COS513:FOUNDATIONS OF PROBABILISTIC MODELS LECTURE 4

JELENA BRADIC, MIHAI CUCURINGU

1. SUM-PRODUCT ALGORITHM

Why do we care?

(i) Tress are important because many well known graphical models can be represented as trees
(ii) This is the basis for the junction tree algorithm
(iii) It is also the basis of an approximate inference algorithm

2. TREES.

Undirected tree is a graph where there is only one path between any two nodes. Directed tree is a graph whose moralization is an undirected tree.

Examples:

The graph on the figure 3 represent an example of a graph whose moralization doesn't have the property of undirected trees (i.e. one path between every two nodes) so it is not a directed tree.

Undirected tree: Under the undirected tree model the joint distribution has the following representation i.e. parametrization:

$$p(x) = \frac{1}{Z} \prod_{v \in V} \Psi(x_v) \prod_{(i,j) \in \mathcal{E}} \Psi(x_i, x_j) \tag{1}$$

FIGURE 1. Undirected Tree

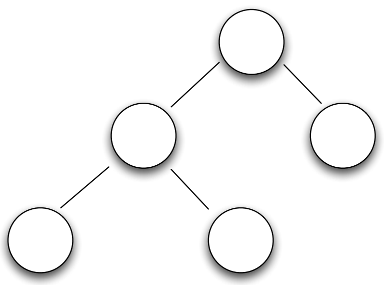


FIGURE 2. Directed Tree

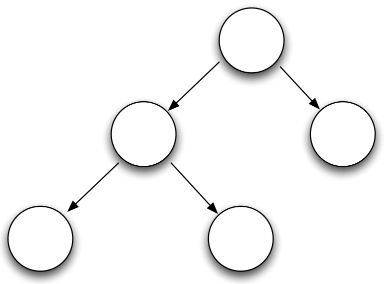
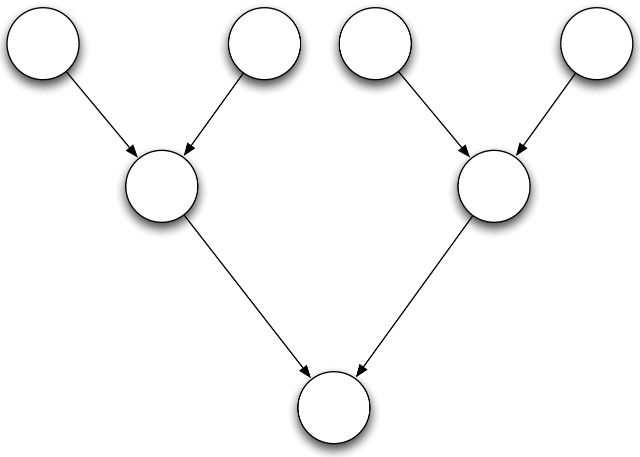


FIGURE 3. Not a Directed Tree



where $\Psi(x_v)$ are defined as singleton potentials and $\Psi(x_i, x_j)$ are defined as pairwise potentials and where Z is a normalizing constant set to make the whole expression on the right hand side of equation (2) sum to 1.

Directed tree: Under the directed tree model the joint now takes the following form:

$$p(x) = p(x_r) \prod_{(i,j)\in\mathcal{E}} p(x_j|x_i). \tag{2}$$

Note that one can parameterize a directed tree as un undirected tree, using the substitutions below, with the restriction that the potentials Ψ are positive:

(i) $\Psi(x_r) = p(x_r)$
(ii) $\Psi(x_i) = 1, \; i \neq r$
(iii) $\Psi(x_i, x_j) = p(x_j|x_i)$

Expressing the joint probability for a directed tree in the undirected form of (2) makes $Z = 1$. From now on we will restrict ourselves to the case of undirected trees.

Also, note that we do not make any special distinction between the unconditional and conditional case, because the two parametrizations can be made formally identicalby by using “evidence potentials“ (the δ-functions $\delta(x_i, \bar{x}_i)$ used to capture conditioning)

3. ELIMINATE.

Recall the Elimination Algorithm:

(i) Pick an ordering such that the query is last
(ii) Put probabilities(potentials) on the active list together with the delta δ functions for evidence
(iii) $(\forall i)$:
    (a) Eliminate each node by taking products of potentials on active list using x_i
    (b) Sum out x_i to form $m_i(S_i)$
    (c) Put $m_i(S_i)$ back to the active list

Note that the evidence does not require any change in the setup and that we have:

$$\Psi^E(x_i) = \begin{cases} \Psi(x_i)\delta(x_i, \bar{x}_i), & \text{if } i \in E \\ \Psi(x_i), & \text{o.w.} \end{cases} .$$

This shows that we needed to use singleton potentials separately from the pairwise one.

