Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.438 ALGORITHMS FOR INFERENCE
Fall 2014

7 The Elimination Algorithm

Thus far, we have introduced directed, undirected, and factor graphs and discussed conversion between these different kinds of graphs. Today we begin our foray into algorithms for inference on graphical models. Recall from the first lecture that the two main inference tasks of interest are:

- **Calculating posterior beliefs.** We have some initial belief or *prior* $p_{\mathsf{x}}(\cdot)$ on an unknown quantity of interest x. We observe $\mathsf{y} = y$, which is related to x via a likelihood model $p_{\mathsf{y}|\mathsf{x}}(y|\cdot)$. We want to compute distribution $p_{\mathsf{x}|\mathsf{y}}(\cdot|y)$.

- **Calculating maximum a posterior estimates.** We want to compute a most probable configuration $\hat{x} \in \arg\max_x p_{\mathsf{x}|\mathsf{y}}(x|y)$.

We look at calculating posterior beliefs today in the context of undirected graphical models. For concreteness, consider a collection of random variables $\mathsf{x}_1, \ldots, \mathsf{x}_N$, each taking on values from alphabet $\mathcal{X}$ and with their joint distribution represented by an undirected graph $\mathcal{G}$ and factoring as

$$p_{\mathbf{x}}(\mathbf{x}) = \frac{1}{Z} \prod_{c \in \mathcal{C}} \varphi_c(x_c),$$

where $\mathcal{C}$ is the set of maximal cliques of $\mathcal{G}$ and Z is the partition function. Marginalization is useful because, for example, we may be interested in learning about x_1 based on observing x_N, i.e., we want the posterior belief $p_{\mathsf{x}_1|\mathsf{x}_N}(\cdot|x_N)$. Calculating this posterior belief requires us to marginalize out $\mathsf{x}_2, \ldots, \mathsf{x}_{N-1}$. More generally, we want to be able to compute

$$p_{\mathbf{x}_{\mathcal{A}}|\mathbf{x}_{\mathcal{B}}}(\mathbf{x}_{\mathcal{A}}|\mathbf{x}_{\mathcal{B}}) = \frac{p_{\mathbf{x}_{\mathcal{A}},\mathbf{x}_{\mathcal{B}}}(\mathbf{x}_{\mathcal{A}}, \mathbf{x}_{\mathcal{B}})}{p_{\mathbf{x}_{\mathcal{B}}}(\mathbf{x}_{\mathcal{B}})}$$

for any disjoint pair of subsets $\mathcal{A}, \mathcal{B} \subset \{1, 2, \ldots, N\}$, $\mathcal{A} \cap \mathcal{B} = \varnothing$, where $\mathcal{A} \cup \mathcal{B}$ may not consist of all nodes in the graph.

In this lecture, we discuss the Elimination Algorithm for doing marginalization. This algorithm works on all undirected graphs and provides an exact solution, albeit possibly with high computational complexity. The analogous algorithm for DAG's is similar; alternatively, we could convert a DAG into an undirected graph via moralization and then apply the Elimination algorithm. We first look at marginalization when there are no observations and then explain how observations can be incorporated into the algorithm to yield posterior beliefs.

7.1 Intuition for the Elimination Algorithm

We first develop some intuition for what the Elimination Algorithm does via the following example graph on random variables $x_1, x_2, \ldots, x_5 \in \mathcal{X}$:

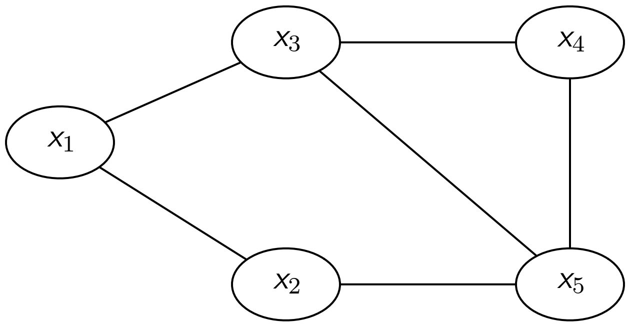


Figure 1: Our running example for today's lecture.

The corresponding probability distribution factors as:

$$p_{\mathsf{x}}(\mathbf{x}) \propto \varphi_{12}(x_1, x_2)\varphi_{13}(x_1, x_3)\varphi_{25}(x_2, x_5)\varphi_{345}(x_3, x_4, x_5), \tag{1}$$

where the φ's are the potential functions for the maximal cliques in the graph.

