$$\begin{aligned}
\mu_\alpha(x_k) &= \sum_{x_{k-1}} \phi_{k-1,k}(x_{k-1}, x_k) \left(\sum_{x_{k-2}} \cdots \right) \\
&= \sum_{x_{k-1}} \phi_{k-1,k}(x_{k-1}, x_k) \cdot \mu_\alpha(x_{k-1})
\end{aligned}$$

The base case is

$$\mu_\alpha(x_1) = \mathbf{1} \ .$$

Now, for the *backward messages*:

Think of these as expressing a collective opinion, based on all the nodes with indices higher than k, about the values of x_k.

$$\begin{aligned}
\mu_\beta(x_k) &= \sum_{x_{k+1}} \phi_{k,k+1}(x_k, x_{k+1}) \left(\sum_{x_{k+2}} \cdots \right) \\
&= \sum_{x_{k+1}} \phi_{k,k+1}(x_k, x_{k+1}) \cdot \mu_\beta(x_{k+1})
\end{aligned}$$

The base case is

$$\mu_\beta(x_n) = \mathbf{1} \ .$$

One pass of "message passing" in each direction along the whole chain yields *all marginal distributions*.

If we have *observed* variable X_j to have value v_j, then we

- Add one more potential

$$\phi_{\text{obs}}(x_j) = \begin{cases} 1 & \text{if } x_j = v_j \\ 0 & \text{otherwise} \end{cases}$$

- Multiply it into $\phi_{j-1,j}$ and $\phi_{j,j+1}$

3.2.2 On a factor graph that is tree

Now we'll generalize the algorithm to apply to factor graphs. Think of the variable X_k. It is connected to a set of factors, which we will call neighbors(X_k). On the "far side" of each factor is a whole set of other factors. These factor sets do not overlap. We will express the marginal at node X_k as a product of new factors F_s, one for each neighbor ϕ_s of X_k, each of which summarizes the effects of the whole set of variables X_w and factors that are on the far side of factor ϕ_s.

We can express the marginal at node k as:

$$\begin{aligned}
\Pr(x_k) &= \sum_{x - \{x_k\}} \Pr(x) \\
&= \sum_{x - \{x_k\}} \prod_i \phi_i(x) \\
&= \sum_{x - \{x_k\}} \prod_{S \in \text{neighbors}(X_k)} F_s(x_k, z_s)
\end{aligned}$$

where z_S is the set of variables that are in the tree rooted at factor S, but not including the children of X_k and $F_s(x_k, z_s)$ represents the product of all the factors in that tree.

Exchanging sums and products in a similar way as for the chain, and adding in one term for each factor S connected to X_k, we can write this as a product of *incoming messages*:

$$\begin{aligned}
\Pr(x_k) &= \prod_{S \in \text{neighbors}(X_k)} \sum_{x_s} F_s(x_k, x_s) \\
&= \prod_{S \in \text{neighbors}(X_k)} \mu_{\phi_S \to X_k}(x_k)
\end{aligned}$$

Now we can express a recursive algorithm in which messages are passed "inward," from the leaves of the tree. There are two kinds of messages: those going from factors to nodes, and those going from nodes to factors.

This is a pretty abstract description. There is a small worked example in the recitation handout.

Factor-to-node messages: Let X_S be the set of variables connected to factor ϕ_S. To compute a message from ϕ_S to X_k, we take the sum over all the values of variables in ϕ_S except for X_k, of the product of the factor ϕ_s applied to those values times the product, over all the all the variables X_m that are neighbors of ϕ_k except for X_k, of the messages going from those variables into ϕ_S:

$$\mu_{\phi_S \to X_k}(x_k) = \sum_{x_S \setminus x_k} \phi_s(x_s) \prod_{X_m \in X_s \setminus X_k} \mu_{X_m \to \phi_S}(x_m) \ .$$

Base case if ϕ is a leaf:

$$\mu_{\phi \to X_i}(x_i) = \phi(x_i) \ .$$

Node-to-factor messages: There is a similar method for computing messages from nodes to factors:

$$\mu_{X_m \to \phi_S}(x_m) = \prod_{\phi_l \in \text{neighbors}(X_m) \setminus \phi_S} \mu_{\phi_l \to X_m}(x_m) \ .$$

Base case if X_i is a leaf:

$$\mu_{X_i \to \phi}(x_i) = 1 \ .$$

Sum-Product Algorithm for finding marginal $\Pr(X_j)$: every variable and factor node computes and sends a message to its remaining neighbor whenever it has received messages from all but one of its neighbors; leaves can send a message immediately. As soon as node X_j has received a message from every one of its neighbors, it does a pointwise multiplication on the messages, and normalizes the result to get $\Pr(X_j)$.

