Discrete directed graphical models

$$p(x_1 \cdots x_n) \equiv p(x), \quad x_i \in \{1 \cdots K_i\}$$

In the discrete case, we can think about the joint distribution as an n-dimensional **probability table** that sums to 1 over all cells

$$\sum_{x_1} \cdots \sum_{x_n} p(x_1 \cdots x_n) = 1$$

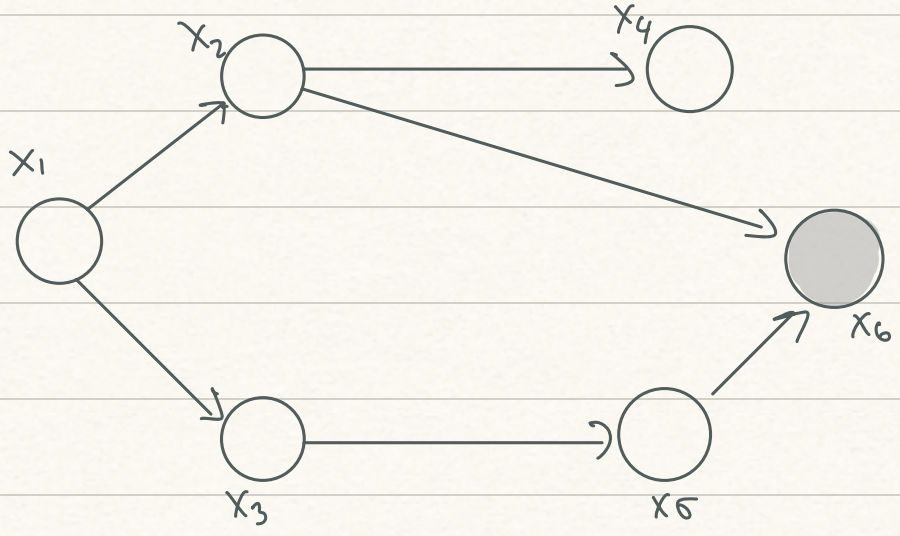
The number of cells in this table is exponential in the number of nodes

$$|p(\cdots)| = \prod_i K_i, \quad \text{e.g. } K^n \text{ if } K_i = K \ \forall i$$

A **directed graphical model** expresses the joint distribution as ← **local probability table**

$$p(x_1 \cdots x_n) = \prod_{i=1}^{n} p(x_i \mid x_{par(i)})$$

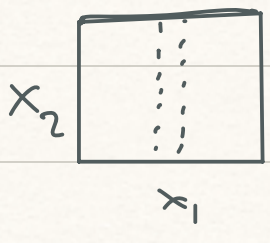
Where par(i) is the set of parents of node i in the graph. Note that this factorization depends on the topological ordering of nodes i = 1...n. Taking the conditional independences into account, we can store the joint distribution using fewer parameters via **local probability tables (LPTs)**. Let's see an example:



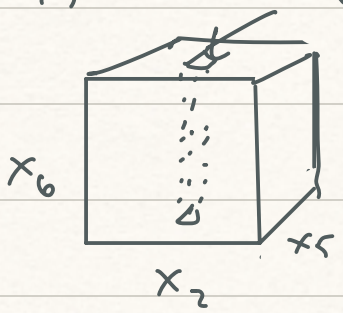
$$p(x_1 \cdots x_6) = p(x_1)\, p(x_2 \mid x_1) \cdots p(x_6 \mid x_2, x_5)$$

local probability tables

$p(x_2 = \cdot \mid x_1 = x_1)$ ←



$p(x_6 = \cdot \mid x_2 = x_2, x_5 = x_5)$ ←



As we see, when there are conditional independences in the model, the total number of parameters we need to represent the joint distribution is just the total number of cells in the local probability tables:

$$\text{total \#cells}: \quad \sum_{i=1}^{n} K_i \prod_{j \in par(i)} K_j$$

Say that K1 = ... = Kn = K, then compare this to how many cells there are in the full table:

$$\sum_{i=1}^{n} K^{|par(i)|} \quad \text{vs} \quad K^n$$

So it goes from being exponential in the number of nodes, to exponential in the size of the largest parent set. In this special case of trees, ever node has only one parent, in which case:

$$nk \quad \text{vs} \quad k^n$$

So graphical models formalize how joint distributions factorize, which we can exploit to fit and query models efficiently. We will explore how to exploit these properties to do efficient exact inference in discrete graphical models today.

