Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science

6.438 Algorithms For Inference
Fall 2014

10 Sum-product on factor tree graphs, MAP elimination algorithm

The previous lecture introduced the beginnings of the sum-product on factor tree graphs with a specific example. In this lecture, we will describe the sum-product algorithm for general *factor tree graphs*.

10.1 Sum-product for factor trees

Recall that a factor graph consists of a bipartite graph between variable nodes and factor nodes. The graph represents the factorization of the distribution into factors corresponding to factor nodes in the graph, in other words

$$p(x_1, \ldots, x_N) \propto \prod_{i=1}^{M} f_i(x_{c_i})$$

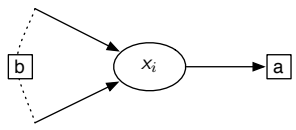
where c_i is the set of variables connected to factor f_i. As in the case with undirected graphs, we restrict our attention to a special class of graphs where inference is efficient.

Definition 1 (Factor tree graph) *A factor tree graph is a factor graph such that the variable nodes form a tree, that is between any two variable nodes, there is a unique path (involving factors).*

10.1.1 Serial version

As in the sum-product algorithm for undirected graphs, there is a serial verison of the sum-product algorithm for factor tree graphs. Because there are factor nodes and variables nodes, we have two types of messages:

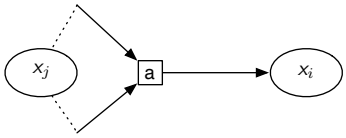
1. Variable to factor messages:



$$m_{i \to a}(x_i) = \prod_{b \in N(i) \setminus a} m_{b \to i}(x_i)$$

The variable node i with degree d_i sends a message towards a that is the product of $d_i - 1$ terms for each value in $|\mathcal{X}|$. This results in a total of $O(|\mathcal{X}| d_i)$ operations.

2. Factor to variable messages:



$$m_{a\to i}(x_i) = \sum_{x_j : j\in N(a)\setminus i} f_a(x_i; x_j, j \in N(a)\setminus i) \prod_{j\in N(a)\setminus i} m_{j\to a}(x_j)$$

The factor node a with degree d_a sends a message towards i that is a summation over $|\mathcal{X}|^{d_a-1}$ terms, each involving a product of d_a terms for each value in $|\mathcal{X}|$ of the neighboring variables. Thus, the total cost is $O(|\mathcal{X}|^{d_a} d_a)$.

In summary, complexity scales as $O(|\mathcal{E}||\mathcal{X}|^{d^*})$ where $d^* = \max_a d_a$ (over factor nodes) and $|\mathcal{E}|$ is the total number of edges in the factor tree graph. Notably, because the graph is a tree, $|\mathcal{E}| = O(N)$.[1]

10.1.2 Parallel version

We can also run the updates in parallel to get the parallel version of the sum-product algorithm for factor tree graphs:

1. **Initialization:** for $(i, a) \in \mathcal{E}$,

$$m^0_{i\to a}(x_i) = 1$$
$$m^0_{a\to i}(x_i) = 1$$

for all $x_i \in \mathcal{X}$.

2. **Update:** for $(i, a) \in \mathcal{E}, t \geq 0$,

$$\textbf{(*)}\ m^{t+1}_{i\to a}(x_i) = \prod_{b\in N(i)\setminus a} m^t_{b\to i}(x_i)$$
$$\textbf{(**)}\ m^{t+1}_{a\to i}(x_i) = \sum_{x_j : j\in N(a)\setminus i} f_a(x_i; x_j, j \in N(a)\setminus i) \prod_{j\in N(a)\setminus i} m^t_{j\to a}(x_j)$$

3. **Marginalization:** for $t \geq$ diameter

$$p^{t+1}_{x_i}(x_i) \propto \prod_{a\in N(i)} m^{t+1}_{a\to i}(x_i).$$

[1] This is the cost to send all of the messages to a single node. If we wanted to send all of the messages the other direction, this would naively cost $O(N^2|\mathcal{X}|^{d^*})$. We can reduce this to $O(N|\mathcal{X}|^{d^*})$ using a technique that we show in the next section.

As we noticed above sending $m_{a\to i}$ costs $O(|\mathcal{X}|^{d_a} d_a)$ and we have to send a similar message for every neighbor of a, so the total complexity cost will scale as $O(|\mathcal{X}|^{d^*} \sum_a d_a^2)$ and in the worst case $O(\sum_a d_a^2)$ will be quadratic in N.