Suppose we want marginal $p_{\mathsf{x}_1}(\cdot)$. The naive brute-force approach is to first compute the joint probability table over all the random variables and then sum out $\mathsf{x}_2, \ldots, \mathsf{x}_5$:

$$p_{\mathsf{x}_1}(x_1) = \sum_{x_2, x_3, x_4, x_5 \in \mathcal{X}} p_{\mathsf{x}}(\mathbf{x}), \tag{2}$$

which requires $O(|\mathcal{X}|^5)$ operations.

Can we do better? Observe that by plugging in factorization (1) into equation (2), we obtain

$$\begin{aligned} p_{\mathsf{x}_1}(x_1) &= \sum_{x_2, x_3, x_4, x_5 \in \mathcal{X}} p_{\mathsf{x}}(\mathbf{x}) \\ &\propto \sum_{x_2, x_3, x_4, x_5 \in \mathcal{X}} \varphi_{12}(x_1, x_2)\varphi_{13}(x_1, x_3)\varphi_{25}(x_2, x_5)\varphi_{345}(x_3, x_4, x_5). \end{aligned}$$

Notice that choosing which order we sum out variables could make a difference in computational complexity!

For example, consider an "elimination ordering" $(5, 4, 3, 2, 1)$ where we sum out x_5 first, x_4 second, and so forth until we get to x_1, which we do not sum out. Then, we can push around some sums:

$$\begin{aligned}
p_{\mathsf{x}_1}(x_1) &\propto \sum_{x_2,x_3,x_4,x_5\in\mathcal{X}} \varphi_{12}(x_1,x_2)\varphi_{13}(x_1,x_3)\varphi_{25}(x_2,x_5)\varphi_{345}(x_3,x_4,x_5) \\
&= \sum_{x_2,x_3,x_4\in\mathcal{X}} \varphi_{12}(x_1,x_2)\varphi_{13}(x_1,x_3) \underbrace{\sum_{x_5\in\mathcal{X}} \varphi_{25}(x_2,x_5)\varphi_{345}(x_3,x_4,x_5)}_{\triangleq m_5(x_2,x_3,x_4)} \\
&= \sum_{x_2,x_3,x_4\in\mathcal{X}} \varphi_{12}(x_1,x_2)\varphi_{13}(x_1,x_3) m_5(x_2,x_3,x_4) \\
&= \sum_{x_2,x_3\in\mathcal{X}} \varphi_{12}(x_1,x_2)\varphi_{13}(x_1,x_3) \underbrace{\sum_{x_4\in\mathcal{X}} m_5(x_2,x_3,x_4)}_{\triangleq m_4(x_2,x_3)} \\
&= \sum_{x_2,x_3\in\mathcal{X}} \varphi_{12}(x_1,x_2)\varphi_{13}(x_1,x_3) m_4(x_2,x_3) \\
&= \sum_{x_2\in\mathcal{X}} \varphi_{12}(x_1,x_2) \underbrace{\sum_{x_3\in\mathcal{X}} \varphi_{13}(x_1,x_3) m_4(x_2,x_3)}_{\triangleq m_3(x_2)} \\
&= \sum_{x_2\in\mathcal{X}} \varphi_{12}(x_1,x_2) m_3(x_2) \\
&\triangleq m_2(x_1).
\end{aligned}$$

What happened here is that we computed intermediate tables m_5, m_4, m_3, and m_2. At the very end, to compute the marginal on x_1, we just normalize m_2 to get $p_{\mathsf{x}_1}(x_1)$:

$$p_{\mathsf{x}_1}(x_1) = \frac{m_2(x_1)}{\sum_{x'\in\mathcal{X}} m_2(x')}.$$

The above procedure is the Elimination Algorithm for our specific example graph using elimination ordering $(5,4,3,2,1)$. Note that the intermediate tables are given the letter m to suggest that these can be interpreted as messages. For example, $m_5(x_2,x_3,x_4)$ can be thought of as all the information x_5 has to give to its neighbors $\mathsf{x}_2, \mathsf{x}_3$, and x_4. Later on, we will see more of these so-called "message-passing" algorithms.

What is the computational complexity? Intuitively, this depends on the elimination ordering. For the elimination ordering $(5,4,3,2,1)$, note that the most expensive summation carried out was in computing m_5, which required forming a table over four variables and summing out x_5. This incurs a cost of $O(|\mathcal{X}|^4)$. Each of the other summations was cheaper, so the overall computational complexity is $O(|\mathcal{X}|^4)$, an improvement over the naive brute-force approach's complexity of $O(|\mathcal{X}|^5)$. But can we do better? In fact, we can! Think about what happens if we use elimination ordering $(4,5,3,2,1)$.