Evidence nodes vs. query nodes

Today we will think about performing **queries** efficiently. A query is just a marginal distribution, and performing a query means computing that marginal. Define the following:

1. x_E — the set of "**evidence nodes**": these are the variables whose value we will condition on
2. x_F — the set of "**query nodes**": these are the variables whose marginal distribution given the evidence nodes we seek to compute
3. $x_R = (x_E \cup x_F)^c$ — all other variables in the graph that aren't evidence or query nodes

Goal:

$$p(x_F \mid x_E) = \frac{\sum_{x_R} p(x)}{\sum_{x_F} \sum_{x_R} p(x)}$$

The goal will be to compute this marginal. This involves marginalizing over all variables that are not evidence or query nodes, which would be exponential in the number of them if done naively

e.g. $O(k^{|R|})$ naively if $K_1 = \cdots = K_n = K$

We will use further special bar notation to denote values that are conditioned on. Let's compute:

$$\text{compute} \quad p(x_1 \mid \bar{x}_6) \equiv p(X_1 = x_1 \mid X_6 = \bar{x}_6)$$

Before expanding this, we will define a useful function which is the "**evidence potential**":

$$\delta_{\bar{x}_6}(x_6) = \begin{cases} 1 & \text{if } x_6 = \bar{x}_6 \\ 0 & \neq \end{cases}$$

For all evidence nodes, we multiply the joint by their evidence potential. This lets us write any conditioning operation as simply summing over those variables:

$$p(x_1, \bar{x}_6) = \sum_{x_2} \cdots \sum_{x_6} p(x_1)\, p(x_2 \mid x_1) \cdots p(x_6 \mid x_2, x_5)\, \delta_{\bar{x}_6}(x_6)$$

Some of the sums push in, since not all local probabilities involve all variables:

$$= p(x_1) \sum_{x_2} p(x_2 \mid x_1) \sum_{x_3} \cdots \underbrace{\sum_{x_6} p(x_6 \mid x_2, x_5)\, \delta_{\bar{x}_6}(x_6)}_{\triangleq m_6(x_2, x_5)}$$

Summing out or "**eliminating**" x6 first reveal an **intermediate factor**, which is a function of x2 and x5. We will interpret this factor as a "**message**" from the eliminated variable x6. (The message interpretation will become clear later.)

$$= p(x_1) \sum_{x_2} \cdots \underbrace{\sum_{x_5} p(x_5 \mid x_3)\, m_6(x_2, x_5)}_{\triangleq m_5(x_2, x_3)}$$

Eliminating x5 next reveals a factor involving only x2 and x3. Since it does not depend on x4, it then pushes past that sum:

$$= p(x_1) \cdots \underbrace{\sum_{x_3} p(x_3 \mid x_1)\, m_5(x_2, x_3)}_{\triangleq m_3(x_1, x_3)} \underbrace{\sum_{x_4} p(x_4 \mid x_2)}_{\triangleq m_4(x_2) = 1}$$

We continue by eliminating x4, x3, x2 and the final form is just p(x1) x the last message from x2:

$$= p(x_1) \underbrace{\sum_{x_2} p(x_2 \mid x_1)\, m_3(x_1, x_2)\, m_4(x_2)}_{\triangleq m_2(x_1)}$$