To find all the marginals:

- Arbitrarily pick a root node
- Do one phase of message passing toward root
- Do one phase of message passing away from root

If the original graph was a directed graph, then the marginals we compute will already be normalized; otherwise, the marginals will not be normalized, but since they are over a single variable, it easy to compute $1/Z$ by summing the values of any one of the unnormalized marginals.

Example of sum-product algorithm Figure 2 illustrates the operation of the sum-product algorithm. It contains a factor graph, with the factors shown in blue tables. The messages in red illustrate an "inward pass" toward node A. We'll work through part if it, starting from the message originating from G.

- The base case for a leaf that is a variable is to pass in a message that is 1 for all values. Recall that a message is mapping from possible values of a variable, in this case, G, to real numbers. In this example, all the variables are binary, so the messages have two numbers, one for variable value 0 and one for variable value 1.
- Now, we compute a message from factor ϕ_{DG} to node D. Recall that

Written F_DG→D in the figure, out of laziness.

$$\mu_{\phi_S \to X_k}(x_k) = \sum_{x_S \setminus x_k} \phi_s(x_s) \prod_{X_m \in X_s \setminus X_k} \mu_{X_m \to \phi_S}(x_m) \ .$$

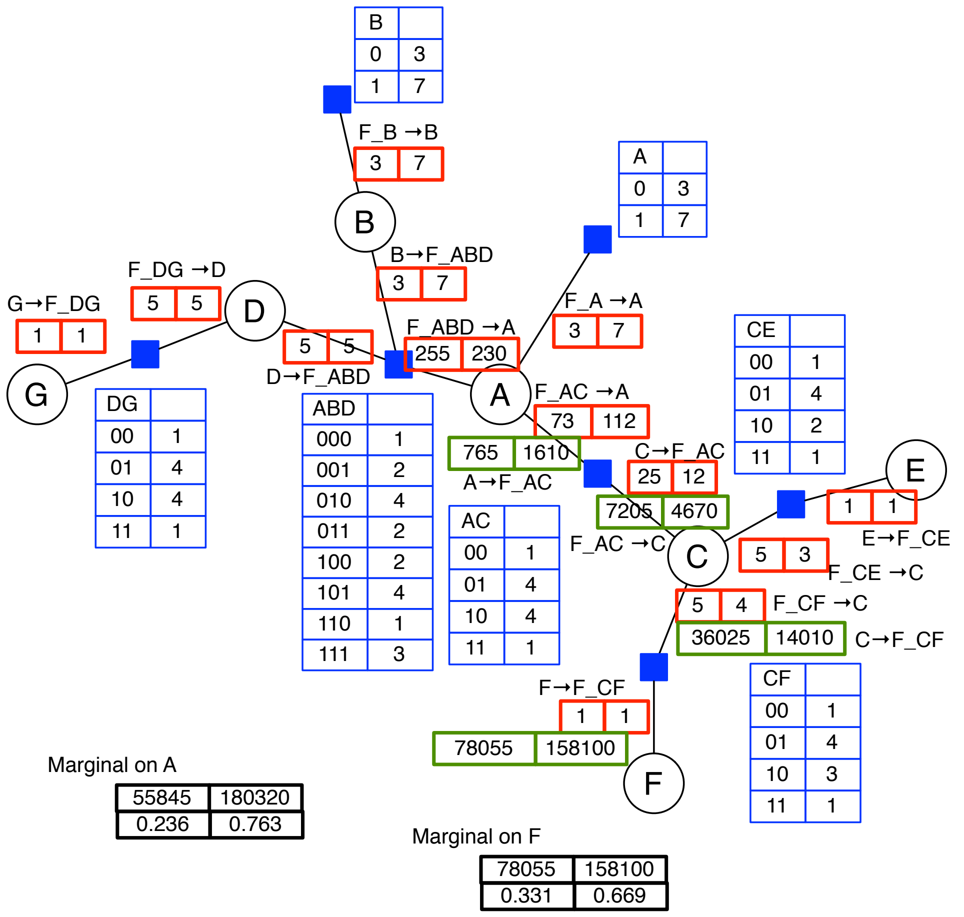

Figure 2: Example partial execution of the sum-product algorithm. All variables are binary. Blue boxes are factors; red boxes are message going in to node A; green boxes are messages going out toward F.