What about if we have some observations? For example, let's say that we observed $x_3 = a$. Then the above procedure still works, provided that we force the value of x_3 to be a and not sum over x_3. The normalization step at the end ensures that we have a valid conditional probability distribution, corresponding to the posterior belief for x_1 given $x_3 = a$. Another way to see this is to note that observing $x_3 = a$ is equivalent to placing a delta-function singleton potential on x_3: $\varphi_3(x_3) = 1$ if $x_3 = a$ and 0 otherwise–then running the Elimination Algorithm as usual gives us the posterior of x_1 given x_3.

7.2 The Elimination Algorithm

As we saw in our example, the Elimination Algorithm is just about marginalizing out variables in a particular order, at each step summing only over the relevant potentials and messages.

In general, the key idea is to maintain a list of "active potentials." This list starts off as the collection of all potential functions for our graph, including possibly the delta-function singleton potentials on observed nodes. Each time we eliminate a variable by summing it out, at least one potential function gets removed from the list of active potentials and a new message (which acts as a potential) gets added to the list. In the earlier example, upon eliminating x_5, potentials φ_{25} and φ_{345} were removed from the list of active potentials whereas m_5 was added to the list. With this bookkeeping in mind, we present the Elimination Algorithm:

```
**Input**: Potentials $\varphi_c$ for $c \in \mathcal{C}$, subset $\mathcal{A}$ to compute marginal $p_{\mathsf{x}_\mathcal{A}}(\cdot)$ over, and
           an elimination ordering $I$
**Output**: Marginal $p_{\mathsf{x}_\mathcal{A}}(\cdot)$
Initialize active potentials $\Psi$ to be the set of input potentials.
**for** *node $i$ in $I$ that is not in $A$* **do**
    Let $S_i$ be the set of all nodes (not including $i$) that share a potential with
    node $i$.
    Let $\Psi_i$ be the set of potentials in $\Psi$ involving $\mathsf{x}_i$.
    Compute
        $$m_i(x_{S_i}) = \sum_{x_i} \prod_{\varphi \in \Psi_i} \varphi(x_i \cup x_{S_i}).$$
    Remove elements of $\Psi_i$ from $\Psi$.
    Add $m_i$ to $\Psi$.
**end**
Normalize
    $$p_{\mathsf{x}_\mathcal{A}}(x_\mathcal{A}) \propto \prod_{\varphi \in \Psi} \varphi(x_\mathcal{A}).$$
```

Algorithm 1: The Elimination Algorithm

Let's analyze the algorithm's complexity. At each step i, we have a table of size $|\mathcal{X}|^{S_i}$. To fill in each element, we sum $|\mathcal{X}|$ terms and multiply $|\Psi_i|$ terms. This means that step i, which results in computing message m_i requires $O(|\mathcal{X}|^{S_i} \cdot |\mathcal{X}||\Psi_i|) = O(|\Psi_i| \cdot |\mathcal{X}|^{|S_i|+1})$ operations. Thus, the total complexity is

$$\sum_i O(|\Psi_i| \cdot |\mathcal{X}|^{|S_i|+1}).$$

We can upper-bound $|\Psi_i|$ with $|\mathcal{C}|$ and $|S_i|$ with $\max_i |S_i|$ to obtain

$$\sum_i O(|\Psi_i| \cdot |\mathcal{X}|^{|S_i|+1}) = \sum_i O(|\mathcal{C}| \cdot |\mathcal{X}|^{\max_i |S_i|+1}) = O(N \cdot |\mathcal{C}| \cdot |\mathcal{X}|^{\max_i |S_i|+1}),$$

recalling that big-O is an upper bound. Note that the quantity $\max_i |S_i|$ dictates how efficient the Elimination Algorithm will be. We next look a way to compute $\max_i |S_i|$.