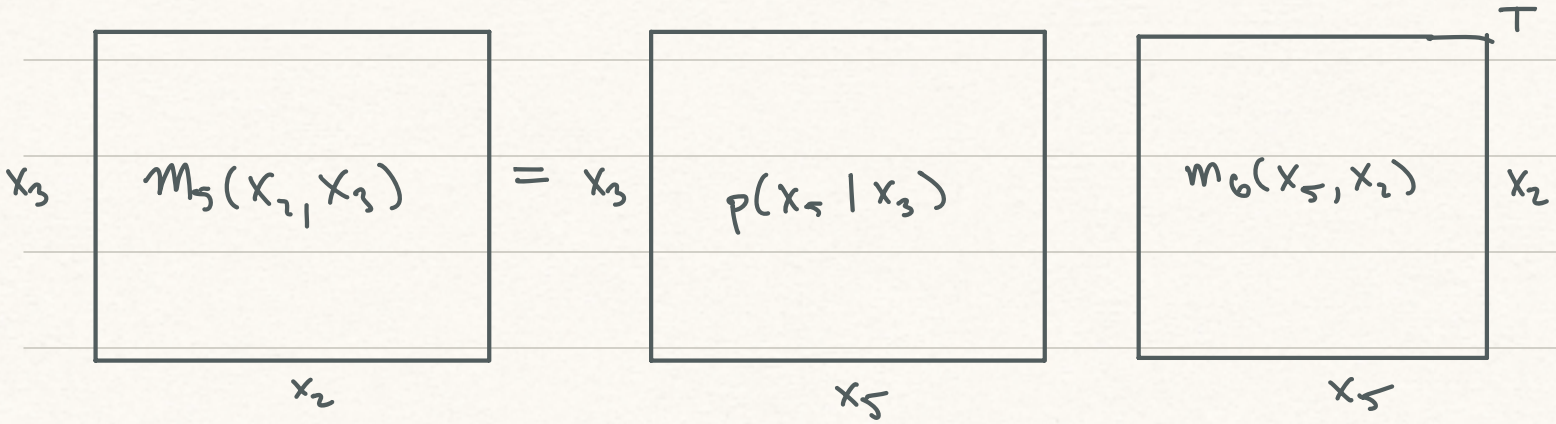
We can thus compute our query as:

$$p(x_1 \mid \bar{x}_6) = \frac{p(x_1)\, m_2(x_1)}{\sum_j p(x_1 = j)\, m_2(x_1 = j)}$$

Comments:

- Iterative **sum-product** procedure
- Eliminated variables left behind **messages** (intermediate factors)
- Each message is a **function** of non-eliminated variables
- We can think about each message as specific kind of **probability table**

$$\text{e.g. } m_5(x_2, x_3) = \sum_{x_5} p(x_5 \mid x_3)\, m_6(x_5, x_2)$$



What exactly is the message computing?

$$m_5(x_2, x_3) = \sum_{x_5} p(x_5 \mid x_3) \underbrace{\sum_{x_6} p(x_6 \mid x_2, x_5)\, \delta_{\bar{x}_6}(x_6)}_{= p(\bar{x}_6 \mid x_2, x_5)}$$

$$= p(x_5, \bar{x}_6 \mid x_2, x_3)$$

$$= p(\bar{x}_6 \mid x_2, x_3)$$

Answer: a table of **conditional evidence probabilities**. These are probabilities of evidence nodes given non-eliminated nodes.

What if we had instead eliminated x2 first?

$$m_2(\cdots) = \sum_{x_2} p(x_2 \mid x_1)\, p(x_4 \mid x_2)\, p(x_6 \mid x_2, x_5)$$

This would have created an intermediate factor of 4 nodes, which would take many more cells of a table to store:

$$\equiv m_2(x_1, x_4, x_5, x_6) \qquad k^4 \text{ terms}$$

So the order of elimination matters. To see this, let's first formalize the variable elimination algorithm, which is a general algorithm for computing any marginal in a graphical model.

The Variable Elimination algorithm for directed graphical models

Input: graph G, target query $p(x_F \mid \bar{x}_E)$

① ORDER$(x_1 \cdots x_n)$ s.t. x_F is last

② INITIALIZE "ACTIVE LIST" of functions as:

$$\mathcal{A} = \left[p(x_i \mid x_{par(i)}) \text{ for } i = 1 \cdots n \right] + \left[\delta_{\bar{x}_i}(x_i) \text{ for } i \in E \right]$$

③ for i in ORDER:

- $\mathcal{A}_i = [\text{all } f(x_i, \cdots) \text{ in } \mathcal{A}]$
- $S_i = [\text{all } x_j \text{ s.t. } f(x_j, x_i, \cdots) \text{ in } \mathcal{A}_i]$
- $\phi_i(x_i, S_i) = \prod_{f \in \mathcal{A}_i} f(x_i, \cdots)$ ← subset of Si

define the message from xi

- $m_i(S_i) = \sum_{x_i} \phi_i(x_i, S_i)$

remove functions from the active set

- $\mathcal{A} = \mathcal{A} - \mathcal{A}_i$

add message from xi to active set

- $\mathcal{A} = \mathcal{A} + m_i(S_i)$

④ RETURN: $\dfrac{\phi_F(x_F, S_F)}{m_F(S_F)} \equiv \dfrac{p(x_F, \bar{x}_E)}{\sum_{x_F} p(x_F, \bar{x}_E)}$

