Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.438 Algorithms for Inference
Fall 2020

12 The Sum-Product Algorithm (BP)

In the last lecture, we introduced the elimination algorithm for performing marginalization of distributions defined over undirected graphical models. We saw that the elimination algorithm always produces an exact solution and can be applied to any graph structure. Furthermore, it is not entirely naïve, in that it makes use of the graph structure to save computation. For instance, recall that for the square lattice graph, the computational complexity of marginalization via the elimination algorithm is $O(|\mathcal{X}|^{\sqrt{N}})$, while brute force computation requires $O(|\mathcal{X}|^N)$ operations.

However, if we wish to retrieve the marginal distributions for multiple different variables, we would need to run the elimination algorithm from scratch each time. This would entail a lot of redundant computation. It turns out that we can recast the computations in the elimination algorithm as messages passed between nodes in the network, and that these messages can be reused for different marginalization queries. The resulting algorithm is known as the *sum-product algorithm* or *belief propagation*.

To introduce the sum-product algorithm, we examine efficient implementation of marginalization in the context of trees. Later we will develop how to adapt and apply the algorithm to broader classes of graphical models.

12.1 The Elimination Algorithm on (Undirected) Trees

Recall that a graph $\mathcal{G}$ is a *tree* if any two nodes are connected by exactly one path. This definition implies that all tree graphs have exactly $N-1$ edges and no cycles.

Throughout this lecture, we will use the recurring example of the tree graphical model shown in Fig. 1. Suppose we wish to obtain the marginal distribution p_{x_1} using the elimination algorithm. The efficiency of the algorithm depends on the elimination ordering chosen. For instance, if we choose the suboptimal ordering $(2, 4, 5, 3, 1)$, the elimination algorithm produces the sequence of graph structures shown in Fig. 2. After the first step, all of the neighbors of x_2 are connected, resulting in a table of size $|\mathcal{X}|^3$. For a more drastic example of a suboptimal ordering, consider the star graph shown in Fig. 3. If we eliminate x_1 first, the resulting graph is a fully connected graph over the remaining variables, resulting in a table of size $|\mathcal{X}|^{N-1}$. This is clearly undesirable.

Fortunately, for tree graphs, it is easy to find an ordering which does not add edges. Recall that, in the context of the elimination algorithm for a generic graph, a seemingly good heuristic was to successively choose a node that has smallest degree to eliminate from the graph. Indeed, the nodes with smallest degree in a tree graph are *leaves*. One can check that every tree graph has at least two leaf nodes: if not, then all nodes have degree 2 or more, which means that the number of edges is at least N.

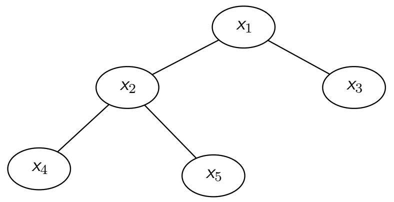

Figure 1: A tree-structured graphical model which serves as a recurring example throughout this lecture.

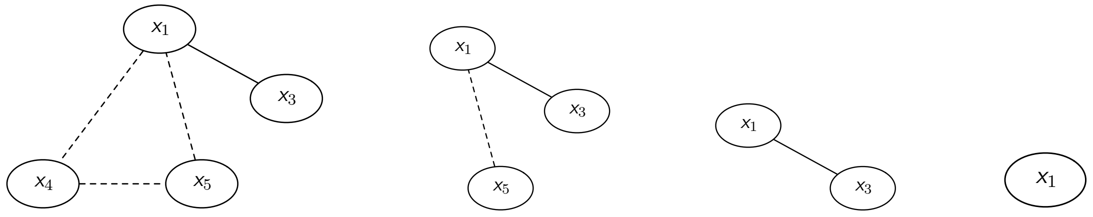

Figure 2: The sequence of graph structures obtained from the elimination algorithm on the graph from Fig. 1 using the suboptimal ordering $(2, 4, 5, 3, 1)$.

But by definition, a tree has $N - 1$ edges. In the case of the graph from Fig. 1, one such ordering is $(4, 5, 3, 2, 1)$. The resulting graphs and messages are shown in Fig. 4.