In this case,

$$\mu_{\phi_{DG}\to D}(d) = \sum_g \phi_{DG}(d,g)\mu_{G\to\phi_{DG}}(g) \enspace .$$

Since the incoming message is all 1, this amounts to summing over all values of G in the factor to get an entry in the outgoing message on D. Because there are only two factors connected to variable D, the message from the factor on B to ϕ_B is the same as the message from B to ϕ_{ABD}.

- Now something interesting happens: we have messages $\mu_{B\to\phi_{ABD}}$ and $\mu_{D\to\phi_{ABD}}$ coming in, so factor ϕ_{ABD} is ready to compute an outgoing message to A. This one is somewhat trickier. We have

$$\mu_{\phi_{ABD}\to A}(a) = \sum_{bd} \phi_{ABD}(abd)\cdot\mu_{B\to\phi_{ABD}}(b)\cdot\mu_{D\to\phi_{ABD}}(d) \enspace .$$

  One way to think about this computing this is to "multiply the tables". We would do this by taking the table for the factor, and then multiplying the messages into the table: so, for instance, to take the incoming message from B, we would multiply all the entries in the factor table that have $B = 0$ by 3, and all the entries in the factor table that have $B = 1$ by 7. The other factor has a value of 5 for both values, so that results in just multiplying all the table entries by 5. Then, to compute a message to A, we sum all the entries in the product table that have value $A = 0$ and get 255; then sum all entries that have value $A = 1$ and get 230. This is the message from this factor to variable A.

- We will assume that this process has taken place in other parts of the tree, resulting in three messages coming into variable A. Now, we can compute the marginal at A:

$$\Pr(X_m) \propto \prod_{\phi_l\in\text{neighbors}(X_m)} \mu_{\phi_l\to X_m}(x_m) \enspace .$$

  In this case,

$$\Pr(A) \propto \mu_{\phi_{ABD}\to A}(a)\cdot\mu_{\phi_A\to A}(a)\cdot\mu_{\phi_{AC}\to A}(a) \enspace .$$

  This gives us an unnormalized potential on A of $(55845, 180320)$, which normalizes to $0.236, 0.763$.

- Now, to illustrate the computation of other marginals, let's see how the computation of $\Pr(F)$ proceeds. The critical thing is that the node A sends a message toward factor ϕ_{AC}, which *is not the marginal on* A. It is computed as usual, taking into account messages from all the factors except ϕ_{AC}. So,

$$\mu_{A\to\phi_{AC}}(a) = \mu_{\phi_{ABC}\to A}(a)\cdot\mu_{\phi_A\to A}(a) \enspace .$$

- This process continues, generating the messages shown in green, until we get a marginal on F of $(0.497, 0.503)$.

Incorporating evidence Just as in variable elimination, we can add in a extra factors that assign value 1 to observed values of the evidence variables and 0 to non-observed values of the evidence variables, and proceed with the sum-product algorithm as above.

Continuous variables If the distributions are appropriately conjugate, then we can run the same algorithm on graphs of continuous variables. For instance, distributions in which the variables have linear-Gaussian dependence on one another can be handled directly, by replacing sums with integrals.

3.2.3 Finding MAP values

If we want to find the most likely assignment of some variables, we might consider running the sum-product algorithm to get all the marginals and then finding the maximum value in each marginal. Although these values individually maximize the marginals, they do not necessarily constitute a maximizing assignment in the joint probability distribution.

This is a homework exercise.

In the sum product algorithm, we took advantage of the fact that

$$ab + ac = a \cdot (b + c) \;\; .$$

Now, we will take advantage of a similar relationship for max:

$$\max(ab, ac) = a \cdot \max(b, c) \;\; .$$

Algebraically, max has the same relationship to multiplication as summation does. That means that our strategy for computing $\max_x \prod_i \phi_i(x)$ can be structurally the same as our strategy for computing $\sum_x \prod_i \phi_i(x)$.

Max-product algorithm If we simply take the sum-product algorithm, change all addition operations to max, and do a single pass inward from leaves to an arbitrarily chosen root note, then we will compute $\max_x \Pr(x)$.