Comments:

- notice that the complexity depends on the sizes of the intermediate sets Si that are created when eliminating each xi
- Si is called the "**elimination clique**"
- Different orderings ORDER(x1...xn) lead to different sequences of Si's, and thus different complexity
- It's generally intractable to determine the optimal ordering. However for **trees** there is a known answer.

remove functions from the active

The Variable Elimination algorithm for trees

The joint distribution for a tree-structured graphical model can always be written in terms of the probability of the **root node** xr times a product over all edges:

$$p(x_1 \cdots x_n) = p(x_r) \prod_{\substack{i \to j \\ \in G}} p(x_j \mid x_i)$$

For defining a general-purpose algorithm define the following "**singleton potentials**" for all nodes xi as:

$$\varphi(x_i) = \begin{cases} 1 & \text{if } x_i \in x_E \\ \delta_{\bar{x}_i}(x_i) & \text{if } x_i \notin x_E \end{cases}$$

For only the **root node**, overwrite the definition to be:

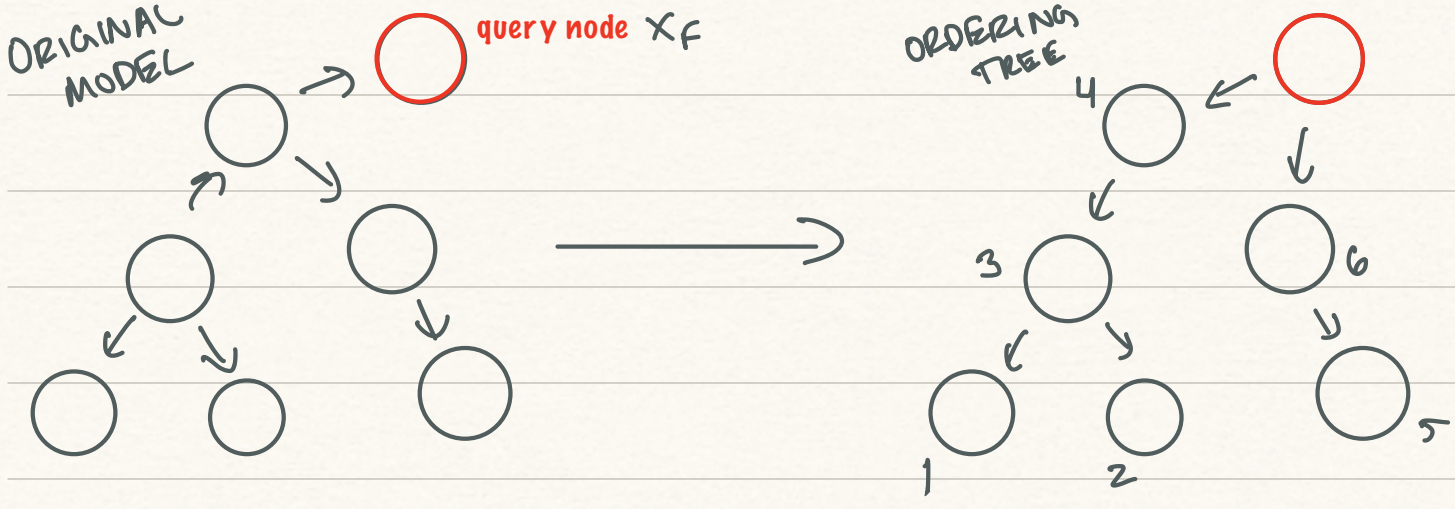
$$\varphi(x_r) = p(x_r)\varphi(x_r)$$

Then define the following "**pairwise potentials**" as:

$$\varphi(x_i, x_j) = p(x_j \mid x_i) \quad \forall \; i \to j \in G$$

The VE algorithm for trees can then be expressed as follows.

