in \mathcal{N}(s)} \left[\max_{\boldsymbol{x}_{\mathcal{V}_t}} p_t(\boldsymbol{x}_{\mathcal{V}_t}|x_s; \mathcal{T}_t) \right] \quad \text{(monotonicity of maximum)}
\end{aligned}
$$

So from (16), we see that the **value functions** are defined as

$$M^*_{t \to s}(x_s) := M^*_{t,s}(x_s) = \max_{\boldsymbol{x}_{\mathcal{V}_t}} \psi_{s,t}(x_s, x_t) \, \underline{p(\boldsymbol{x}_{\mathcal{V}_t}; \mathcal{T}_t)} \tag{27}$$

$$\nu_s(x_s) = \psi_s(x_s) \prod_{t \in \mathcal{N}(s)} M^*_{t,s}(x_s). \tag{28}$$

Following similar derivation as (18), we have the **Bellman equation** for **max-product** algorithm:

$$M^*_{t,s}(x_s) = \max_{x_t} \left\{ \psi_{s,t}(x_s, x_t) \psi_t(x_t) \prod_{u \in \mathcal{N}(t) - \{s\}} M^*_{u,t}(x_t) \right\} \quad \forall t \in \mathcal{N}(s), \; s \in \mathcal{V} \tag{29}$$

For HMM, since there are only one input $u = t - 1$ besides $t + 1$ in the neighborhood of t, $M^*_{t,t+1}(x_{t+1}) = \max_{x_t} M^*_{(t-1),t}(x_t) \; \psi_{t,t+1}(x_t, x_{t+1}) \psi_t(x_t)$. This is the objective of Viterbi decoding.

6 Junction tree representation

Given a graph with cycles, a natural idea is to cluster its nodes so as to form a ***clique tree*** – that is, an *acyclic* graph whose *nodes* are formed by the *maximal cliques* of $\mathcal{G}$. In order to apply the message passing algorithm to clique trees, a clique tree satisfy an additional restriction so as to

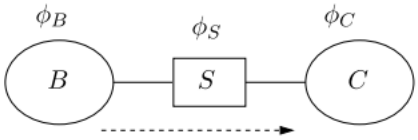

Fig. 2.12 A message-passing operation between cliques B and C via the separator set S.

Figure 4: Message passing through seperator clique.

ensure **correctness** of these computations. In particular, since a given vertex $s \in \mathcal{V}$ may appear in multiple cliques (say C_1 and C_2), what is required is a mechanism for enforcing ***consistency*** among the different appearances of the variable x_s. It turns out that the following property is ***necessary*** *and* ***sufficient*** to enforce such consistency:

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

Junction tree can be constructed using ***triangulation***. We say that a graph is ***triangulated*** if every cycle of length *four* or longer has a *chord*, meaning an edge joining a pair of nodes that are not adjacent on the cycle.

The following is the **key theorem** behind the junction tree algorithm.

Theorem 6.1 *[Jordan, 2003, Wainwright et al., 2008]*
For an undirected graph, $\mathcal{G}$, the following properties are equivalent*:*

- *Graph $\mathcal{G}$ is* ***triangulated****.*
- *The clique graph of $\mathcal{G}$ has a* ***junction tree****.*
- *There is an elimination ordering for $\mathcal{G}$ that does not lead to any added edges.*

See Figure 3 for construction of junction tree via triangulation.

This result underlies the ***junction tree algorithm*** [Jordan, 2003] for exact inference on *arbitrary graphs*:

1. Given a graph with cycles $\mathcal{G}$, ***triangulate*** it by adding edges as necessary.
2. Form a ***junction tree*** associated with the triangulated graph $\tilde{\mathcal{G}}$.
3. Run a ***tree inference*** algorithm on the junction tree.

To describe the inference on junction tree, we need to introduce potential functions not only on the cliques in the junction tree, but also on the ***separators*** in the junction tree. A *separator* $S = C_1 \cap C_2$ is the intersections of cliques that are *adjacent* in the junction tree. Let $\phi_C(\cdot)$ denote a potential function defined over the subvector $\boldsymbol{x}_C = (x_t, t \in C)$ on clique C, and let $\phi_S(\cdot)$ denote a potential on a separator S.

We **initialize** the clique potentials by assigning each *compatibility function* in the original graph to (exactly) one *clique potential* and taking the **product** over these compatibility functions. The

separator potentials are initialized to **unity**.

$$\phi_C(\boldsymbol{x}_C) \leftarrow \prod_{c \subset C} \psi_c(\boldsymbol{x}_c)$$
$$\phi_S(\boldsymbol{x}_S) \leftarrow 1$$

Given this set-up, the basic **message-passing** step of the junction tree algorithm are given by

$$\widetilde{\phi}_S(\boldsymbol{x}_S) \leftarrow \sum_{\boldsymbol{x}_{B-S}} \phi_B(\boldsymbol{x}_B) \tag{30}$$

$$\phi_C(\boldsymbol{x}_C) \leftarrow \frac{\widetilde{\phi}_S(\boldsymbol{x}_S)}{\phi_S(\boldsymbol{x}_S)} \phi_C(\boldsymbol{x}_C) \tag{31}$$

The above describe the message passing from clique B to clique C, as illustrated in Figure 4.

It can be verified that if a message is passed from B to C, and subsequently from C to B, then the resulting clique potentials are consistent with each other, meaning that **they agree** with respect to the vertices S.

References

Umberto Bertele and Francesco Brioschi. On non-serial dynamic programming. *J. Comb. Theory, Ser. A*, 14(2):137–148, 1973.

Dimitri Bertsekas. *Dynamic programming and optimal control: Volume I*, volume 1. Athena scientific, 2012.

Michael I Jordan. An introduction to probabilistic graphical models. *University of California, Berkeley*, 2003.

Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

Martin J Wainwright, Michael I Jordan, et al. Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 1(1–2):1–305, 2008.