Max-sum algorithm Multiplying all those small probabilities can lead to serious loss of precision; to make the computation better conditioned, we can work with log probability values. Because log is monotonic, it preserves the maximum, so:

$$\log \max_x \prod_i \phi_i(x) = \max_x \log \prod_i \phi_i(x) = \max_x \sum_i \log \phi_i(x) \;\; .$$

The distributive property is preserved because

$$\max(a + b, a + c) = a + \max(a, c) \;\; .$$

So, if we change the max-product algorithm by replacing products of factor values by sums of logs of factor values, we get the *max-sum* algorithm, which has the following messages:

$$\mu_{\phi_S \to X_k}(x_k) = \max_{x_S \setminus x_k} \log \phi_s(x_s) + \sum_{X_m \in X_s \setminus X_k} \mu_{X_m \to \phi_S}(x_m) \;\; .$$

$$\mu_{X_m \to \phi_S}(x_m) = \sum_{\phi_l \in \text{neighbors}(X_m) \setminus \phi_S} \mu_{\phi_l \to X_m}(x_m) \;\; .$$

Base case if ϕ is a leaf:

$$\mu_{\phi \to X_i}(x_i) = \log \phi(x_i) \;\; .$$

Base case if X_i is a leaf:

$$\mu_{X_i \to \phi}(x_i) = 0 \;\; .$$

Backtracking Now, what we might really want to find is the entire maximizing assignment, not just its probability. Unfortunately, we can't do it using the straightforward extension of sum-product and doing an "outward" pass of message propagation through the algorithm, in case there are multiple joint assignments with the same maximizing probability that have different assignments.

Not to be confused with backtracking search...

This is sometimes called "decoding" and has a connection to coding theory.

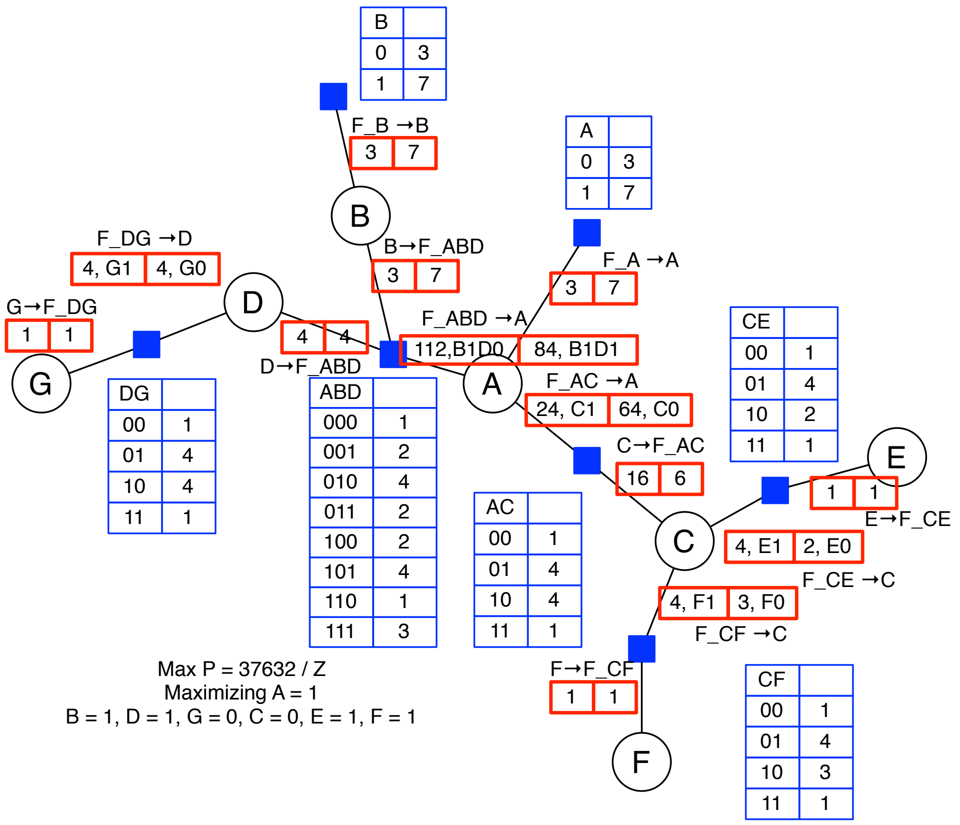

Figure 3: Example execution of the max-product algorithm. All variables are binary. Blue boxes are factors; red boxes are message going in to node A, and also include the local assignments that were selected to compute a particular maximum value; maximizing assignment shown in black text at the bottom.