First, ORDER(x1....xn) variables by 1) creating a new tree with the **query node at the root** and all arrows pointing away from it, and 2) order nodes by **depth-first search** in the new tree. Example:



Second, initialize the active list to be all singleton and pairwise potentials (as defined above):

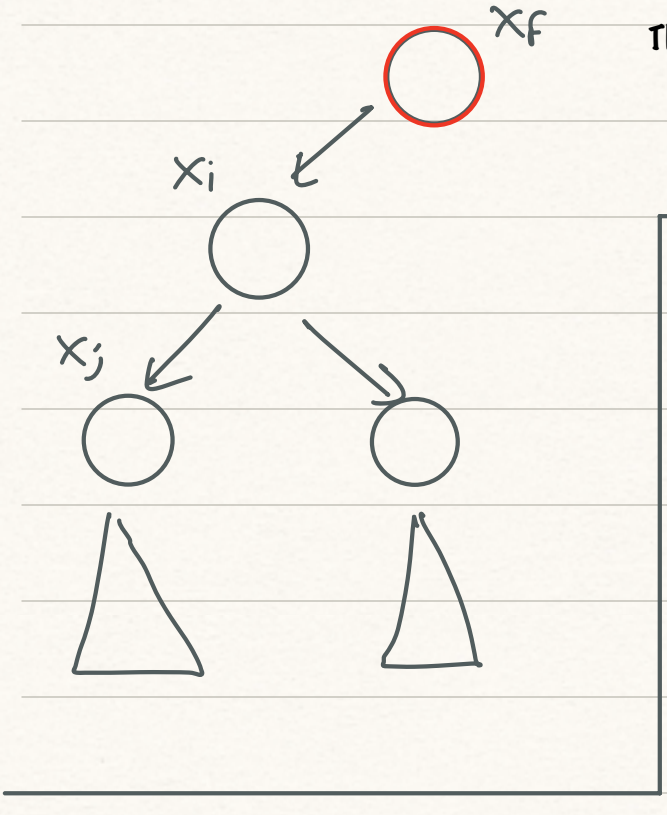
$$\mathcal{A} = [\varphi(x_i, x_j) \ \forall \ i \to j] + [\varphi(x_i) \ \forall \ i]$$

Third, run the for-loop as stated in the original algorithm. In this case, each message would be:

$$m_j(\cdots) = \sum_{x_j} \prod_{f \in k_j} f(m_{j1} \cdots)$$

$$= \sum_{x_j} \varphi(x_j)\varphi(x_i, x_j) \prod_{k} m_k(x_j, \cdots)$$

Each mk(...) is a message from an eliminated node that involves xj. Due to the ordering, we are eliminating nodes bottom up (in the ordering tree, not the original model):

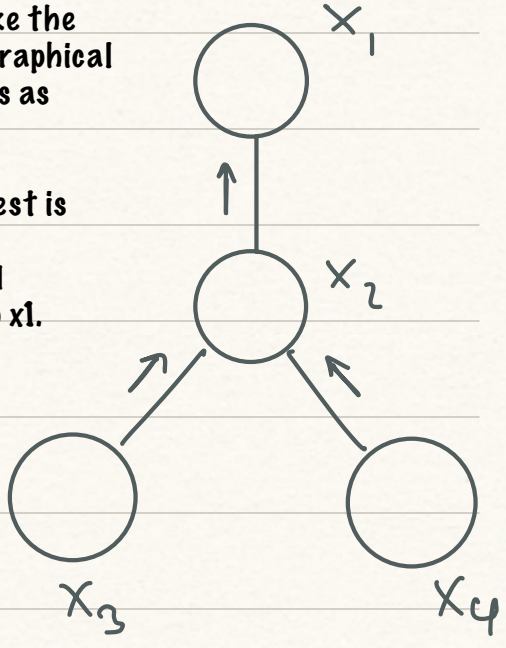


This means that each message only has **one receiver**:

$$m_k(x_j, \cdots) \equiv m_{k \to j}(x_j)$$

Here's a worked example. Take the following tree-structured graphical model (which we will express as undirected).

