Outline

1. variable elimination (VE)
2. the VE algorithm on graphs
3. the VE algorithm on trees
4. the sum-product (SP) algorithm
5. extensions / connections

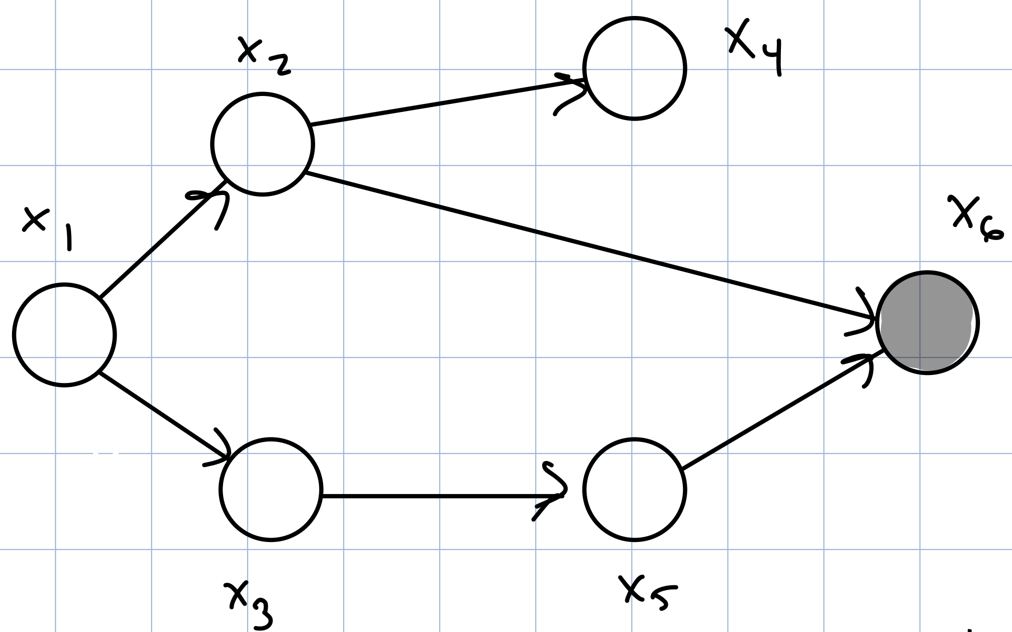
Variable elimination

- model $p(x_1 \dots x_n) \equiv p(x)$, $x_i \in \{1 \dots k\}$
- 3 subsets of variables
  - x_E "evidence" nodes
  - x_F "query" nodes
  - $x_R \equiv (x_E \cup x_F)^c$ "everything else"
- Goal: $\underline{p(x_F \mid x_E)}$

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

- Naively: $O(k^{|R|})$ to compute $p(x_E, x_F)$

- Example:



- compute $p(x_1 \mid \bar{x}_6) \equiv p(X_1 = x_1 \mid X_6 = \bar{x}_6)$
- Define the "evidence potential"

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

- $P(x_1, \bar{x}_6)$

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

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

- "intermediate factor"
- function of x_2, x_5
- variable x_6 is <u>eliminated</u>
- $m_6(\cdots)$ is a "message" from x_6

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

This factor does not depend on x_4, so ...

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

$$= P(x_1) \sum_{x_2} P(x_2|x_1)\, m_3(x_1, x_2)\, m_4(x_2)$$

$$= P(x_1)\, m_2(x_1)$$

This computes $P(x_1, \bar{x}_6)$. The <u>evidence</u> is then:

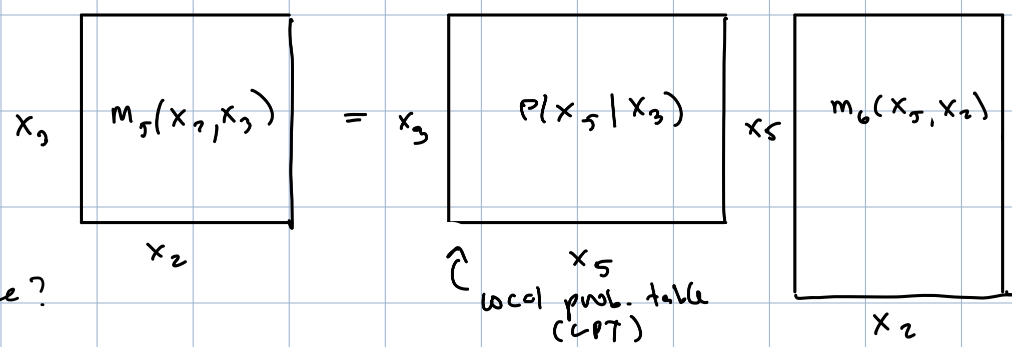
$$P(\bar{x}_6) = \sum_{x_1} P(x_1)\, m_2(x_1)$$