In a tree graph, the maximal cliques are exactly the edges. Therefore, by the Hammersley-Clifford Theorem, if we assume the joint distribution is strictly positive, we can represent it (up to normalization) as a product of potentials $\psi_{ij}(x_i, x_j)$ for each edge (i, j). However, it is often convenient to include unary potentials as well, so we will assume a redundant representation with unary potentials $\phi_i(x_i)$ for each variable x_i. In other words, we assume the factorization

$$p_{\mathsf{x}}(\mathbf{x}) = \frac{1}{Z} \prod_{i \in \mathcal{V}} \phi_i(x_i) \prod_{(i,j) \in \mathcal{E}} \psi_{ij}(x_i, x_j). \tag{1}$$

The messages produced in the course of the algorithm are:

$$\begin{aligned} m_4(x_2) &= \sum_{x_4} \phi_4(x_4)\, \psi_{24}(x_2, x_4) \\ m_5(x_2) &= \sum_{x_5} \phi_5(x_5)\, \psi_{25}(x_2, x_5) \\ m_3(x_1) &= \sum_{x_3} \phi_3(x_3)\, \psi_{13}(x_1, x_3) \\ m_2(x_1) &= \sum_{x_2} \phi_2(x_2)\, \psi_{12}(x_1, x_2)\, m_4(x_2)\, m_5(x_2) \end{aligned} \tag{2}$$

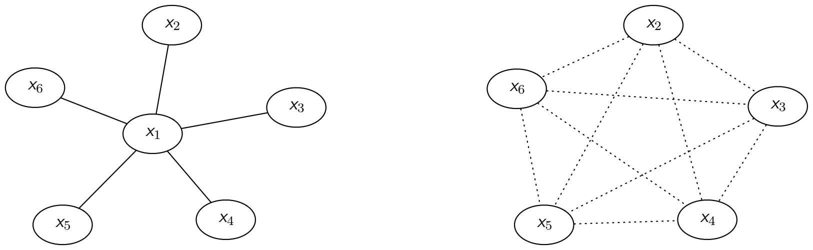

Figure 3: A star-shaped graph and the resulting graph after eliminating variable x_1.

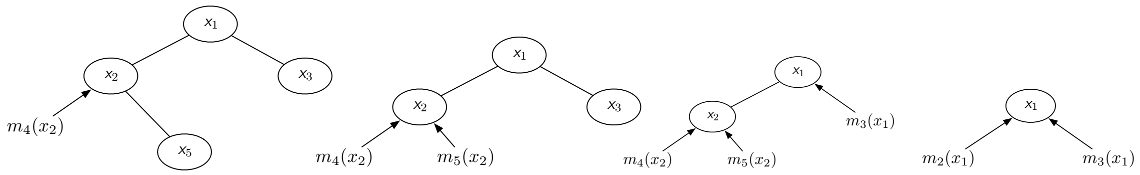

Figure 4: The sequence of graph structures and messages obtained from the elimination algorithm on the graph from Fig. 1 using an optimal ordering $(4, 5, 3, 2, 1)$.

Finally, we obtain the marginal distribution over x_1 by multiplying the incoming messages with its unary potential, and then marginalizing. In particular,

$$p_{x_1}(x_1) \propto \phi_1(x_1)\, m_2(x_1)\, m_3(x_1). \tag{3}$$

What is the computational complexity of this algorithm? Each of the messages produced has $|\mathcal{X}|$ values, and computing each value requires summing over $|\mathcal{X}|$ terms. Since this must be done for each of the $N-1$ edges in the graph, the total complexity is $O(N|\mathcal{X}|^2)$, i.e. linear in the graph size and quadratic in the alphabet size.[1]

[1]Note that this analysis does not include the time for computing the products in each of the messages. A naïve implementation would, in fact, have higher computational complexity. However, with the proper bookkeeping, we can reuse computations between messages to ensure that the total complexity is $O(N|\mathcal{X}|^2)$.

12.2 The Sum-Product Algorithm on Trees

Returning to Fig. 1, suppose we want to compute the marginal for another variable x_3. If we use the elimination ordering (5, 4, 2, 1, 3), the resulting messages are:

$$\begin{aligned} m_5(x_2) &= \sum_{x_5} \phi_5(x_5)\, \psi_{25}(x_2, x_5) \\ m_4(x_2) &= \sum_{x_4} \phi_4(x_4)\, \psi_{24}(x_2, x_4) \\ m_2(x_1) &= \sum_{x_2} \phi_2(x_2)\, \psi_{12}(x_1, x_2)\, m_4(x_2)\, m_5(x_2) \\ m_1(x_3) &= \sum_{x_1} \phi_1(x_1)\, \psi_{13}(x_1, x_3)\, m_2(x_1) \end{aligned} \tag{4}$$

Notice that the first three messages m_1, m_4, and m_2 are all strictly identical to the corresponding messages from the previous computation. The only new message to be computed is $m_1(x_3)$, as shown in Fig. 5.