On an undirected tree, chose the following ordering, with f denoting the query node and E the set of evidence nodes:

- treat f as the root (change the view)
- ”direct” all the edges away from f
- set an order such that each node is being ated after its children

Note that for any querry, this gives an efficient inference algorithm, the reconstitued graphs of this tree are the same, with the complexity given by the size of the largest clique (i.e. 2)

Example: As an example, consider the following graph on Figure 4. Building a reconstituted graph is an iterative process, at each step a node is eliminated and its parents get connected. If we start from the root of the tree, in this example the tree stays unchanged and hence it is easy to work with.

Note that the size of the maximal clique in this particular example is 2. This leads us to having $O(k^2)$ is every marginal can take k discrete possible values.

FIGURE 4. Reconstituion of a tree is the same tree

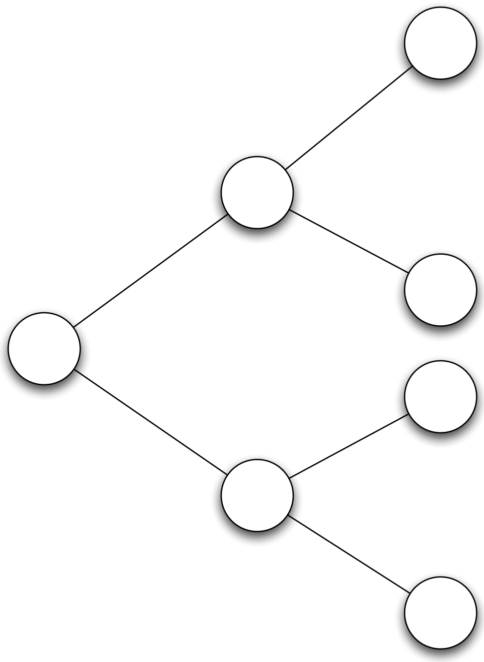
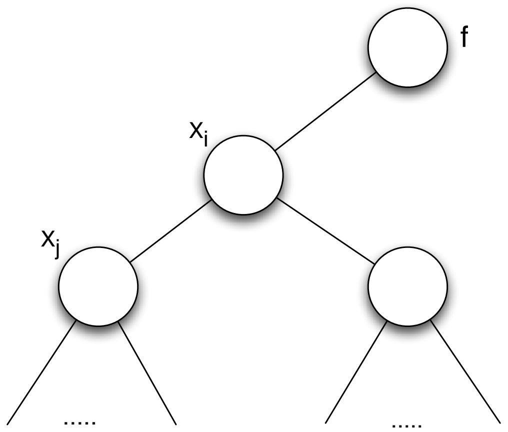


FIGURE 5. Reconstituion of a tree is the same tree



4. ELIMINATION STEP

Consider i and j such that i is closer to the root than j is like in the Figure 5. The factor created when x_i is eliminated will be a product of the following functions:

(i) $\Psi(x_j)$

(i) $\Psi(x_i, x_j)$, as it contains x_i as one of its arguments
(i) No function containing x_k-descendent of x_j can appear because it has been eliminated before x_j by the ordering
(i) No function containing a node x_l which is not in the subtree of x_j, since there is no edge between x_l and any descendant of x_j.

Hence, once x_j is summed out, the intermediate factor is only a function of x_i. Denote this factor by

$$m_{ji}(x_i) \quad \text{message from } j \to i.$$

Then

$$(*) \qquad m_{ji}(x_i) = \sum_{x_j} \Psi(x_j)\Psi(x_i, x_j) \prod_{k \in N(j)\setminus i} m_{kj}(x_j)$$

where $N(j)$ denotes the neighbors of j. At the top of the tree we will have:

$$(**) \qquad p(x_f|\bar{x}_E) \propto \Psi(x_f) \prod_{k \in N(f)\setminus i} m_{kf}(x_f)$$

since the root has no parents and thus all pairwise potentials disappear.

The eliminate algorithm on trees now becomes equivalent to solving a system of equations:

$$\textbf{eliminate algorithm on trees} = \{(*), (**)\}$$

Hence we can guess right away that the solution of this system of equations will be based on some dynamic programming techniques.

4.1. **Example:** Lets consider the following structural graph as presented in Figure6.