And the posterior is then:

$$P(x_1 | \bar{x}_6) = \frac{P(x_1, \bar{x}_6)}{P(\bar{x}_6)}$$

Discussion:

- iterative product–sum procedure
- eliminated variables leave behind <u>messages</u>
- the message $m_i(\cdots)$ from x_i is a <u>function</u> which takes as arguments the values of <u>non-eliminated</u> variables
- recall that $m_5(x_2, x_3) = \sum_{x_5} P(x_5 \mid x_3)\, m_6(x_5, x_2)$
- think of these as tables:



- what <u>is</u> a message?

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

$$= P(\bar{x}_6 \mid x_2, x_3) \quad \text{(a conditional evidence table)}$$

- look at the graph and imagine eliminating (i.e., marginalizing out) x_3
- <u>Question</u>: what if we had eliminated x_2 first?

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

$$\equiv m_2(x_1, x_4, x_5, x_6) \quad \text{(4d table)}$$

- <u>Intuition</u>: complexity depends on the <u>ordering</u>, and a good ordering exploits the cond. indep. structure

The variable elimination (VE) algorithm

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

① ORDER $x_{1\ldots n}$ with x_F last

② initialize "active list" of functions

$$\mathcal{A} = [\underbrace{p(x_i \mid x_{\pi_i})}_{\text{(LPTs)}} \text{ for } i = 1 \ldots n]$$

$$+ [\underbrace{\delta_{\bar{x}_i}(x_i) \text{ for } i \in E}_{\text{(evidence potentials)}}]$$

③ for i in ORDER:

- $\mathcal{A}_i = [\text{all } f(x_i, \ldots) \text{ in } \mathcal{A}]$
- $S_i = [\text{all } j \text{ s.t. } f(x_i, x_j, \ldots) \text{ in } \mathcal{A}_i]$
- $\phi_i(x_i, s_i) = \prod_{f \in \mathcal{A}_i} f(x_i, \ldots)$
- $m_i(S_i) = \sum_{x_i} \prod_{f \in \mathcal{A}_i} f(x_i, \underbrace{\ldots}_{\text{some subset of } S_i})$
- "eliminate" $\rightarrow$ $\mathcal{A} = \mathcal{A} - \mathcal{A}_i$
- $\mathcal{A} = \mathcal{A} + m_i(S_i)$

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

- Notice that the complexity depends on the sizes of the intermediate S_i sets that are created to eliminate x_i

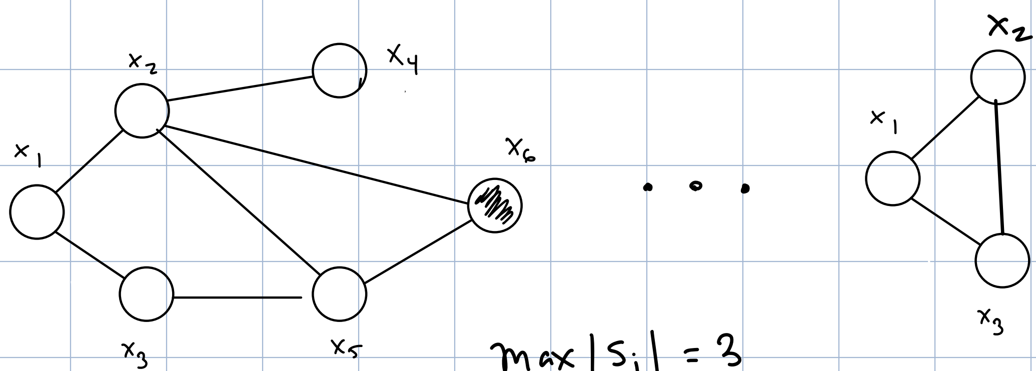
- S_i is called the <u>elimination clique</u>

- for a given ORDER, we can see the sequence of S_i's graphically, by converting to an undirected graph:

