EE512A – Advanced Inference in Graphical Models

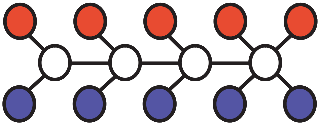
— Fall Quarter, Lecture 2 —

http://j.ee.washington.edu/~bilmes/classes/ee512a_fall_2014/

Prof. Jeff Bilmes

University of Washington, Seattle
Department of Electrical Engineering
http://melodi.ee.washington.edu/~bilmes

Oct 1st, 2014



Announcements

- Reading assignments, posted to our canvas announcements page (https://canvas.uw.edu/courses/914697/announcements): intro.pdf, ugms.pdf on undirected graphical models, and tree_inference.pdf on trees.
- Slides from previous time this course was offered are at our previous web page (http://j.ee.washington.edu/~bilmes/classes/ee512a_fall_2011/) and even earlier at http://melodi.ee.washington.edu/~bilmes/ee512fa09/.

Class Road Map - EE512a

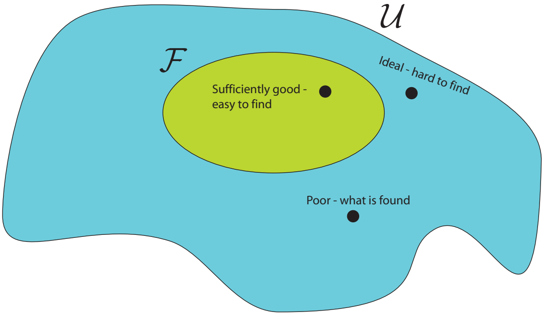
- L1 (9/29): Introduction, Families, Semantics
- L2 (10/1): MRFs, Inference on Trees
- L3 (10/6):
- L4 (10/8):
- L5 (10/13):
- L6 (10/15):
- L7 (10/20):
- L8 (10/22):
- L9 (10/27):
- L10 (10/29):
- L11 (11/3):
- L12 (11/5):
- L13 (11/10):
- L14 (11/12):
- L15 (11/17):
- L16 (11/19):
- L17 (11/24):
- L18 (11/26):
- L19 (12/1):
- L20 (12/3):
- Final Presentations: (12/10):

Finals Week: Dec 8th-12th, 2014.

Machine learning within restricted families

- Let $\mathcal{U}$ be the universe of all distributions over N r.v.s.
- Sample data, along with domain knowledge, used to select resulting $p(x)$ from $\mathcal{U}$ that is "close enough" to $p_{\text{true}}(x_1, \dots, x_N)$.
- Searching within $\mathcal{U}$ is infeasible/intractable/impossible.
- Desire a restricted but useful family $\mathcal{F} \subset \mathcal{U}$.

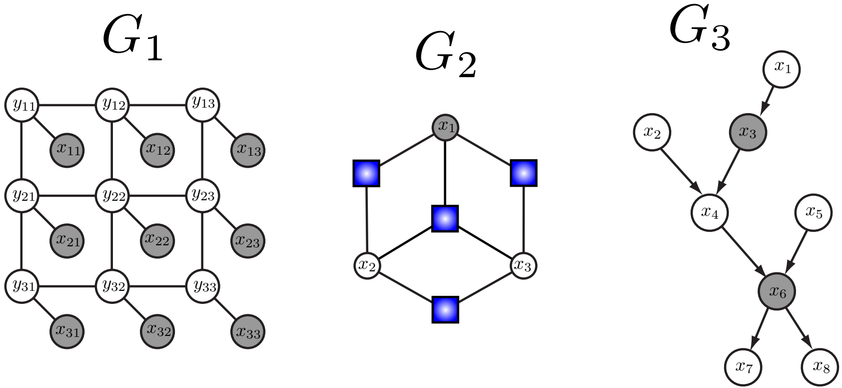
- Size of $\mathcal{U}$ too large, complex, and with many local optima.
- Obtainable solution in $\mathcal{F}$ better than feasible solution in $\mathcal{U}$
- Graphical models provide a framework for specifying $\mathcal{F} \subseteq \mathcal{U}$



Graphical Models

- A graphical model is a visual, abstract, and mathematically formal description of properties of families of probability distributions (densities, mass functions)

There are many types of graphical models, for example Markov random fields (left), factor graphs (center), and Bayesian networks (right):



Graphical Model

- Each type of graphical model requires a certain type of graph (e.g., undirected, or DAG) and a set of rules (or "Markov properties") to define the GM.
- A graphical model is a pair $(G, \mathcal{M}) = ((V, E), \mathcal{M})$, a graph G and a set of properties $\mathcal{M}$ that define what the graphical model means.
- Conceptually, one can think of a property $r \in \mathcal{M}$ as a predicate on a graph and a distribution, so $r(p, G) \in \{\text{true}, \text{false}\}$.
- $(G, \mathcal{M})$ consists of a family of distributions over x_V where all predicates hold. That is

$$\mathcal{F}(G, \mathcal{M}) = \{p : p \text{ is a distribution over } X_V \text{ and },\\ r(p, G) = \text{true}, \forall r \in \mathcal{M}\} \tag{2.6}$$

- $\mathcal{F}(G, \mathcal{M}) \subseteq \mathcal{U}$

What is graphical model inference?