We begin with an inward pass, but with a bit of extra bookkeeping. When we compute inward message from each clique,

$$\mu_{\phi_S \to X_k}(x_k) = \max_{x_S \setminus x_k} \log \phi_s(x_s) + \sum_{X_m \in X_s \setminus X_k} \mu_{X_m \to \phi_S}(x_m) \enspace .$$

we keep track, for each value of x_k, of a set $Y^*_S(x_k)$ of possible maximizing joint assignments to $X_S \setminus X_k$, which were responsible for the max value in the message.

Then, for whatever node X_j we decided to use as the root in the first pass of max-sum, we know that $x^*_j = \arg\max_{x_j} \Pr(X_j = x_j)$ is a component of some MAP assignment. Now, we begin an outward pass. Instead of passing a probability message into a clique ϕ_S, we pass in the maximizing value x^*_k of the variable that was missing when it was evaluated. Now, that cliques selects any member of $Y^*_S(x_k)$, which provides maximizing assignments for the variables $X_S \setminus X_k$. These maximizing assignments are propagated to any cliques that are neighbors of variables in $X_S \setminus X_k$, and so on, out to the trees.

Example of max-product algorithm The figure 3 shows an example execution of the max-product algorithm. We will walk through some parts of the computation.

We didn't take logs and do max-sum here, in order to keep the numbers intuitively understandable; but in a real example, it's much more numerically stable to do max-sum.

- This time, let's start from node F. The message $\mu_{F \to \phi_{CF}}$ is the base case, $(1, 1)$. We multiply it into the factor ϕ_{CF} . Now, instead of summing out values of F, we maxi-

Which has no effect because it is $(1, 1)$.

mize over them. The result is a message on C, which says that for $C = 0$ we an attain a value of 4, if $F = 1$, and that for $C = 1$ we can attain a value of 3, if $F = 0$.

- At node C, we multiply the incoming messages $(4, 3)$ and $(4, 2)$, to get an outgoing message $\mu_{C \to \phi_{AC}}$ of (16, 6).
- We keep going...let's see what finally happens at node A. We have incoming messages on all the arcs, and just do a pointwise multiply to get a factor on A of $(8064, 37632)$. This potential **is not** a marginal distribution on A. But we can conclude that value of A in an assignment of values to all the variables that maximizes $\Pr(X)$ is $A = 1$.

Because that value is higher in the potential.

- Now, we can do the "backtracking" to find the rest of the assignment. We see that, for $A = 1$, to get the value 84 from ϕ_{ABD}, we need $B = 1$ and $D = 1$.
- If $D = 1$, then to get value 4 from ϕ_{DG}, we need to have $G = 0$.
- Similarly, in the other part of the network, we see we need $C = 0$, which forces $E = 1$ and $F = 1$.

3.3 Converting graphs to trees

The message-passing algorithms are only well-defined on factor graphs that are trees. If we want to do exact inference on a graph that is not a tree, we must convert it into a tree. It is possible to look at the sequence of factors that get generated during variable elimination and use them to create a tree of cliques, which is sometimes called a *junction tree*. Then, there is a message passing algorithm on the junction tree that can be used to compute all the marginals. It is exponential in the tree width (as is variable elimination).

4 Approximate Inference

If we have a graph with a large tree-width, or with a mixture of distributions that makes it impossible to represent intermediate messages, then we have to fall back on approximate inference methods. We will explore three different strategies.

4.1 Loopy BP

One easy approximation method, in models for which belief propagation is appropriate (e.g., all discrete or linear Gaussian models), but where the factor graph is not a tree, is to apply the message passing algorithm. Now, instead of just doing two passes, we could continue to apply it, iteratively.

- This algorithm will not always converge. It is possible to increase the chance of convergence by, instead of computing a new message each time through, to average the old message with the newly-calculated message, and propagate the average instead.
- If it does converge, it might converge to the wrong answer. In particular, it can become overconfident about certain values because information that some evidence yields about a variable might be "double counted" if there are multiple paths through the graph from the evidence to the node in question.