  ① moralize

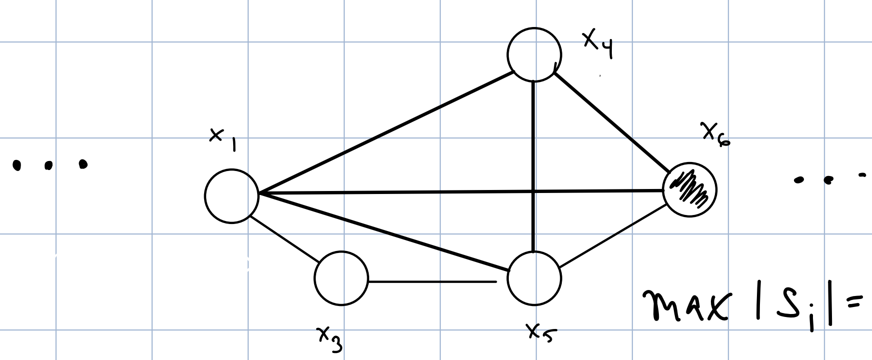
  ② for i in ORDER:
  remove i
  connect all neighbors(i)

e.g. ORDER $= \{6, 4, 5, 2, 3\}$



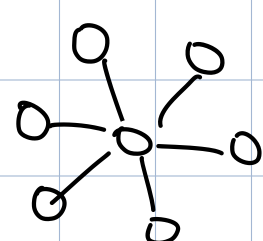
$\max |S_i| = 3$

e.g. ORDER $= \{2, 5, 3, 4, 6\}$



$\max |S_i| = 4$

consider also:



VE for undirected graphs

- Recall the joint in an UGM:

$$P(x_1 \ldots x_n) = \frac{1}{Z} \prod_{c \in C} \varphi(x_c)$$

- Note that

$$P(x_F \mid \bar{x}_E) = \frac{\sum_{x_R} P(x_F, x_R, \bar{x}_E)}{\sum_{x_F} \sum_{x_R} P(x_F, x_R, \bar{x}_E)}$$

$$= \frac{\cancel{\frac{1}{Z}} \sum_{x_R} \prod_{c \in C} \varphi(x_c)}{\cancel{\frac{1}{Z}} \sum_{x_F} \sum_{x_R} \prod_{c \in C} \varphi(x_c)}$$

$$\underset{x_F}{\propto} \sum_{x_R} \prod_{c \in C} \varphi(x_c)$$

- So computing arbitrary marginals boils down to iterative sum-product of potentials $\varphi(x_c) > 0$

- VE also: Same as above except initialize $\mathcal{F}$ with $[\varphi_c(x_c)$ for $c \in C]$ instead of CPTs

- For convenience also singleton potentials $\varphi(x_i)$

$$P(x_1 \ldots x_n) \propto \prod_i \varphi(x_i) \prod_c \varphi(x_c)$$

- This makes conditioning simple:

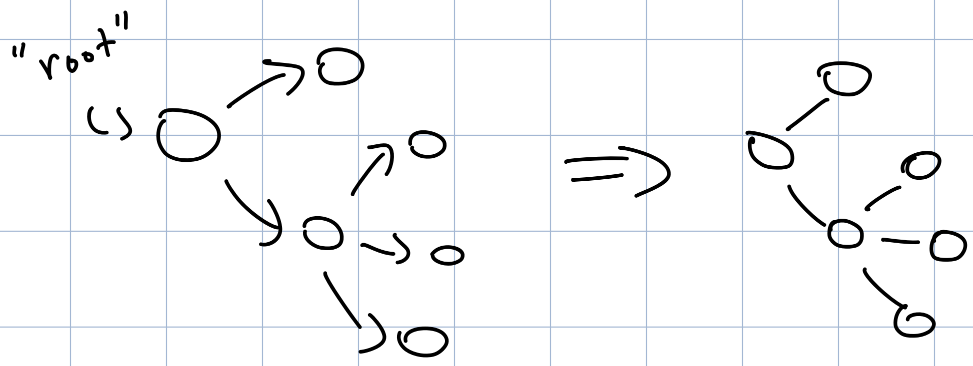
$$\varphi(x_i) := \varphi(x_i)\, \delta_{\bar{x}_i}(x_i) \quad \text{if } i \in E$$