- Inference, as we saw, is computing probabilistic queries such as:
  1. probability of evidence (marginalize the hidden variables)
  $$p(\bar{x}_E) \tag{2.8}$$
  2. posterior probability, for $S \subseteq V \setminus E$ do
  $$p(x_S|\bar{x}_E) \tag{2.9}$$
  3. most probable assignment, for $S \subseteq V \setminus E$ do
  $$\underset{x_S \in \mathcal{D}_{X_S}}{\operatorname{argmax}} p(x_S, \bar{x}_E). \tag{2.10}$$
- Given a graph G, we want to derive this just based just on $(G, \mathcal{M})$ and derive this automatically.
- We want to understand the computational complexity of the procedure based just on $(G, \mathcal{M})$.
- amortization: we want to derive a procedure that works for any $p \in \mathcal{F}(G, \mathcal{M})$ for a given rule set.

Graphical Models

- A graphical model consists of a graph and a set of rules or properties $\mathcal{M}$ (often called *Markov properties*).
- Unlike $\mathcal{U}$, which is the family of all distributions over n r.v.s, $\mathcal{F}(G, \mathcal{M}) \subseteq \mathcal{U}$ is a subset of distributions.
- Any member of $\mathcal{F}(G, \mathcal{M})$ must respect the constraints that are specified by the GM.
- Any distribution that does not respect even one of the GM's constraints is not a member of the family.
- In a GM, the constraints take the form of factorization (which are most often, conditional independence constraints).
- Factorization is useful since it allows for the distributive law to enable the use of dynamic programming for much faster exact inference than naive.
- Finding best way of doing inference is entirely graph theoretic operation.

Markov random fields

- One type of graphical model (we'll study in this course).
- Has its origin in statistical physics (Boltzmann distributions, Ising models of atomic spin) and image processing (grid-based models).
- Example Ising model: Let $W = [w_{ij}]_{ij}$ be a matrix of weights. Note that many of these weights might be zero. Let $s = [s_i]_i = (s_1, s_2, \ldots, s_n)$ be a vector of binary random variables, $s_i \in \{-1, +1\}$. Define the "energy" as

$$E(s) = -\sum_{ij} s_i s_j w_{ij} \tag{2.1}$$

- Then define a distribution over s as

$$p(s) = \frac{1}{Z} \exp(-E(s)/T) \tag{2.2}$$

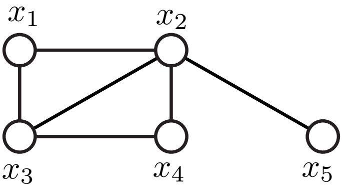
where T is the temperature of the model and $Z = \sum_s \exp(-E(s)/T)$ is a normalizing constant.

Markov random fields
Ising model (cont.)

- Most often s corresponds to a grid (i.e., s is really a matrix or 3D-matrix).
- Ising model: w_{ij} determines the interaction style of variables: if $w_{ij} = 0$ the no interaction. If $w_{ij} > 0$ then more probable for $s_i = s_j = \pm 1$. If $w_{ij} < 0$ then more probable for $s_i \neq s_j$.
- We can think of matrix W and vector s as a graph, $G = (V, E)$ where s corresponds to V and W corresponds to E — that is, $(i, j) = e \in E$ only when $w_{ij} \neq 0$.
- We might expect that any Ising model $p \in \mathcal{F}(G, \mathcal{M}^{(\text{mrf})})$ for appropriately defined MRF rules.

Clique Factorization

- The "Cliques" of a graph $G = (V, E)$, or $\mathcal{C}(G)$, in a graph are the set of fully connected nodes.
- If $C \in \mathcal{C}(G)$ and $u, v \in C$ then $(u, v) \in E(G)$
- In the following graph



cliques are $\mathcal{C} = \{\{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{1, 2\}, \{1, 3\}, \{2, 3\},$ $\{1, 2, 3\}, \{3, 4\}, \{2, 4\}, \{2, 3, 4\}, \{2, 5\}\}$.

Clique Factorization

- Given graph G with cliques $\mathcal{C}(G)$, consider a probability distribution that can be represented as follows:

$$p(x_V) = \frac{1}{Z} \prod_{C \in \mathcal{C}(G)} \phi_C(x_C) \tag{2.3}$$

$$Z = \sum_{x_V} \prod_{C \in \mathcal{C}} \phi_C(x_C) \tag{2.4}$$

- Actually, we don't always need Z explicitly since it is a constant and can be distributed into the factors in a variety of ways, leading to:

$$p(x_V) = \prod_{C \in \mathcal{C}(G)} \phi_C(x_C) \tag{2.5}$$

where only the factorization is depicted.

Clique Factorization

- More formally, consider the following family:

$$\mathcal{F}(G, \mathcal{M}^{(\mathsf{cf})}) = \Big\{p : \forall C \in \mathcal{C}(G), \exists \psi_C(x_C) \geq 0 \text{ and } p(x_V) = \prod_{C \in \mathcal{C}(G)} \psi_C(x_C)\Big\} \tag{2.6}$$

Clique Factorization

- More formally, consider the following family:

$$\mathcal{F}(G, \mathcal{M}^{(\text{cf})}) = \{p : \forall C \in \mathcal{C}(G), \exists \psi_C(x_C) \geq 0 \text{ and } p(x_V) = \prod_{C \in \mathcal{C}(G)} \psi_C(x_C) \Big\} \tag{2.6}$$

- are the clique factors unique?