Say that our query of interest is P(x1). In this case, variable elimination would then send messages up "up the tree" to x1.



$$m_{3 \to 2}(x_2) = \sum_{x_3} \varphi(x_3)\varphi(x_2, x_3), \quad m_{4 \to 2}(x_2) = \sum_{x_4} \cdots$$

$$m_{2 \to 1}(x_2) = \sum_{x_2} \varphi(x_2)\varphi(x_2, x_1) m_{3 \to 2}(x_2) m_{4 \to 2}(x_2)$$

$$p(x_1) \propto \varphi(x_1) m_{2 \to 1}(x_1)$$

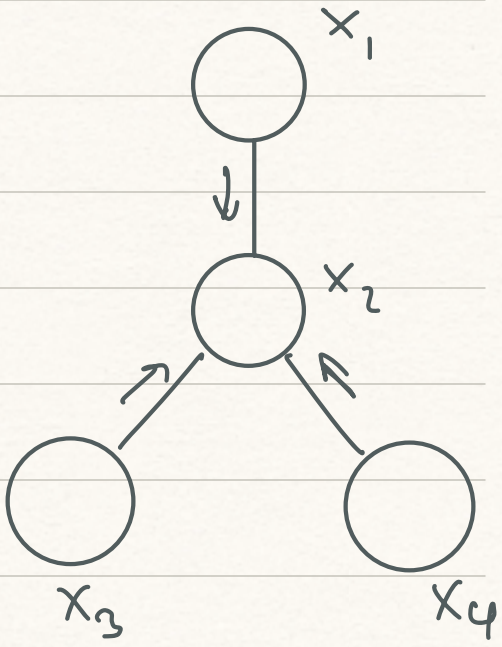
Now let's see the sequence of messages if instead our goal was P(x2).

Messages now flow "up the tree" to x2.

$$m_{1\to 2}(x_2) = \cdots$$

$$m_{3\to 2}(x_2) = \cdots$$

$$m_{4\to 2}(x_2) = \cdots$$



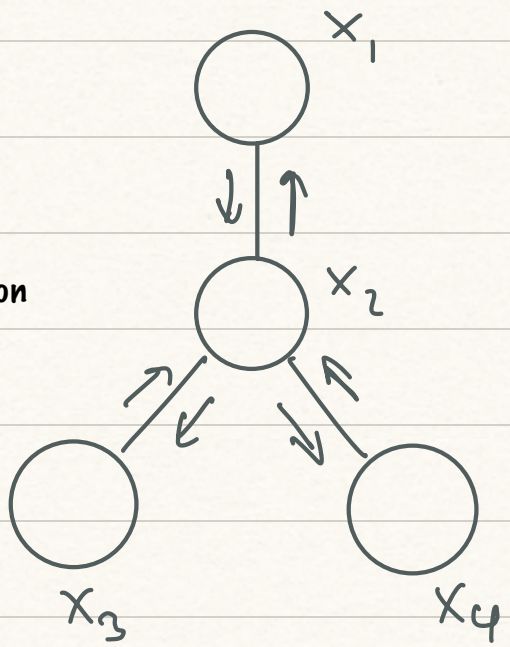
Notice that two of these messages are **exactly the same** as the messages we generated last time when running the VE algorithm to compute P(x1). This suggests that if we want to compute the marginals for many variables, then running VE many times is wasting computation...

The sum-product algorithm

The SP algorithm is a message-passing algorithm that computes **all singleton marginals** in tree-structured graphical models using only **2 messages per edge** (i.e., the cost of 2 runs of variable elimination).

Message passing protocol: A node i sends a message to node j **only after** it has received messages from all other neighbors.

$$m_{i\to j}(x_j) = \sum_{x_i} \varphi(x_i)\,\varphi(x_i, x_j) \prod_{k \in \text{neigh}(x_i)\setminus x_j} m_{k\to i}(x_i)$$

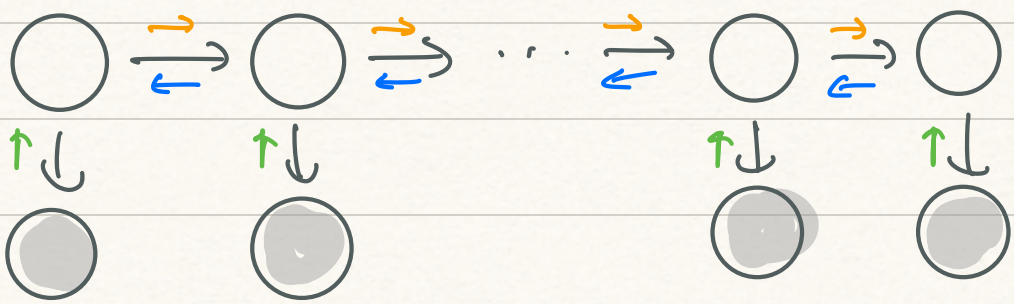


After passing all messages (two per edge), we can compute any singleton marginal as:

$$p(x_i \mid \bar{x}_E) \propto \varphi(x_i) \prod_{j \in \text{neigh}(i)} m_{j\to i}(x_i)$$