With a little cleverness, we can provide a more efficient implementation of (*) and (**):

(*) Compute

$$m_i^{t+1}(x_i) = \prod_{b \in N(i)} m_{b\to i}^t(x_i)$$

then,

$$m_{i\to a}^{t+1}(x_i) = \frac{m_i^{t+1}(x_i)}{m_{a\to i}^t(x_i)}.$$

Thus the total cost of computing (*) per variable node i is $O(|\mathcal{X}|d_i)$ instead of $O(|\mathcal{X}|d_i^2)$.

(**) Compute

$$\tilde{f}_a^{t+1}(x_k, k \in N(a)) = f_a(x_k; k \in N(a)) \prod_{k \in N(a)} m_{k\to a}^t(x_k)$$

then,

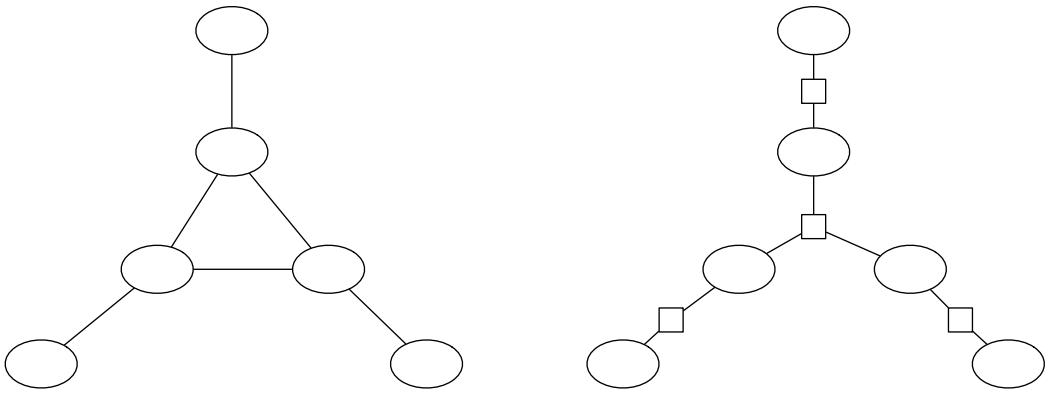
$$m_{a\to i}^{t+1}(x_i) = \sum_{x_j : j \in N(a)\setminus i} \frac{\tilde{f}_a^{t+1}(x_k, k \in N(a))}{m_{i\to a}^t(x_i)}.$$

Thus the total cost of computing (**) per factor node a is $O(|\mathcal{X}|^{d_a} d_a)$.

Thus the overall complexity cost for a single iteration of the parallel algorithm will be $O(N|\mathcal{X}|^{d^*})$.

10.2 Factor tree graph > Undirected tree graph

The factor tree graph representation is strictly more general than the undirected tree graph representation. Recall that by the Hammersley-Clifford theorem, we can represent any undirected graph as a factor graph by creating a factor node for each clique potential and connecting the respective variable nodes. In the case of an undirected tree graph, the only clique potentials are along edges, so it's clear the factor graph representation will also be a tree. Now, we consider an example where the undirected graph is *not* a tree, but its factor graph is:



As shown in the figure, the graphical model on the left has 4 maximal cliques. The factor graph representation of it is on the right, and it does not have any loops, even though the undirected graph did. A necessary and sufficient condition for this situation is provided in the solution of Problem 2.2.

On the left hand graph, we cannot apply the sum-product algorithm because the graph is not a tree, but on the right hand side we can. What happened here? The factor in the middle of the graph has 3 nodes, and as we saw above, the overall complexity of the algorithm depends strongly on the maximal degree node. Taking this example further, consider converting the complete undirected graph on N vertices into a factor graph. It is a star graph with a single factor node, so it is a tree Unfortunately, the factor node has N vertices, so the cost of running sum-product is $O(N|\mathcal{X}|^N)$, so we don't get anything for free.

10.3 MAP elimination algorithm

The primary focus of the prior material has been on algorithms for computing marginal distributions. In this section, we will be talking about the second inference problem: computing the mode or MAP.

Specifically, consider a collection of N random variables $x_1, \ldots x_N$ each taking values in $|\mathcal{X}|$ with their joint distribution represented by a graphical model $\mathcal{G}$. For concreteness, let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected graphical model, so that the distribution factorizes as:

$$\begin{aligned} p_{\mathbf{x}}(\mathbf{x}) &\propto \prod_{C \in \mathcal{C}} \phi_C(x_C) \\ &= \frac{1}{Z} \prod_{C \in \mathcal{C}} \phi_C(x_C), \end{aligned}$$

where $\mathcal{C}$ is the set of maximal cliques in $\mathcal{G}$ and Z is the partition function. Our goal is to compute $\mathbf{x}^* \in \mathcal{X}^N$ such that

$$\mathbf{x}^* \in \arg\max_{\mathbf{x} \in \mathcal{X}^N} p_{\mathbf{x}}(\mathbf{x}).$$

Note that we have written $\in$ instead of $=$ because there may be several configurations that result in the maximum and we are interested in finding one of them.