MaxClique Factorization

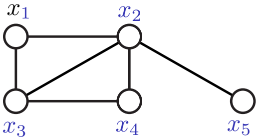
- The "MaxCliques" of a graph $G = (V, E)$, or $\mathcal{C}^{(mc)}(G)$, in a graph are the set of fully connected nodes that can't be enlarged

MaxClique Factorization

- The "MaxCliques" of a graph $G = (V, E)$, or $\mathcal{C}^{(mc)}(G)$, in a graph are the set of fully connected nodes that can't be enlarged — adding any node to a maxclique renders it no longer a clique.

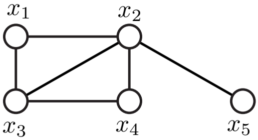
MaxClique Factorization

- The "MaxCliques" of a graph $G = (V, E)$, or $\mathcal{C}^{(mc)}(G)$, in a graph are the set of fully connected nodes that can't be enlarged — adding any node to a maxclique renders it no longer a clique.
- MaxCliques of previous graph (repeated below) are $\{\{1, 2, 3\}, \{2, 3, 4\}, \{2, 5\}\}$



MaxClique Factorization

- The "MaxCliques" of a graph $G = (V, E)$, or $\mathcal{C}^{(mc)}(G)$, in a graph are the set of fully connected nodes that can't be enlarged — adding any node to a maxclique renders it no longer a clique.
- MaxCliques of previous graph (repeated below) are $\{\{1, 2, 3\}, \{2, 3, 4\}, \{2, 5\}\}$



- New properties $\mathcal{M}^{(\mathsf{mcf})}$ based on maxcliques define family

$$\mathcal{F}(G, \mathcal{M}^{(\mathsf{mcf})}) = \Big\{ p : \forall C \in \mathcal{C}^{(mc)}(G), \exists \psi_C(x_C) \geq 0 \text{ and } p(x_V) = \prod_{C \in \mathcal{C}^{(\mathsf{mc})}} \psi_C(x_C) \Big\} \tag{2.7}$$

Comparisons of families

- How do $\mathcal{F}(G, \mathcal{M}^{(\mathsf{cf})})$ and $\mathcal{F}(G, \mathcal{M}^{(\mathsf{mcf})})$ compare?

Comparisons of families

- How do $\mathcal{F}(G, \mathcal{M}^{(\text{cf})})$ and $\mathcal{F}(G, \mathcal{M}^{(\text{mcf})})$ compare?

Lemma 2.3.1

$\mathcal{F}(G, \mathcal{M}^{(cf)}) \subseteq \mathcal{F}(G, \mathcal{M}^{(mcf)})$

Comparisons of families

- How do $\mathcal{F}(G, \mathcal{M}^{(\text{cf})})$ and $\mathcal{F}(G, \mathcal{M}^{(\text{mcf})})$ compare?

Lemma 2.3.1

$\mathcal{F}(G, \mathcal{M}^{(cf)}) \subseteq \mathcal{F}(G, \mathcal{M}^{(mcf)})$

Lemma 2.3.2

$\mathcal{F}(G, \mathcal{M}^{(cf)}) \supseteq \mathcal{F}(G, \mathcal{M}^{(mcf)})$

Comparisons of families

- How do $\mathcal{F}(G, \mathcal{M}^{(\text{cf})})$ and $\mathcal{F}(G, \mathcal{M}^{(\text{mcf})})$ compare?

Lemma 2.3.1

$\mathcal{F}(G, \mathcal{M}^{(cf)}) \subseteq \mathcal{F}(G, \mathcal{M}^{(mcf)})$

Lemma 2.3.2

$\mathcal{F}(G, \mathcal{M}^{(cf)}) \supseteq \mathcal{F}(G, \mathcal{M}^{(mcf)})$

- Therefore

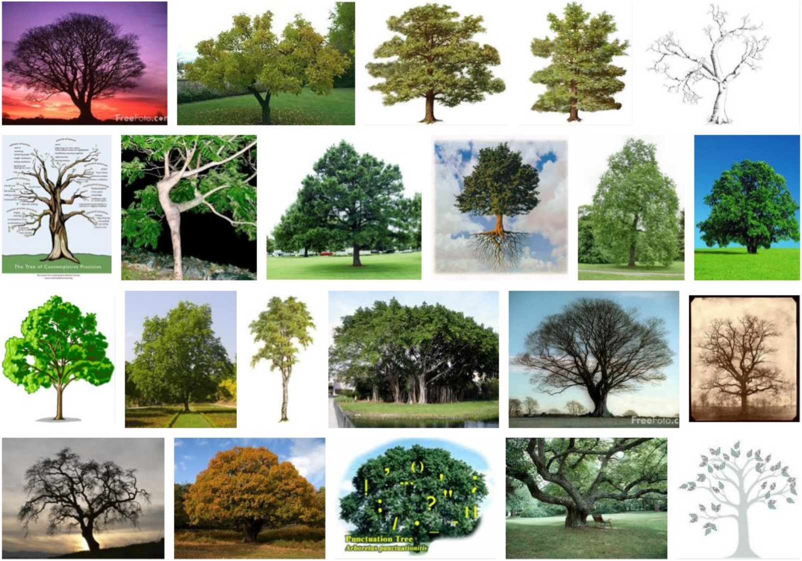
Corollary 2.3.3

$\mathcal{F}(G, \mathcal{M}^{(cf)}) = \mathcal{F}(G, \mathcal{M}^{(mcf)})$

- Since rules are identical, we use $\mathcal{M}^{(\text{f})}$ for clique factorization, and family $\mathcal{F}(G, \mathcal{M}^{(\text{f})})$.
- Often, it is not so obvious that different families are identical.
- Equally often, different families are indeed different.