Trees

Def'n (directed): $|\mathrm{par}(x_i)| = 1 \quad \forall i \neq r$

Def'n (undirected): $|\mathrm{paths}(x_i, x_j)| = 1 \quad \forall i, j$

Fact: "Moralized" directed tree = undirected tree



Parameterization directed $\Rightarrow$ undirected:

$$P(x_1 \ldots x_n) = \underset{\text{"root"}}{p(x_r)} \prod_{i \to j} p(x_j \mid x_i)$$

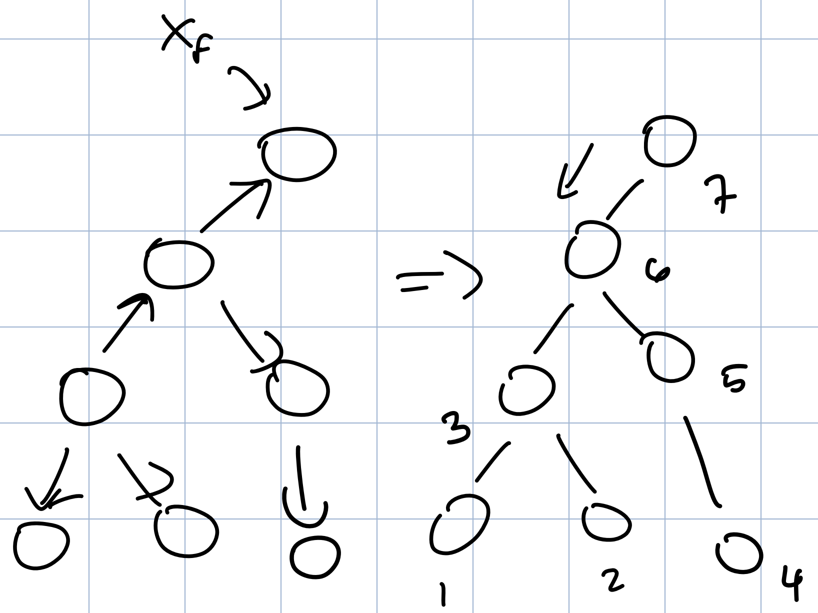
$$\Downarrow$$

$$\boxed{\begin{aligned} \psi(x_i, x_j) &\triangleq p(x_j \mid x_i) \quad \forall\, i \to j \\ \psi(x_r) &\triangleq p(x_r) \\ \psi(x_i) &\triangleq 1 \quad \forall\, i \neq r \end{aligned}}$$

(where again we will redefine the singletons to condition on evidence $\psi(x_i) = \psi(x_i)\,\delta_{\bar{x}_i}(x_i)$)

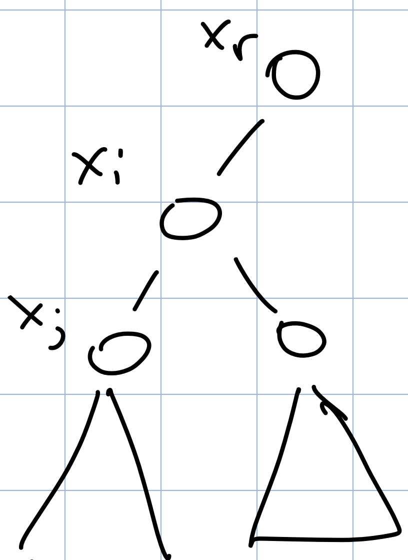
VE for trees

- Query node : X_f
- To determine the ORDER :
  1. Set X_f to the root X_r
  2. Direct edges away from X_f
  3. Order by <u>depth</u> <u>first</u>



$\max(|S_i|) = 2$

- Now eliminate <u>bottom-up</u>



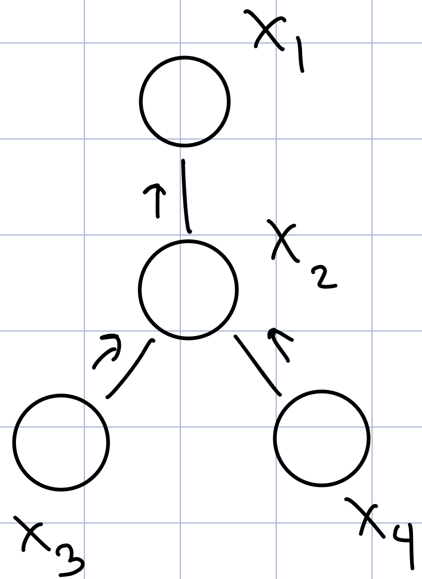
$$m_j(\cdots) = \sum_{j} \prod_{f \in A_j} f(m_j \cdots)$$

$$= \sum_{j} \psi(x_j)\,\psi(x_i, x_j) \prod_{k \in \text{child}(j)} m_k(\cdots)$$

$$= m_j(x_i) \equiv m_{j \to i}(x_i)$$

"<u>message</u>"