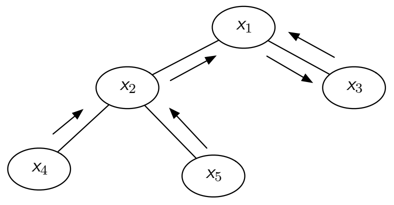

Figure 5: All messages required to compute the marginals over $p_{\mathsf{x}_1}(x_1)$ and $p_{\mathsf{x}_3}(x_3)$ for the graph in Fig. 1.

As this example suggests, we can obtain the marginals for every node in the graph by computing $2(N-1)$ messages, one for each direction along each edge. As shown in Fig. 6, when computing the message $m_{i\to j}(x_j)$, we need the incoming messages $m_{k\to i}(x_i)$ for its other neighbors $k \in \mathcal{N}(i)\backslash\{j\}$, where we introduce as convenient notation that $\mathcal{N}(i)$ is the set of nodes that are neighbors of node i, for any $i \in \mathcal{V}$. Therefore, we need to choose an ordering over messages such that the prerequisites are available at each step. One way to do this is through the following two-step procedure.

- Choose an arbitrary node i as the root, and generate messages going towards it using the elimination algorithm described in Section 12.1.
- Compute the remaining messages, working outwards from the root.

We now combine these insights into the *sum-product algorithm on trees*. Messages are computed in the order given above using the rule:

$$m_{i\to j}(x_j) = \sum_{x_i} \phi_i(x_i)\, \psi_{ij}(x_i, x_j) \prod_{k\in\mathcal{N}(i)\backslash\{j\}} m_{k\to i}(x_i). \tag{5}$$

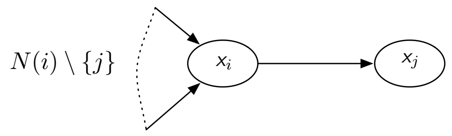

Figure 6: The message $m_i(x_i)$ depends on each of the incoming messages $m_k(x_i)$ for x_i's other neighbors $\mathcal{N}(i)\backslash\{j\}$.

Note that, in order to disambiguate messages sent from node i, we explicitly write $m_{i\to j}(x_j)$ rather than simply $m_i(x_j)$. Then, the marginal for each variable is obtained using the formula:

$$p_{x_i}(x_i) \propto \phi_i(x_i) \prod_{j\in\mathcal{N}(i)} m_{j\to i}(x_i). \tag{6}$$

We note that the sum-product algorithm can also be viewed as a dynamic programming algorithm for computing marginals. This view will become clearer when we discuss hidden Markov models.

12.3 The Parallel Sum-Product Algorithm

The sum-product algorithm as described in Section 12.2 is inherently sequential: the messages must be computed in sequence to ensure that the prerequisites are available at each step. However, the algorithm was described in terms of simple, local operations corresponding to different variables, which suggests that it might be parallelizable. Indeed, if the updates (5) are repeatedly applied in parallel, it is possible to show that the messages will eventually converge to their correct values. More precisely, letting $m_i^t(x_j)$ denote the messages at time step t, we apply the following procedure:

1. Initialize all messages $m^0_{i\to j}(x_j) = 1$ for all $(i,j) \in E$.

2. Iteratively apply the update

$$m^{t+1}_{i\to j}(x_j) = \sum_{x_i} \phi_i(x_i)\, \psi_{ij}(x_i, x_j) \prod_{k\in\mathcal{N}(i)\backslash\{j\}} m^t_{k\to i}(x_i) \tag{7}$$

Intuitively, this procedure resembles fixed point algorithms for solving equations of the form $f(x) = x$. Fixed point algorithms choose some initial value x_0 and then iteratively apply the update $x^{t+1} = f(x^t)$. We can view (5) not as an update rule, but as a set of equations to be satisfied. The rule (7) can be viewed as a fixed point update for (5). Indeed,

Theorem 1. *Given a tree graph $\mathcal{G}$ with diameter d, the messages (5) of parallel sum product algorithm converge after $t \geq d$ iterations and they are correct.*

Note that this parallel procedure entails significant overhead: each iteration of the algorithm requires computing the messages associated with every edge. We saw in Section 12.1 that this requires $O\left(N|\mathcal{X}|^2\right)$ time. This is the price we pay for parallelism. Parallel sum-product is unlikely to pay off in practice unless the diameter of the tree is small. However, in a later lecture we will see that it naturally leads to *loopy belief propagation*, where the update rule (7) is applied to graphs that are not trees.