Trees

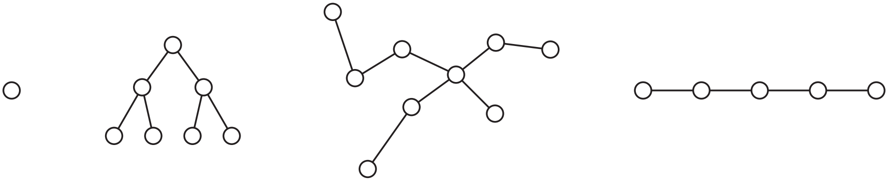
Trees


The Tree of Contemplative Practices
Punctuation Tree
Arboretus punctuationitis

Trees defined

Definition 2.4.1

A graph $G = (V, E)$ is a *forest* if it is the case that for all $u, v \in V$, there is no more than one path that connects u to v in G. Given a forest G, if for all $u, v \in G$ there is a unique path connecting u and v, then it is called a *connected forest* or just simply a *tree*.



Trees defined in many ways

Theorem 2.4.2 (Trees, Berge)

Let $G = (V, E)$ be an undirected graph with $|V| = n > 2$. Then each of the following properties are equivalent and each can be used to define when G is a tree:

- *G is connected and has no cycles*

Trees defined in many ways

Theorem 2.4.2 (Trees, Berge)

Let $G = (V, E)$ be an undirected graph with $|V| = n > 2$. Then each of the following properties are equivalent and each can be used to define when G is a tree:

- *G is connected and has no cycles*
- *G has $n - 1$ edges and has no cycles,*

Trees defined in many ways

Theorem 2.4.2 (Trees, Berge)

Let $G = (V, E)$ be an undirected graph with $|V| = n > 2$. Then each of the following properties are equivalent and each can be used to define when G is a tree:

- *G is connected and has no cycles*
- *G has $n - 1$ edges and has no cycles,*
- *G is connected and contains exactly $n - 1$ edges,*

Trees defined in many ways

Theorem 2.4.2 (Trees, Berge)

Let $G = (V, E)$ be an undirected graph with $|V| = n > 2$. Then each of the following properties are equivalent and each can be used to define when G is a tree:

- *G is connected and has no cycles*
- *G has $n-1$ edges and has no cycles,*
- *G is connected and contains exactly $n-1$ edges,*
- *G has no cycles. Exactly one cycle created if edge added to G.*

Trees defined in many ways

Theorem 2.4.2 (Trees, Berge)

Let $G = (V, E)$ be an undirected graph with $|V| = n > 2$. Then each of the following properties are equivalent and each can be used to define when G is a tree:

- *G is connected and has no cycles*
- *G has $n - 1$ edges and has no cycles,*
- *G is connected and contains exactly $n - 1$ edges,*
- *G has no cycles. Exactly one cycle created if edge added to G.*
- *G is connected, and if any edge is removed, the remaining graph is not connected,*

Trees defined in many ways

Theorem 2.4.2 (Trees, Berge)

Let $G = (V, E)$ be an undirected graph with $|V| = n > 2$. Then each of the following properties are equivalent and each can be used to define when G is a tree:

- *G is connected and has no cycles*
- *G has $n - 1$ edges and has no cycles,*
- *G is connected and contains exactly $n - 1$ edges,*
- *G has no cycles. Exactly one cycle created if edge added to G.*
- *G is connected, and if any edge is removed, the remaining graph is not connected,*
- *Every pair of vertices of G is connected by one unique path.*

Trees defined in many ways

Theorem 2.4.2 (Trees, Berge)

Let $G = (V, E)$ be an undirected graph with $|V| = n > 2$. Then each of the following properties are equivalent and each can be used to define when G is a tree:

- *G is connected and has no cycles*
- *G has $n - 1$ edges and has no cycles,*
- *G is connected and contains exactly $n - 1$ edges,*
- *G has no cycles. Exactly one cycle created if edge added to G.*
- *G is connected, and if any edge is removed, the remaining graph is not connected,*
- *Every pair of vertices of G is connected by one unique path.*
- *G can be generated as follows: Start with v, repeatedly choose next vertex, and connect it with edge to exactly one previous vertex.*

Trees - and inference

- Size of any maxclique in tree is two. Any set $S \subset V(T)$ with $|S| > 2$ induces a forest.

Trees - and inference

- Size of any maxclique in tree is two. Any set $S \subset V(T)$ with $|S| > 2$ induces a forest.
- Any $p \in \mathcal{F}(T, \mathcal{M}^{(\mathrm{f})})$ has factors of size at most two.
- This has important consequences for inference.

Trees - and inference

- Size of any maxclique in tree is two. Any set $S \subset V(T)$ with $|S| > 2$ induces a forest.
- Any $p \in \mathcal{F}(T, \mathcal{M}^{(\mathsf{f})})$ has factors of size at most two.
- This has important consequences for inference.
- A *chain* is a set of nodes connected in succession.