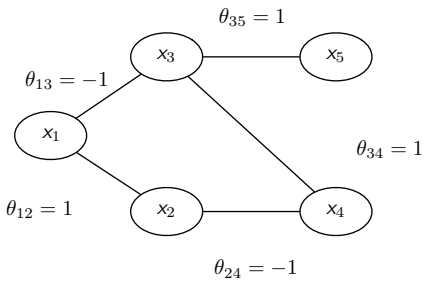
The sum-product algorithm gives us a way to compute the marginals $p_{x_i}(x_i)$, so we might wonder if setting $x_i^* \in \arg\max_{x_i} p_{x_i}(x_i)$ produces a MAP assignment. This works in situations where the variables are independent, there is only one positive configuration, or when the variables are jointly Gaussian. However, consider the following example with binary variables x_1 and x_2:

$$\begin{aligned} p_{x_1,x_2}(0,0) &= 1/3 - \epsilon \\ p_{x_1,x_2}(1,0) &= 1/3 \\ p_{x_1,x_2}(0,1) &= 0 \\ p_{x_1,x_2}(1,1) &= 1/3 + \epsilon. \end{aligned}$$

$\arg\max_{x_1} p_{x_1}(x_1) = 1$ and $\arg\max_{x_2} p_{x_2}(x_2) = 0$, but that is not the MAP. It is clear that we need a more general algorithm.

10.4 MAP Elimination algorithm example

We will start with a motivating example of the MAP elimination algorithm. Consider a distribution over $(x_1, \ldots, x_5)$ described by the graphical model below. Let each $x_i \in \{0, 1\}$, for $1 \leq i \leq 5$.



$$p(\mathbf{x}) \propto \exp(\theta_{12}x_1x_2 + \theta_{13}x_1x_3 + \theta_{24}x_2x_4 + \theta_{34}x_3x_4 + \theta_{35}x_3x_5)$$

For this example, we will consider the specific values $\theta_{12} = \theta_{34} = \theta_{35} = 1, \theta_{13} = \theta_{24} = -1$. So, plugging in those values, our distribution is

$$\begin{aligned} p(\mathbf{x}) &\propto \exp(x_1x_2 - x_1x_3 - x_2x_4 + x_3x_4 + x_3x_5) \\ &= \frac{1}{Z} \underbrace{\exp(x_1x_2 - x_1x_3 - x_2x_4 + x_3x_4 + x_3x_5)}_{F(x_1,\ldots,x_5)} \end{aligned}$$

Our goal is to efficiently compute:

$$\mathbf{x}^* \in \arg\max_{\mathbf{x}\in\mathcal{X}^N} p_{\mathbf{x}}(\mathbf{x}) \propto F(x_1, \ldots, x_5).$$

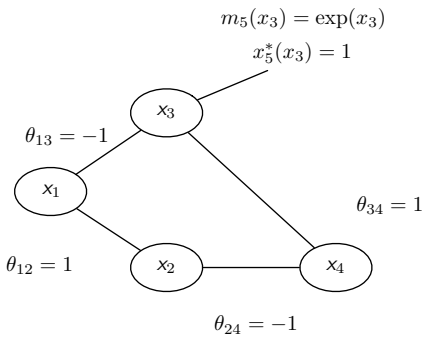
To efficiently compute this mode $\mathbf{x}^*$, let us adapt the elimination algorithm for computing the marginals of a distribution. First, we will fix an elimination order of nodes: $5, 4, 3, 2, 1$. Then

$$\max_{\mathbf{x}} p_{\mathbf{x}}(\mathbf{x}) = \max_{x_1,\ldots,x_4} \exp(x_1x_2 - x_1x_3 - x_2x_4 + x_3x_4) \underbrace{\left(\max_{x_5} \exp(x_3x_5)\right)}_{\triangleq m_5(x_3)}$$

where we have grouped the terms involving x_5 and moved the $\max_{x_5}$ past all of the terms that do not involve x_5. We see that

$$m_5(x_3) = e^{x_3}$$

here. We also store the specific value of x_5 that maximized the expression (e.g. when $x_3 = 1$, $m_5(x_3) = e$ and $x_5^* = 1$) and we will see why this is important to track later. In this case, the maximizer of x_5 is 1 if $x_3 = 1$ and 0 or 1 if $x_3 = 0$. We are free to break ties arbitrarily, so we break the tie so that $x_5^*(x_3) = x_3$.