12.4 Efficient Implementation

In our complexity analysis from Section 12.1, we swept under the rug the details of exactly how the messages are computed. Consider the example of the star graph shown in Fig. 3, where this time we intelligently choose the center node x_1 as the root. When we compute the outgoing messages $m_{1\to j}(x_j)$, we must first multiply together all the incoming messages $m_{k\to 1}(x_1)$ and then sum them over all possible $x_1 \in \mathcal{X}$. Since there are $N-2$ of these, the product and summation requires $\Theta(|\mathcal{X}|^2 N)$ computations. This must be done for each of the $N-1$ outgoing messages, so these products contribute approximately $|\mathcal{X}|^2 N^2$ computations in total. This is quadratic in N, which is worse than the linear dependency we stated earlier. More generally, for a tree graph $\mathcal{G}$, these products require $O\left(|\mathcal{X}|^2 \sum_i d_i^2\right)$ time, where d_i is the degree (number of neighbors) of node i.

However, in parallel sum-product, we can share computation between the different messages by computing them simultaneously as follows:

1. Compute
$$\mu_i^t(x_i) = \left(\prod_{k\in\mathcal{N}(i)} m_{k\to i}^t(x_i)\right)\phi_i(x_i) \tag{8}$$

2. For all $j \in \mathcal{N}(i)$, compute
$$m_{i\to j}^{t+1}(x_j) = \sum_{x_i\in\mathcal{X}} \frac{\psi_i(x_i, x_j)\,\mu_i^t(x_i)}{m_{j\to i}^t(x_i)} \tag{9}$$

Using this algorithm, each update (8) can be computed in $O(|\mathcal{X}|d_i)$ time, so the cost per iteration is $O\left(|\mathcal{X}|\sum_i d_i\right) = O\left(|\mathcal{X}|N\right)$. Computing (9) requires summing over $|\mathcal{X}|$ different values. Therefore, the overall running time is $O\left(|\mathcal{X}|^2 N\right)$ per iteration, or $O\left(|\mathcal{X}|^2 N d\right)$ total where d is the diameter of the tree graph. A similar strategy can be applied to the sequential algorithm to achieve a running time of $O(|\mathcal{X}|^2 N)$.

12.5 Loopy Belief Propagation

We now shift our attention to general undirected graphical models (not necessarily trees) that have pairwise potentials with the following notation

$$p(x) = \frac{1}{Z} \prod_{(i,j)\in E} \psi_{ij}(x_i, x_j)$$

and study the resulting parallel sum-product (belief propagation) algorithm with iterates

$$m_{i\to j}^{t+1}(x_j) \propto \sum_{x_i} \psi_{ij}(x_i, x_j) \prod_{k\in\mathcal{N}(i)\setminus\{j\}} m_{k\to i}^{t}(x_i)$$

and final *beliefs*

$$b_i(x_i) \propto \prod_{k\in\mathcal{N}(i)} m_{k\to i}^{t}(x_i)\,. \tag{10}$$

If the graph is a tree, $b_i(x_i)$ is the true marginal $p_{\mathsf{x}_i}(x_i)$ at node i (as long as t is larger than the diameter of the tree). Unfortunately, in general graphical models that are not trees, the belief is not necessarily the marginal distribution. Nevertheless, one may hope that the belief gives an approximation of the true marginal distribution. For this reason we use separate notation for the belief computed using loopy BP (i.e., BP applied to graphs with cycles) as compared to the true marginal.