- A chain is a tree but not vice verse

Trees - and inference

- Size of any maxclique in tree is two. Any set $S \subset V(T)$ with $|S| > 2$ induces a forest.
- Any $p \in \mathcal{F}(T, \mathcal{M}^{(\mathrm{f})})$ has factors of size at most two.
- This has important consequences for inference.
- A *chain* is a set of nodes connected in succession.

- A chain is a tree but not vice verse
- If p factors w.r.t. a chain then

$$p(x) = \prod_{i=1}^{N-1} \psi_{i,i+1}(x_i, x_{i+1}) \tag{2.8}$$

Trees - and inference

- Size of any maxclique in tree is two. Any set $S \subset V(T)$ with $|S| > 2$ induces a forest.
- Any $p \in \mathcal{F}(T, \mathcal{M}^{(\mathrm{f})})$ has factors of size at most two.
- This has important consequences for inference.
- A *chain* is a set of nodes connected in succession.

- A chain is a tree but not vice verse
- If p factors w.r.t. a chain then

$$p(x) = \prod_{i=1}^{N-1} \psi_{i,i+1}(x_i, x_{i+1}) \tag{2.8}$$

- Suppose we wish to compute $p(x_3, x_4)$. then

$$p(x_3, x_4) = \sum_{x_1} \sum_{x_2} \sum_{x_5} \sum_{x_6} \cdots \sum_{x_N} p(x_1, x_2, \dots, x_N) \tag{2.9}$$

Trees - and inference

- Let $r \triangleq \mathsf{D}_{X_i}\ \forall i$
- This requires $O(r^N)$ ops, as in:

---

foreach $(x_3, x_4) \in \mathcal{D}_{X_3} \times \mathcal{D}_{X_4}$ **do**
  Compute $\sum_{x_1} \sum_{x_2} \sum_{x_5} \sum_{x_6} \cdots \sum_{x_N} p(x_1, x_2, \ldots, x_N)$

Trees - and inference

- Let $r \triangleq \mathrm{D}_{X_i} \; \forall i$
- This requires $O(r^N)$ ops, as in:

---

1 **foreach** $(x_3, x_4) \in \mathcal{D}_{X_3} \times \mathcal{D}_{X_4}$ **do**

2 Compute $\sum_{x_1} \sum_{x_2} \sum_{x_5} \sum_{x_6} \cdots \sum_{x_N} p(x_1, x_2, \dots, x_N)$

- Very wasteful!! Does not take advantage of the distributive law in $\mathbb{R}$ (i.e., $ab + ac = a(b + c)$).

$$\sum_{x_1, x_2, \dots, x_N} \left(\left(\prod_{c \in \text{factors not involving } x_i} \psi_c \right) \left(\prod_{c \in \text{factors involving } x_i} \psi_c \right) \right)$$

$$= \sum_{x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_N} \left(\prod_{c \in \text{factors not involving } x_i} \psi_c \right) \sum_{x_i} \left(\prod_{c \in \text{factors involving } x_i} \psi_c \right)$$

Trees - and inference

- We can exploit this property - move sum as far to right as possible. Take the case where $N = 5$, for example:

$$p(x_3, x_4) = \sum_{x_2} \sum_{x_5} \psi_{3,4}(x_3, x_4)\psi_{4,5}(x_4, x_5)\psi_{2,3}(x_2, x_3) \underbrace{\sum_{x_1} \psi_{1,2}(x_1, x_2)}_{\phi_{\cancel{1},2}(x_2)}$$

$$= \sum_{x_2} \sum_{x_5} \psi_{3,4}(x_3, x_4)\psi_{4,5}(x_4, x_5)\psi_{2,3}(x_2, x_3)\phi_{\cancel{1},2}(x_2) \quad (2.10)$$

where $\phi_{\cancel{1},2}(x_2)$ is a function of x_2 only. The notation $\cancel{1}$ indicates that x_1 has been summed away.

Trees - and inference

- We can exploit this property - move sum as far to right as possible. Take the case where $N = 5$, for example:

$$p(x_3, x_4) = \sum_{x_2} \sum_{x_5} \psi_{3,4}(x_3, x_4)\psi_{4,5}(x_4, x_5)\psi_{2,3}(x_2, x_3) \underbrace{\sum_{x_1} \psi_{1,2}(x_1, x_2)}_{\phi_{\cancel{1},2}(x_2)}$$

$$= \sum_{x_2} \sum_{x_5} \psi_{3,4}(x_3, x_4)\psi_{4,5}(x_4, x_5)\psi_{2,3}(x_2, x_3)\phi_{\cancel{1},2}(x_2) \quad (2.10)$$

  where $\phi_{\cancel{1},2}(x_2)$ is a function of x_2 only. The notation $\cancel{1}$ indicates that x_1 has been summed away.

- Node x_1 has been "eliminated" since once marginalized, it never appears in future summations.

Trees - and inference

- We can exploit this property - move sum as far to right as possible. Take the case where $N = 5$, for example:

$$p(x_3, x_4) = \sum_{x_2} \sum_{x_5} \psi_{3,4}(x_3, x_4)\psi_{4,5}(x_4, x_5)\psi_{2,3}(x_2, x_3) \underbrace{\sum_{x_1} \psi_{1,2}(x_1, x_2)}_{\phi_{\cancel{1},2}(x_2)}$$

$$= \sum_{x_2} \sum_{x_5} \psi_{3,4}(x_3, x_4)\psi_{4,5}(x_4, x_5)\psi_{2,3}(x_2, x_3)\phi_{\cancel{1},2}(x_2) \quad (2.10)$$

  where $\phi_{\cancel{1},2}(x_2)$ is a function of x_2 only. The notation $\cancel{1}$ indicates that x_1 has been summed away.
- Node x_1 has been "eliminated" since once marginalized, it never appears in future summations.
- Computing $\phi_{\cancel{1},2}(x_2)$ costs only $O(r^2)$.