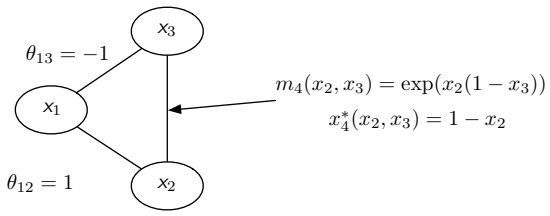


Now we eliminate x_4 in the same way, by grouping the terms involving x_4 and moving the $\max_{x_4}$ past terms that do not involve it.

$$\max_{\mathbf{x}} p_{\mathbf{x}}(\mathbf{x}) = \max_{x_1,x_2,x_3} \exp(x_1x_2 - x_1x_3)m_5(x_3) \underbrace{\left(\max_{x_4} \exp(x_3x_4 - x_2x_4)\right)}_{\triangleq m_4(x_2,x_3)}$$

with

$$m_4(x_2, x_3) = \exp(x_3(1 - x_2))$$
$$x_4^*(x_2, x_3) = 1 - x_2.$$



Similarly for x_3,

$$\begin{aligned}
\max_{\mathbf{x}} p_{\mathbf{x}}(\mathbf{x}) &= \max_{x_1,x_2} \exp(x_1 x_2) \left(\max_{x_3} \exp(-x_1 x_3) m_5(x_3) m_4(x_2, x_3) \right) \\
&= \max_{x_1,x_2} \exp(x_1 x_2) \left(\max_{x_3} \exp(-x_1 x_3 + x_3 + x_3(1 - x_2)) \right) \\
&= \max_{x_1,x_2} \exp(x_1 x_2) \underbrace{\left(\max_{x_3} \exp(-(x_1 + x_2) x_3) \right)}_{\triangleq m_3(x_1,x_2)}
\end{aligned}$$

with

$$\begin{aligned}
m_3(x_1, x_2) &= 1 \\
x_3^*(x_1, x_2) &= 0.
\end{aligned}$$

Finally,

$$\max_{\mathbf{x}} p_{\mathbf{x}}(\mathbf{x}) = \max_{x_1,x_2} \exp(x_1 x_2)$$

implies that $x_1^* = x_2^* = 1$. Now we can use the information we stored along the way to compute the MAP assignment. $x_3^*(1, 1) = 0$, $x_4^*(1, 0) = 0, x_5^*(0) = 1$. Thus the optimal assignment is:

$$\mathbf{x}^* = (1, 1, 0, 0, 1).$$

As in the elimination algorithm for computing marginals, the primary determinant of the computation cost is the size of the maximal clique in the reconstituted graph. Putting this together, we can describe the MAP elimination algorithm for general graphs:

Input: Potentials φ_c for $c \in \mathcal{C}$ and an elimination ordering I
Output: $\mathbf{x}^* \in \arg\max_{\mathbf{x} \in \mathcal{X}^N} p_{\mathbf{x}}(\cdot)$
Initialize active potentials Ψ to be the set of input potentials.
for *node i in I* **do**
Let S_i be the set of all nodes (not including i and previously eliminated nodes) that share a potential with node i.
Let Ψ_i be the set of potentials in Ψ involving x_i.
Compute

$$m_i(x_{S_i}) = \max_{x_i} \prod_{\varphi \in \Psi_i} \varphi(x_i \cup x_{S_i})$$
$$x_i^*(x_{S_i}) \in \arg\max_{x_i} \prod_{\varphi \in \Psi_i} \varphi(x_i \cup x_{S_i})$$

where ties are broken arbitarily.
Remove elements of Ψ_i from Ψ.
Add m_i to Ψ.
end
Produce $\mathbf{x}^*$ by traversing I in a reverse order

$$x_j^* = x_j^*(x_k^* : j < k \leq N).$$

Algorithm 1: The MAP Elimination Algorithm

As in the context of computing marginals, the overall cost of the MAP elimination algorithm is bounded above as

$$\begin{aligned} \text{overall cost} &\leq |\mathcal{C}| \sum_i |\mathcal{X}|^{|S_i|+1} \\ &\leq N|\mathcal{C}||\mathcal{X}|^{\max_i |S_i|+1}. \end{aligned}$$

The MAP elimination algorithm and elimination algorithm for marginals are closely related and they both rely on the relationship between sums and products and max and products. Specifically, the key property was that

$$\max_{x,y} f(x)g(x,y) = \max_x \left(f(x) \max_y g(x,y) \right)$$
$$\sum_{x,y} f(x)g(x,y) = \sum_x \left(f(x) \sum_y g(x,y) \right).$$

6.438 Algorithms for Inference

Fall 2014