Suppose we want to calculate the marginal $p(x_1)$. Then elimination order becomes $4-3-2-1$ so we get to compute $m_{42}(x_2)$, $m_{32}(x_2)$, $m_{21}(x_1)$:

$$\begin{aligned}
m_{42}(x_2) &= \sum_{x_4} \Psi(x_4)\Psi(x_2, x_4) \\
m_{32}(x_2) &= \sum_{x_3} \Psi(x_4)\Psi(x_2, x_3) \\
m_{21}(x_1) &= \sum_{x_2} m_{32}(x_2)m_{42}(x_2)\Psi(x_2)\Psi(x_1, x_2) \\
p(x_1|x_E) &\propto \Psi(x_1)m_{21}(x_1)
\end{aligned}$$

What about computing the marginal $p(x_2)$? We use the same graph but we change the root node: Figure7. If the elimination is $3-4-1-2$, we need to compute the following messages:

FIGURE 6. $p(x_1)$

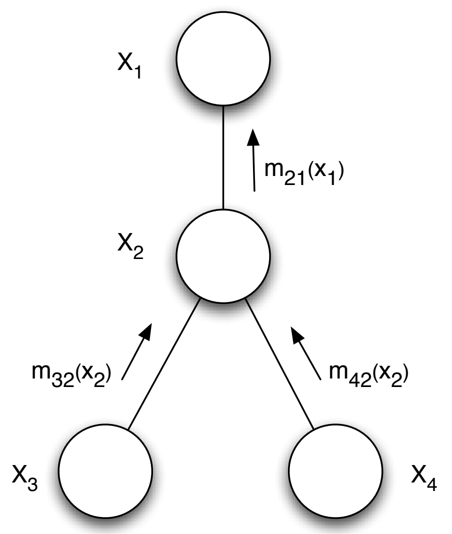
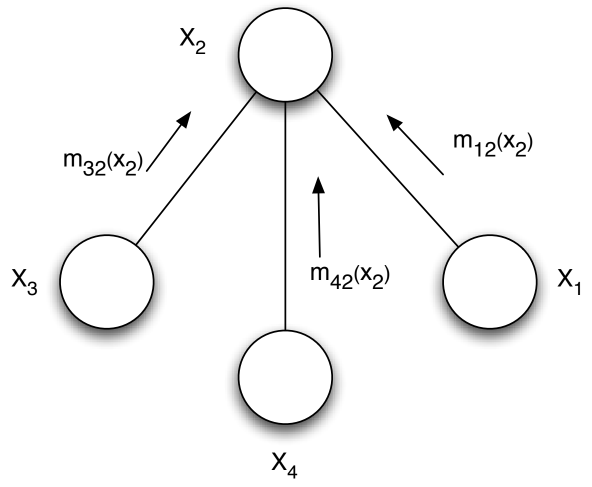
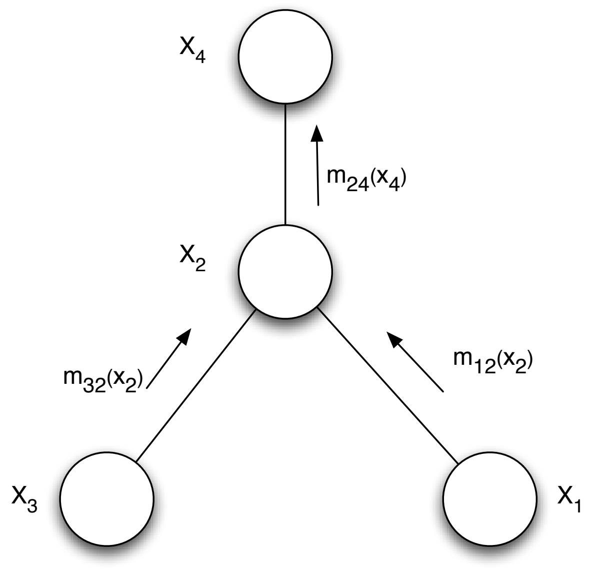


FIGURE 7. $p(x_2)$



$$\begin{aligned} m_{42}(x_2) &= \sum_{x_4} \Psi(x_4)\Psi(x_2,x_4) \\ m_{32}(x_2) &= \sum_{x_3} \Psi(x_4)\Psi(x_2,x_3) \\ m_{12}(x_2) &= \sum_{x_1} \Psi(x_1)\Psi(x_2,x_1) \end{aligned}$$

FIGURE 8. $p(x_4)$



Notice that $m_{42}(x_2)$ and $m_{32}(x_2)$ are exactly the same messages we used when computing $p(x_1)$! When computing the marginal $p(x_4)$ (see Figure 8. for the shape of the tree), again we need to redo some of the computations from above.

The key insight begind the SUM-PRODUCT algorithm is that messages can be "reused". At the cost of computing all such possible messages in the tree, we almost get for free all marginals (that we would otherwise get by considering a different elimination order for each marginal).

5. SUM-PRODUCT ALGORITHM

Sum-product algorithm is based on **(*)**, **(**)** and the following protocol:

Message Passing Protocol: *A node can send a message to a neighbor only and only when it has received messages from all its other neighbors.*

Note that such a protocal can be implemented either using a parallel algorithm or following a sequential implementation (in which messages are computed according to a "schedule").

Remark: To test whether a graph is a tree (i.e. it is *connected* and *acyclic*), simply go through each nod (using BFS or DFS) and check if each node is visited exactly once.

FIGURE 9. Message passing protocol