Forwards-backwards as the sum-product algorithm

We saw last time how the forward and backward messages let us compute any singleton marginal exactly in HMMs. HMMs are trees, and forwards-backwards is just a special case of the sum-product algorithm.



$\alpha_t(z_t)$

$\beta_t(z_t)$

$l_t(z_t)$

Max-product algorithm

A very similar message-passing algorithm for trees is the max-product algorithm, which allows us to compute the maximum-a-posteriori (MAP) assignment of all nodes using only two sets of messages.

$$\text{Goal: } x^* = \arg\max_x p(x)$$

The key idea is that the max operation distributes across the factors (just like the sum operations):

$$p(X = x^*) = \max_x p(x)$$

$$= \max_{x_1} p(x_1) \max_{x_2} p(x_2 \mid x_1) \cdots$$

Starting at the leaves, and sending messages up the tree, each node computes **two messages** to its parent:

$$m_{j \to i}(x_i) = \max_{x_j} \varphi(x_j) \varphi(x_i, x_j) \prod_{k \in \text{neigh}(j) \setminus x_i} m_{k \to j}(x_j)$$

"

$$m^*_{j \to i}(x_i) = \arg\max_{x_j}$$

"

In other words, it sends two tables, one of which stores the maximum joint probability of descendents achievable for any input value of xi, and the other which stores the corresponding argmax of xj for any input value of xi. Once the messages reach the root node, we then find the MAP by going back down the tree, starting with finding the max joint probability over all nodes, and the argmax over the root:

$$\max p(x) = \max_{x_r} \varphi(x_r) \prod_{k \in \text{neigh}(r)} m_{k \to r}(x_r)$$

"

$$x_r^* = \arg\max_{x_r}$$

"

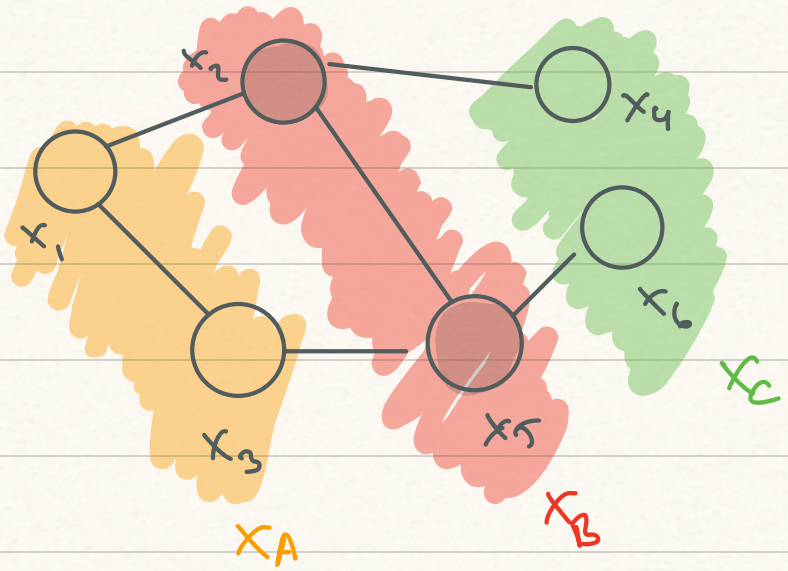
We can then assign every node, going down the tree as:

$$x_j^* = \arg\max_{j \to i} m^*(x_i^*)$$

The well-known **Viterbi algorithm** is just the max-product algorithm for HMMs.