Example:



Goal: $P(X_1)$

$$m_{3\to 2}(x_2) = \sum_{x_3} \psi(x_3)\psi(x_2, x_3)$$

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

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

$$P(x_1) \propto \psi(x_1)\, m_{2\to 1}(x_1)$$

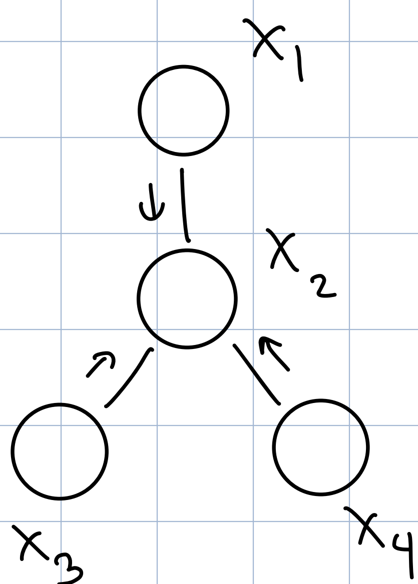
Goal: $P(X_2)$

Notice that:

$$m_{3\to 2} = \cdots$$

$$m_{4\to 2} = \cdots$$

are the same as above.



Goal: $P(X_4)$

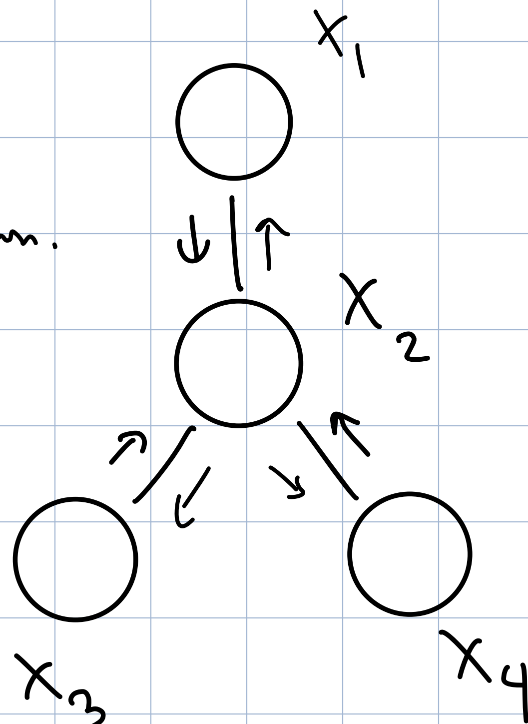
$$m_{1\to 2} = \cdots \text{ (same)}$$

$\longrightarrow$ wasting computation for multiple queries

The sum-product algorithm

- computes <u>all</u> marginals in a tree
- only 2 messages per node
- <u>schedule</u>: choose <u>any</u> node as root.

  Messages up the tree, and messages down.