Trees - and inference

- We have expression that does not involve x_1, lets next sum away x_2.

$$p(x_3, x_4) = \sum_{x_5} \psi_{3,4}(x_3, x_4)\psi_{4,5}(x_4, x_5) \underbrace{\sum_{x_2} \psi_{2,3}(x_2, x_3)\phi_{\cancel{1},2}(x_2)}_{\phi_{\cancel{1},\cancel{2},3}(x_3)} \tag{2.11}$$

$$= \sum_{x_5} \psi_{3,4}(x_3, x_4)\psi_{4,5}(x_4, x_5)\phi_{\cancel{1},\cancel{2},3}(x_3) \tag{2.12}$$

Trees - and inference

- We have expression that does not involve x_1, lets next sum away x_2.

$$p(x_3, x_4) = \sum_{x_5} \psi_{3,4}(x_3, x_4)\psi_{4,5}(x_4, x_5) \underbrace{\sum_{x_2} \psi_{2,3}(x_2, x_3)\phi_{\cancel{1},2}(x_2)}_{\phi_{\cancel{1},\cancel{2},3}(x_3)} \tag{2.11}$$

$$= \sum_{x_5} \psi_{3,4}(x_3, x_4)\psi_{4,5}(x_4, x_5)\phi_{\cancel{1},\cancel{2},3}(x_3) \tag{2.12}$$

- $\phi_{\cancel{1},\cancel{2},3}(x_3)$ - both x_1 and x_2 are eliminated, only function of x_3.

Trees - and inference

- We have expression that does not involve x_1, lets next sum away x_2.

$$p(x_3, x_4) = \sum_{x_5} \psi_{3,4}(x_3, x_4)\psi_{4,5}(x_4, x_5) \underbrace{\sum_{x_2} \psi_{2,3}(x_2, x_3)\phi_{\cancel{1},2}(x_2)}_{\phi_{\cancel{1},\cancel{2},3}(x_3)} \tag{2.11}$$

$$= \sum_{x_5} \psi_{3,4}(x_3, x_4)\psi_{4,5}(x_4, x_5)\phi_{\cancel{1},\cancel{2},3}(x_3) \tag{2.12}$$

- $\phi_{\cancel{1},\cancel{2},3}(x_3)$ - both x_1 and x_2 are eliminated, only function of x_3.
- Again, only $O(r^2)$

Trees - and inference

- Next, we sum away (eliminate) x_5 (moving sums in as far as possible).

$$p(x_3, x_4) = \psi_{3,4}(x_3, x_4)\phi_{\cancel{1},\cancel{2},3}(x_3) \underbrace{\sum_{x_5} \psi_{4,5}(x_4, x_5)}_{\phi_{\cancel{5},4}(x_4)} \tag{2.13}$$

$$= p(x_3, x_4) = \psi_{3,4}(x_3, x_4)\phi_{\cancel{1},\cancel{2},3}(x_3)\phi_{\cancel{5},4}(x_4) \tag{2.14}$$

Trees - and inference

- Next, we sum away (eliminate) x_5 (moving sums in as far as possible).

$$p(x_3, x_4) = \psi_{3,4}(x_3, x_4)\phi_{\cancel{1},\cancel{2},3}(x_3) \underbrace{\sum_{x_5} \psi_{4,5}(x_4, x_5)}_{\phi_{\cancel{5},4}(x_4)} \tag{2.13}$$

$$= p(x_3, x_4) = \psi_{3,4}(x_3, x_4)\phi_{\cancel{1},\cancel{2},3}(x_3)\phi_{\cancel{5},4}(x_4) \tag{2.14}$$

- Again, only $O(r^2)$ to produce $\phi_{\cancel{5},4}(x_4)$

Trees - and inference

- Next, we sum away (eliminate) x_5 (moving sums in as far as possible).

$$p(x_3, x_4) = \psi_{3,4}(x_3, x_4)\phi_{\cancel{1},\cancel{2},3}(x_3) \underbrace{\sum_{x_5} \psi_{4,5}(x_4, x_5)}_{\phi_{\cancel{5},4}(x_4)} \tag{2.13}$$

$$= p(x_3, x_4) = \psi_{3,4}(x_3, x_4)\phi_{\cancel{1},\cancel{2},3}(x_3)\phi_{\cancel{5},4}(x_4) \tag{2.14}$$

- Again, only $O(r^2)$ to produce $\phi_{\cancel{5},4}(x_4)$
- Entire computation is $O(r^2)$
- Length N chain can be done in $O(Nr^2)$.

Trees - and inference

- Get $O(r^2)$ if we eliminate variables in order $(1, 2, 5)$

Trees - and inference

- Get $O(r^2)$ if we eliminate variables in order $(1, 2, 5)$
- Other orders also have $O(r^2)$, such as $(5, 1, 2)$ or $(1, 5, 2)$ and would still obtain $p(x_3, x_4)$.