We will now give a probabilistic interpretation of what BP is doing. For now, this interpretation will yield intuition for why loops result in erroneous marginal computations and also allow us to develop the *computation tree* method for analyzing BP. Define

$$\widetilde{m}_{i\to j}^{t+1}(x_i) = \prod_{k\in\mathcal{N}(i)\setminus\{j\}} m_{k\to i}^{t}(x_i)$$

and then observe that

$$m_{i\to j}^{t+1}(x_j) \propto \sum_{x_i} \psi_{ij}(x_i, x_j)\widetilde{m}_{i\to j}^{t+1}(x_i)\,.$$

The tilde BP messages satisfy the update equations

$$\widetilde{m}_{i\to j}^{t+1}(x_i) \propto \prod_{k\in\mathcal{N}(i)\setminus j} \left(\sum_{x_k} \psi_{ik}(x_i, x_k)\widetilde{m}_{k\to i}^{t}(x_k)\right) \tag{11}$$

and yield belief (estimate of marginal) at node i given by

$$b_i^t(x_i) \propto \prod_{k\in\mathcal{N}(i)} \left(\sum_{x_k} \psi_{ik}(x_i, x_k)\widetilde{m}_{k\to i}^{t}(x_k)\right)\,. \tag{12}$$

Comparing $\widetilde{m}^{t+1}_{i\to j}(x_i)$ to $b_i(x_i)$ in (10), the former is exactly $b_i(x_i)$ but with $m^{t+1}_{j\to i}(x_i)$ removed. In a tree, we can interpret $\widetilde{m}_{i\to j}$ as the marginal of x_i in *a different graphical model*: the subtree including i when edge (i,j) is removed. This means that the recursion in (11) can be interpreted as a recursion for marginal probabilities of subtrees:

Lemma 1. *Let T be a tree with root i and let $\mu(x) \propto \prod_{kl\in\mathcal{E}(T)} \psi_{kl}(x_k,x_l)$ be a distribution on T. For any node j let $a(j)$ be the "ancestor" of j, i.e., the node at distance 1 from j along the path from j to i. Let $T(j)$ be the subtree of T rooted at j obtained by removing edge $(j,a(j))$ and let $\mu^{T(j)}_{j\to a}(x_j)$ be the marginal of x_j with respect to the model $\mu^{T(j)}_{j\to a}(x_{T(j)}) \propto \prod_{kl\in\mathcal{E}(T(j))} \psi_{kl}(x_k,x_l)$. Then $\mu^{T(j)}_{j\to a(j)}(x_j)$ satisfies precisely the BP recursion in* (11),

$$\mu^{T(j)}_{j\to a(j)}(x_j) \propto \prod_{k\in\mathcal{N}(j)\setminus\{a\}} \sum_{x_k} \psi_{kj}(x_k,x_j)\mu^{T(k)}_{k\to j}(x_k)\,.$$

The lemma will be useful later.

Why are loops a problem for BP? The message $\widetilde{m}^{t+1}_{i\to j}(x_i)$ can be thought of as i telling j "Based on the incoming messages from my other neighbors, if we were not interacting, then I think my marginal would be the following." The issue is that presence of loops can cause the message $\widetilde{m}_{i\to j}$ to impact future iterations of the same message. This is because the messages from j to its neighbors are impacted by i's message to j, and when there are loops this effect can propagate back to i.

We next introduce a tool called the *computation tree*, that lets us account for presence of loops in analyzing BP.

12.6 Computation Trees

Given a graph $G=(\mathcal{V},\mathcal{E})$, we form the computation tree $T_i(G)=(\mathcal{V}_i,\mathcal{E}_i)$ for G rooted at i by starting with root i and exploring the graph via non-reversing paths (see Fig 7). If the component of i in G is a tree, then the computation tree is finite, otherwise $T_i(G)$ is infinite. Note, that T_i is a "graph covering" as the mapping describing the correspondence of nodes, $\pi:\mathcal{V}_i\to\mathcal{V}$, is onto (i.e., every node in $\mathcal{V}$ is mapped to) and $(i,j)\in\mathcal{E}_i \iff (\pi(i),\pi(j))\in\mathcal{E}$.

We will associate to each edge $(j,k)\in\mathcal{E}_i$ the potential $\psi_{\pi(j),\pi(k)}$ in the original tree that goes with the edge $(\pi(j),\pi(k))$. It will often be easier to just write ψ_{jk} for this potential, with the π implicit.