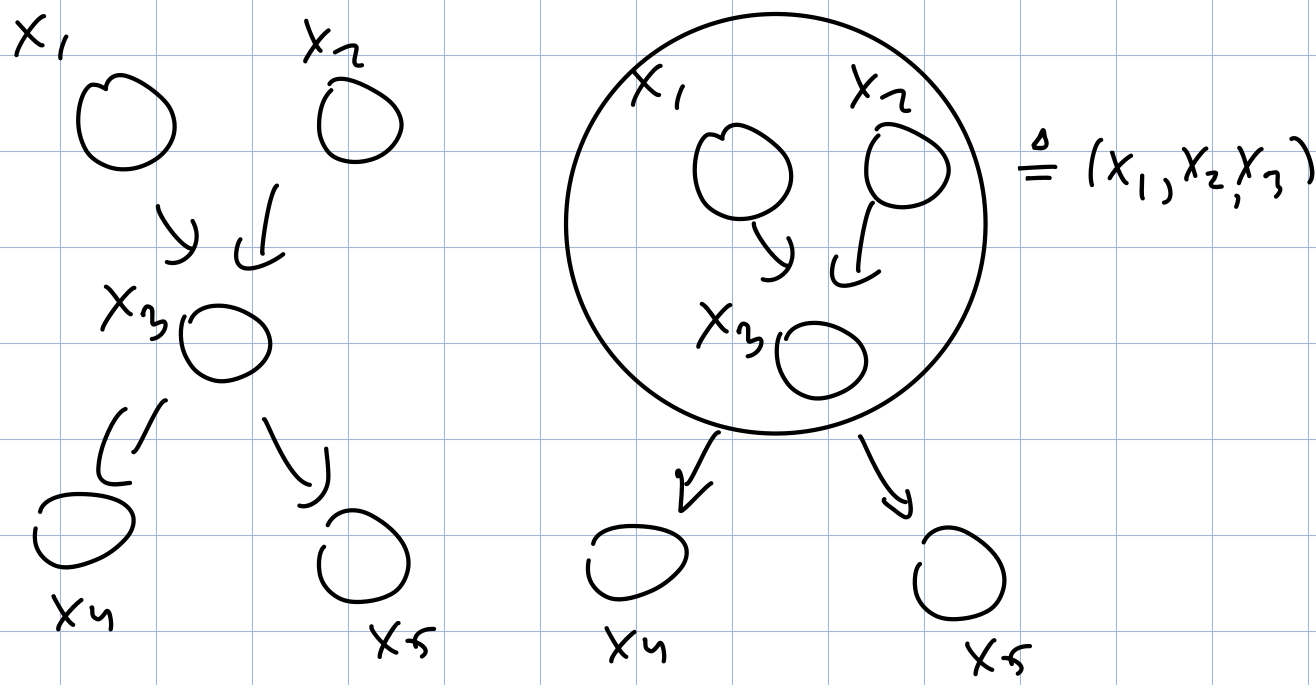
- <u>Claim</u>: all marginals can be computed from $\{m_{i\to j}, m_{j\to i}\}$ using this schedule.
- $P(x_i \mid \bar{x}_E) \propto \varphi(x_i) \prod_{j \in Neigh(i)} m_{j\to i}(x_i)$
- <u>Proof</u>: Def'n of the undirected tree...
- Another perspective: asyncronous <u>message-passing</u>
- <u>Message-passing protocol</u>:

  A node i sends message to j $m_{i\to j}$ only after it has received messages from other neighbors $k \in Neigh(i) \setminus j$.

- For <u>trees</u>, we can confirm that a fixed schedule produces messages that satisfy the protocol.

Junction tree algorithm

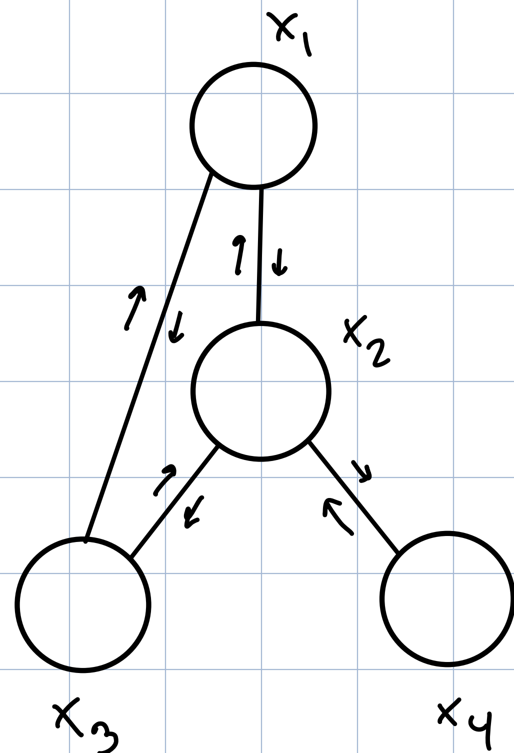
- Convert any graph into a tree of super nodes, then run SP.