Trees - and inference

- Get $O(r^2)$ if we eliminate variables in order $(1, 2, 5)$
- Other orders also have $O(r^2)$, such as $(5, 1, 2)$ or $(1, 5, 2)$ and would still obtain $p(x_3, x_4)$.
- Not all orders have same efficiency, consider order $(2, 1, 5)$.

$$p(x_3, x_4) = \sum_{x_1, x_5} \psi_{3,4}(x_3, x_4)\psi_{4,5}(x_4, x_5) \underbrace{\sum_{x_2} \psi_{1,2}(x_1, x_2)\psi_{2,3}(x_2, x_3)}_{\phi_{\cancel{2},1,3}(x_1, x_3)} \tag{2.15}$$

$$= \sum_{x_5} \psi_{3,4}(x_3, x_4)\psi_{4,5}(x_4, x_5) \sum_{x_1} \phi_{\cancel{2},1,3}(x_1, x_3) \tag{2.16}$$

$$= \psi_{3,4}(x_3, x_4)\phi_{\cancel{2},\cancel{1},3}(x_3) \sum_{x_5} \psi_{4,5}(x_4, x_5) \tag{2.17}$$

$$= \psi_{3,4}(x_3, x_4)\phi_{\cancel{2},\cancel{1},3}(x_3)\phi_{\cancel{5},4}(x_4) \tag{2.18}$$

Trees - and inference

- Problem: Sum over x_2 in Eq. 2.15 has cost $O(r^3)$. Total complexity is $O(r^3)$ which is unboundedly worse than $O(r^2)$.

Trees - and inference

- Problem: Sum over x_2 in Eq. 2.15 has cost $O(r^3)$. Total complexity is $O(r^3)$ which is unboundedly worse than $O(r^2)$.
- Some orders inextricably couple together factors, others don't.

Trees - and inference

- Problem: Sum over x_2 in Eq. 2.15 has cost $O(r^3)$. Total complexity is $O(r^3)$ which is unboundedly worse than $O(r^2)$.
- Some orders inextricably couple together factors, others don't.
- How do we ensure the best (fastest) elimination order? Graph tells us.

Trees - and inference

- Problem: Sum over x_2 in Eq. 2.15 has cost $O(r^3)$. Total complexity is $O(r^3)$ which is unboundedly worse than $O(r^2)$.
- Some orders inextricably couple together factors, others don't.
- How do we ensure the best (fastest) elimination order? Graph tells us.
- Key Problem: there **exist no functions** $g(a)$ and $h(c)$ that constitute a factorization of a sum as in:

$$g(a)h(c) = \sum_b f_1(a,b)f_2(b,c) \tag{2.19}$$

Trees - and inference

- Problem: Sum over x_2 in Eq. 2.15 has cost $O(r^3)$. Total complexity is $O(r^3)$ which is unboundedly worse than $O(r^2)$.
- Some orders inextricably couple together factors, others don't.
- How do we ensure the best (fastest) elimination order? Graph tells us.
- Key Problem: there **exist no functions** $g(a)$ and $h(c)$ that constitute a factorization of a sum as in:

$$g(a)h(c) = \sum_b f_1(a,b) f_2(b,c) \tag{2.19}$$

- In general, for disjoint variables $A, B, C \subseteq V$, the function

$$f(x_A, x_C) = \sum_{x_B} f_1(x_A, x_B) f_2(x_B, x_C) \tag{2.20}$$

does not factor, exists no g, h such that $f(x_A, x_C) = g(x_A)h(x_C)$.

Elimination

- Existence of $f_1(x_A, x_B)$ suggests that $G[A \cup B]$ should be a clique, and existence of $f_2(x_B, x_C)$ suggests $G[B \cup C]$ should be a clique.
- After summation, existence of $f(x_A, x_C)$ suggests that $G[A \cup C]$ should also be a clique *(if it is not already)*.
- Graph-theoretic operation for eliminating a variable in a graph:

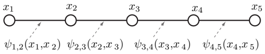
Definition 2.4.3

Elimination: To *eliminate* a node $v \in V$ in an undirected graph G, we first connect all neighbors of v and then remove v and all v's adjacent edges from the graph.

- Once eliminated, former neighbors of v form a clique.
- Additional edges added (if any) are called *fill-in* edges. We'll use $F \subseteq V \times V$ for these.

Example elimination on chain

Reconstituted graphs also given

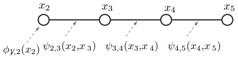


Eliminate x_1

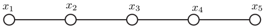
$$\sum_{x_1} \psi_{1,2}(x_1,x_2)\psi_{2,3}(x_2,x_3)\psi_{3,4}(x_3,x_4)\psi_{4,5}(x_4,x_5)$$

$$= \psi_{2,3}(x_2,x_3)\psi_{3,4}(x_3,x_4)\psi_{4,5}(x_4,x_5)\sum_{x_1}\psi_{1,2}(x_1,x_2)$$

$$= \psi_{2,3}(x_2,x_3)\psi_{3,4}(x_3,x_4)\psi_{4,5}(x_4,x_5)\phi_{\cancel{1},2}(x_2)$$



Reconstituted Graph: No fill-in edges

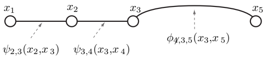


Eliminate x_4

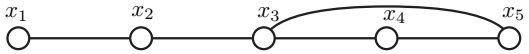
$$\sum_{x_4} \psi_{1,2}(x_1,x_2)\psi_{2,3}(x_2,x_3)\psi_{3,4}(x_3,x_4)\psi_{4,5}(x_4,x_5)$$

$$= \psi_{1,2}(x_1,x_2)\psi_{2,3}(x_2,x_3)\sum_{x_4}\psi_{3,4}(x_3,x_4)\psi_{4,5}(x_4,x_5)$$