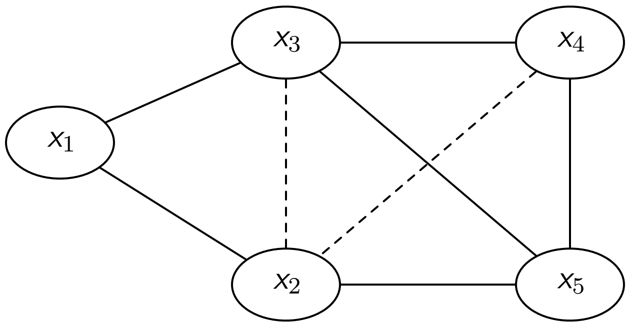
7.3 Reconstituted Graph

To compute $\max_i |S_i|$, we will look at what new edges we introduced during marginalization. Returning to our running example graph from Figure 1 and using elimination ordering $(5, 4, 3, 2, 1)$, upon summing out x_5, we are left with potentials $\varphi_{12}, \varphi_{13}, m_5(x_2, x_3, x_4)$, which can be viewed as a new graph where the m_5 term couples $\mathsf{x}_2, \mathsf{x}_3$, and x_4, i.e., effectively we have added edges $(\mathsf{x}_2, \mathsf{x}_3)$ and $(\mathsf{x}_2, \mathsf{x}_4)$.

In general, when eliminating a node, the node "sends" a message that couples all its neighbors into a clique. With this analysis, in our running example:

- After eliminating x_5: make $\{x_2, x_3, x_4\}$ into a clique (add edges (x_2, x_3) and (x_2, x_4))
- After eliminating x_4: make $\{x_2, x_3\}$ into a clique (already done, so no edge added)
- After eliminating x_3: make $\{x_1, x_2\}$ into a clique (already done, so no edge added)
- After eliminating x_2: nothing left to do

By introducing the added edges into our original graph from Figure 1, we arrive at what is called the *reconstituted graph*:

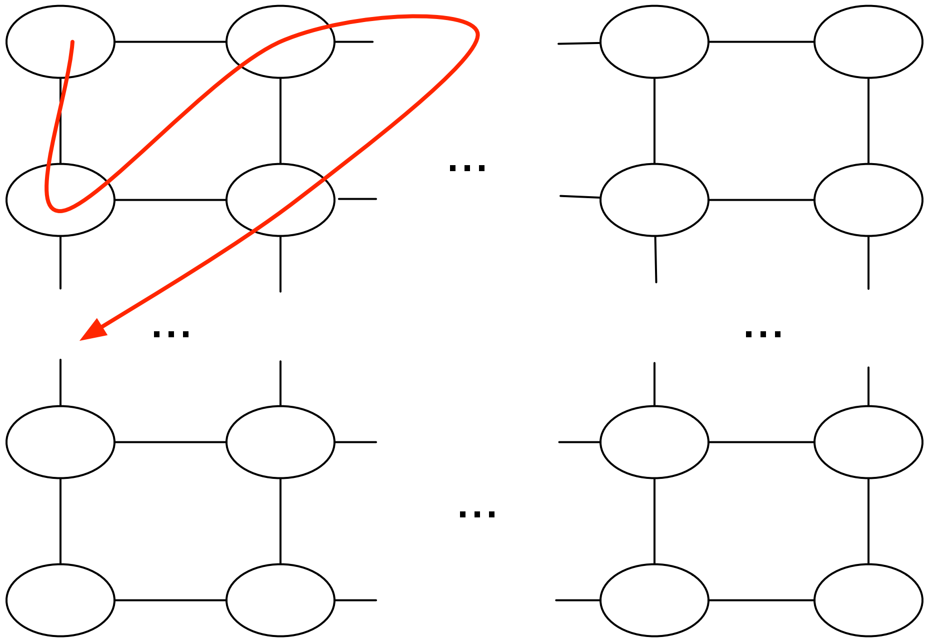


Here, we denote the added edges via dotted lines.

The reconstituted graph has two key properties. First, the largest clique size in the reconstituted graph is $\max_i |S_i|+1$ (note that S_i does not include node i) from our complexity analysis earlier. Second, the reconstituted graph is always chordal, which means that if we want to make a graph chordal, we could just run the Elimination Algorithm and then construct the reconstituted graph. Note that the reconstituted graph depends on the elimination ordering. To see this, try the elimination ordering $(4, 5, 3, 2, 1)$. The minimum time complexity achieveable using the Elimination Algorithm for a graph is thus given by $\min_I \max_i |S_i|$, referred to as the *treewidth* of the graph.

7.4 Grid graph

We end with a neat result, which will be stated without proof. Consider the grid graph over $N = n^2$ nodes as shown below:



Brute-force marginalization requires $O(|\mathcal{X}|^N)$ operations. However, it is possible to achieve an elimination ordering where $\max_i |S_i| = \sqrt{N}$, using the zig-zag pattern denoted by the arrow above. In fact, any planar graph has an elimination order such that $\max_i |S_i| = O(\sqrt{N})$.