- Exponential in size of super nodes

Belief propagation (BP)

- SP = BP on trees
- For more graphs BP is "loopy" and must be iterated until convergence
- Convergence not guaranteed



- When LBP converges it *often* produces good solutions
- One can show that LBP is *minimizing* the

<u>Bethe free energy</u>

$$KL(Q \| P) = \sum_x Q(x) \log Q(x) - \sum_x Q(x) \log P(x)$$

$$= -H_Q(x) - \mathbb{E}_Q[\log P(x)]$$

$$= -H_Q(x) - \mathbb{E}_Q[\log f(x)] + \log Z$$

- So we can minimize $KL(Q \| P)$ without doing inference in $P(x)$

$$\hookrightarrow \min_Q KL(Q \| P) = \max_Q \underbrace{\mathbb{E}_Q[\log f(x)] + H_Q(x)}_{\equiv \text{"ELBO"} \equiv -\text{"Free Energy"}}$$

- $H_Q(x) = \sum_{x_1} \cdots \sum_{x_n} Q(x) \log \frac{1}{Q(x)}$

  often intractable

- For trees, this is tractable, for the same reasons BP is.
- $H^{Bethe}(x)$ computes $H_Q(x)$ as if the graph were a tree:

$$Q^{Bethe}(x) = \prod_{j \to i} \frac{Q(x_i, x_j)}{Q(x_j) Q(x_i)} \prod_i Q(x_i)$$

(products of singleton and pairwise marginals)

- (This is exact for a tree.)
- $F^{Bethe} = -\mathbb{E}_{Q^B}[\log f(x)] + H^{Bethe}(x)$
- $Q^{Bethe}(x)$ is only "legal" if it is in the <u>marginal polytope</u>

$$M(G) = \{(Q(x_i), Q(x_i, x_j)) : \exists\, Q \text{ with those marginals}\}$$

- Searching over this space is hard. Instead, LBP searches in $L(G) \supset M(G)$
- A <u>fixed point</u> of LBP is

$$\underset{Q \in L(G)}{\operatorname{argmin}} F^{Bethe}(P, Q)$$

- Often works well, but Bethe free energy is not guaranteed to be near the free energy and LBP is not guaranteed to converge

---

<u>Connections</u>

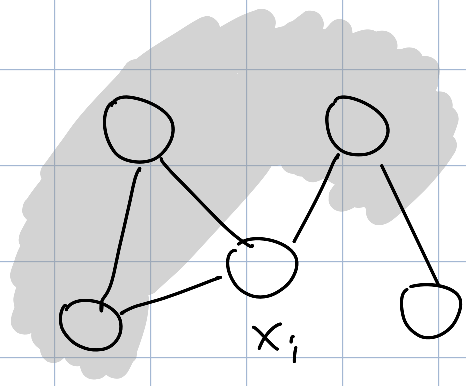
- LBP is a form of <u>variational inference</u>
- More well-known VI is based on

$$\underset{Q \in \mathcal{Q}}{\operatorname{argmax}} \mathbb{E}_Q[\log f(x)] + H_Q(x)$$

for a simple class of tractable $\mathcal{Q}$

- EM is a special case of VI

- Many other inference algos are a form of iterative message-passing
- e.g., Gibbs sampling



← "Markov blanket" of x_i : M_i

x_i

for iter in 1...S :
    for i in nodes :
        $x_i \sim p(x_i \mid x_{M_i})$
        "complete conditional"

- For <u>exact</u> inference, discrete or <u>Gaussian</u>
- $$p(x) = \frac{1}{Z} \prod_{(i,j)} \varphi(x_i, x_j) \prod_i \varphi_i(x_i)$$

$$\varphi(x_i, x_j) \overset{\circ}{=} \exp(x_i^T V_{ij} x_j)$$

$$\varphi(x_i) \overset{\circ}{=} \exp\left(\frac{1}{\delta_i}(x_i - \mu_i)^2\right)$$

- Same BP messages as above...

$$p(x_i) \propto \varphi(x_i) \prod_{j \in neigh(i)} m_{j \to i}(x_i)$$
$$\propto \mathcal{N}(x_i, \ldots)$$

- Generalizes the "Kalman filter"

Max product algorithm

Goal: $x^* = \arg\max_x P(X)$ (MAP)

$$P(x^*) = \max_x P(x)$$
$$= \max_{x_1} P(x_1) \max_{x_2} P(x_2 | x_1) \ldots$$

(max is also distributive)

Messages (for trees):

up the tree:

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

$$\delta_{j \to i}(x_i) = \arg\max_{x_j} \text{ " } \qquad \text{"}$$

at the root:

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

$$x_r^* \leftarrow \arg\max_{x_r} \text{ " } \qquad \text{"}$$

down the tree:

$$x_j^* \leftarrow \delta_{j \to i}(x_i^*)$$