Say that nodes are at level s if their distance to the root is s and let $T^{(t)}_i(G)$ denote the truncation of $T_i(G)$ to the nodes within the first $t+1$ levels. (Thus, in the case $t=0$ the children of node i are included.) Define $\partial T^{(t)}_i$ to be the nodes at the furthest level (i.e., level $t+1$) and introduce the *boundary conditions* (i.e., node potentials at the level $t+1$ nodes),

$$\{\widetilde{\eta}_l(x_l) : l\in\partial T^{(t)}_i(G)\}\,.$$

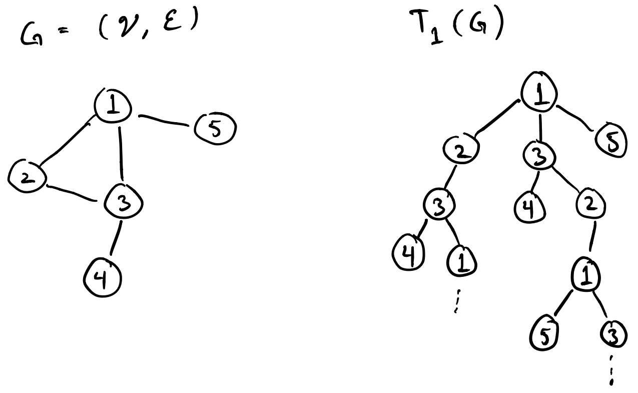

Figure 7: Example computation tree rooted at node 1.

Then we can define a new probability distribution over the tree with these boundary conditions according to

$$\mu^{(t,i)}(x) \propto \prod_{ij \in T_i^{(t)}} \psi_{ij}(x_i, x_j) \prod_{k \in \partial T_i^{(t)}} \widetilde{\eta}_{k \to a(k)}(x_k)\,. \tag{13}$$

Note that the probability distribution over our tree may have many copies of a given edge potential inherited from the original graphical model.

What the following theorem states is that the BP estimate in our loopy graph is the marginal in the computation tree with appropriate boundary conditions. This will be useful because we can then understand BP by understanding the probability measure on the computation tree.

Theorem 1. *Let $b_i^{(t)}(x_i)$ be the BP estimate* (12) *with respect to p after t iterations of the $\widetilde{m}$ messages on G. Let $a(l)$ denote the ancestor (parent) of l in the tree $T_i^{(t)}(G)$ (if l is a leaf then this is the unique neighbor of l). If the boundary condition on $T_i^{(t)}(G)$ is*

$$\widetilde{\eta}_k(x_k) = \widetilde{m}^0_{k \to a(k)}(x_k)$$

for each $k \in \partial T_i^{(t)}$, then for $t \geq 1$, $b_i^t(x_i) = \mu^{(t,i)}(x_i)$. In words, the belief at node i after t steps of BP is equal to the marginal at the root of the appropriate model on the computation tree $T_i^{(t)}(G)$.

Proof. Let j be at level $t+1-s$ on $T_i(G)$ and a be its parent. Let $T(j)$ be the subtree of $T_i(G)$ below j and $\mu^{T(j)}_{j \to a}(x_j)$ be the marginal of x_j with respect to the model on tree $T(j)$ (with boundary conditions on the leaves inherited from the full tree model). We will show by induction that for any $0 \leq s \leq t$,

$$\mu^{T(j)}_{j \to a}(x_j) = \widetilde{m}^s_{j \to a}(x_j)\,. \tag{14}$$

The base case, $s = 0$, follows from the choice of boundary. To see this, note that $T(j)$ consists of the single node j, since j is at level $t+1$ and is thus in $\partial T_i^{(t)}$. As specified in (13), the distribution $\mu^{T(j)}_{j \to a}(x_j) \propto \widetilde{\eta}_j(x_j) = \widetilde{m}^0_{j \to a(j)}(x_j)$ as required.

Now the inductive step. Assume the statement in (14) holds true for all nodes at level $t+1-s$ and that j is at level $(t+1)-(s+1) = t-s$. Note that j's children are at level $t+1-s$. Then

$$\begin{aligned}
\mu_{j\to a}^{T(j)}(x_j) &\propto \prod_{k\in\mathcal{N}(j)\setminus\{a\}} \sum_{x_k} \psi_{kj}(x_k, x_j)\mu_{k\to j}^{T(k)}(x_k) && \text{by Lemma 1} \\
&\propto \prod_{k\in\mathcal{N}(j)\setminus\{a\}} \sum_{x_k} \psi_{kj}(x_k, x_j)\widetilde{m}_{k\to a}^{s}(x_k) && \text{by inductive hypothesis} \\
&\propto \widetilde{m}_{j\to a}^{s+1}(x_j) && \text{by update equation (11) for } \widetilde{m}.
\end{aligned}$$

We can then repeat the same argument for the root. □