Undirected graphical models

A formalism that has been lurking in this lecture are **undirected graphical models** (UGMs). A UGM also describes a joint distribution, but more simply: two variables are conditionally independent if there is no path between them that doesn't go through an observed variable:



$$x_A \perp\!\!\!\perp x_C \mid x_B$$

UGMs are parameterized using **potential functions**:

"kernel"

$$p(x_1 \cdots x_n) = \frac{k(x_1 \cdots x_n)}{Z}$$

"normalizing constant" or "partition function"

$$p(x_1 \cdots x_n) = \prod_{c \in C} \varphi_c(x_c)$$

potential function for "**maximal clique**" c

Here the product is over all **maximal cliques** in the graph (i.e., sets of nodes that are fully connected in the graph, and which would not continue to be fully connected if any other node was added).

Each potential function takes in a maximal clique and outputs a non-negative potential:

$$\varphi_c(x_c) > 0$$

Notice this is **not a local probability**. It can be greater than 1. In order to normalize the distribution, we need to compute the "**partition function**":

$$Z = \sum_{x_1} \cdots \sum_{x_n} \prod_c \varphi_c(x_c)$$

Algorithms like variable elimination or sum-product can be modified to compute this efficiently in certain types of graphs. In fact, many of these algorithms were originally developed for UGMs.

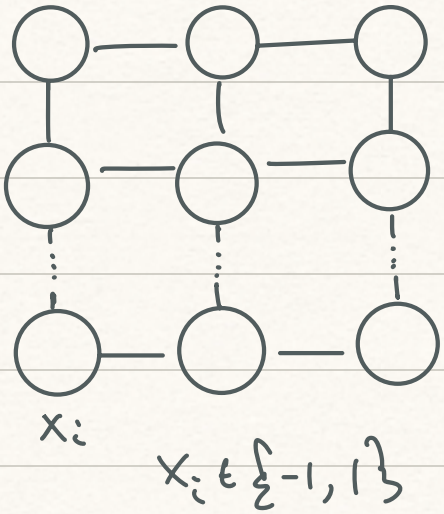
UGMs are useful for modeling **spatial correlation** where there is no clear ordering between the variables.

A classic example is the **Ising model** from physics, where models the "spin" of atoms arranged on a lattice.

The overall **energy H(x1 ...xn)** of a certain configuration is given by:

$$H(x_1 \cdots x_n) = \sum_{i,j} x_i x_j J_{ij}$$

$$\equiv \varphi(x_i, x_j)$$



where Jij simply says whether there is an edge between (i,j)

$$J_{ij} = \begin{cases} c & \text{if } (i,j) \in G \\ 0 & \text{otherwise} \end{cases}$$

High energy states are improbable. So the probability of any given configuration of spins is then:

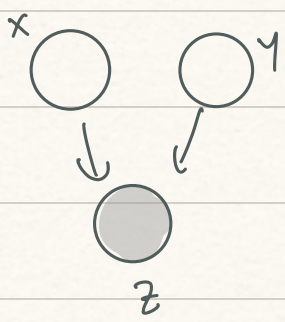
$$p(x_1 \cdots x_n) \propto \exp(-H(x_1 \cdots x_n))$$

A wide class of models called **energy-based models** extends this idea to arbitrary graphs.

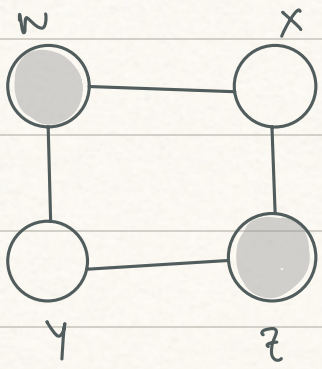
Directed vs undirected graphical models

DGMs and UGMs provide different formalisms with different strengths. With DGMs, we typically first draw the graph, and then read off the conditional independences implied by the graph (which isn't always obvious). This is useful for when we have intuitions about the generative process of our data. With UGMs on the other hand, the graph is directly defined by dependencies. This is better for when we are only aware of complex dependence (such as spatial structure) in our data.

Not all families of models (i.e., sets of conditional independences) can be expressed by both DGMs and UGMs. On the left and right are examples of sets of conditional independences that can **only** be expressed by DGMs and **only** by UGMs.



$x \perp\!\!\!\perp y$ and $x \not\perp\!\!\!\perp y \mid z$



$y \perp\!\!\!\perp x \mid w, z$

$w \perp\!\!\!\perp z \mid x, y$