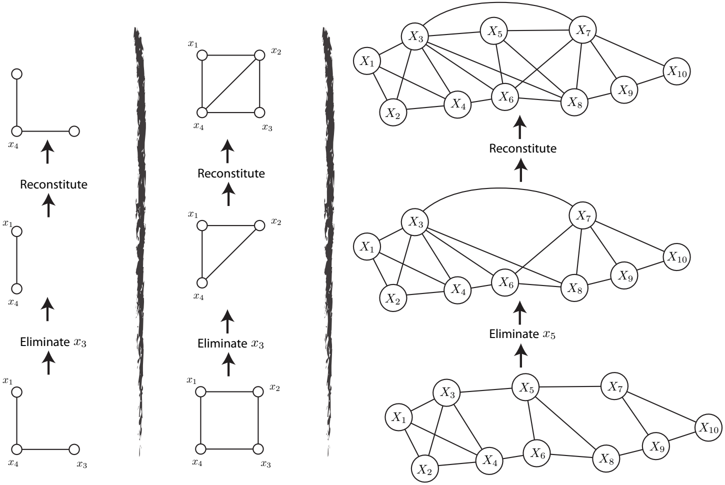
$$= \psi_{2,3}(x_2,x_3)\psi_{3,4}(x_3,x_4)\ \phi_{\cancel{4},3,5}(x_3,x_5)$$



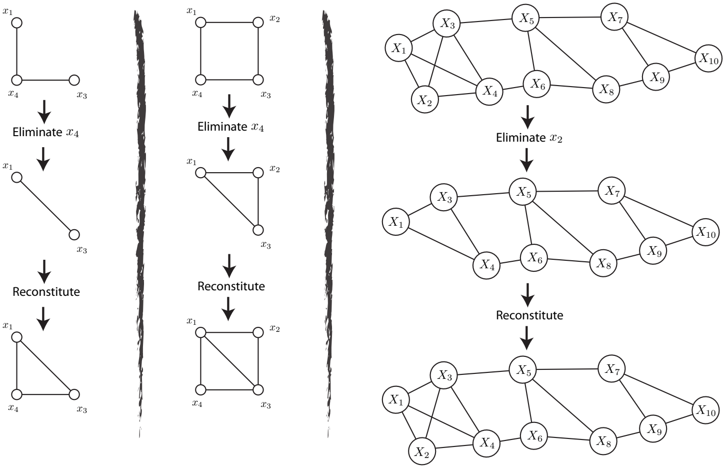
Reconstituted Graph: One fill-in edge



Example elimination on graphs



Example elimination on graphs



Elimination

- Those variables inextricably coupled after computational elimination are exactly those variables connected together by edge in graphical elimination

Elimination

- Those variables inextricably coupled after computational elimination are exactly those variables connected together by edge in graphical elimination
- Those newly coupled variables can only be represented by a single factor

Elimination

- Those variables inextricably coupled after computational elimination are exactly those variables connected together by edge in graphical elimination
- Those newly coupled variables can only be represented by a single factor
- When forming new factor

$$f(x_A, x_C) = \sum_{x_b} f_1(x_A, x_b) f_2(x_b, x_C) \tag{2.21}$$

Computation is $O(r^{|A \cup C|+1})$ for scalar sum over x_b, exponential in size of resulting coupling.

Elimination

- Those variables inextricably coupled after computational elimination are exactly those variables connected together by edge in graphical elimination
- Those newly coupled variables can only be represented by a single factor
- When forming new factor

$$f(x_A, x_C) = \sum_{x_b} f_1(x_A, x_b) f_2(x_b, x_C) \tag{2.21}$$

  Computation is $O(r^{|A \cup C|+1})$ for scalar sum over x_b, exponential in size of resulting coupling.
- Graphically, the sets A, C correspond to the nodes that are neighbors of $b \in V$ at the time of elimination.

Elimination

- Those variables inextricably coupled after computational elimination are exactly those variables connected together by edge in graphical elimination
- Those newly coupled variables can only be represented by a single factor
- When forming new factor

$$f(x_A, x_C) = \sum_{x_b} f_1(x_A, x_b) f_2(x_b, x_C) \tag{2.21}$$

  Computation is $O(r^{|A \cup C|+1})$ for scalar sum over x_b, exponential in size of resulting coupling.
- Graphically, the sets A, C correspond to the nodes that are neighbors of $b \in V$ at the time of elimination.
- So neighbors of a node determine the (exponential) cost of doing a variable elimination.

Variable Elimination

- Therefore, goal is to find node $v \in V$ to eliminate that
  1. has only one neighbor (so that no new edges are added), or
  2. have neighbors that are already connected so that eliminating the node will not add any new edges.
- If we cannot find a node $v \in V$ that satisfies these two goals, we must accept some fill-in $F \neq \emptyset$ will occur.
- Computationally, we might as well have had those additional edges in the graph to begin with (those edges are inevitable, so can add them beforehand).
- If $F \neq \emptyset$, like we are solving problem for more general family. I.e., rather than $\mathcal{F}(G, \mathcal{M}^{(\mathrm{f})})$, where $G = (V, E)$, we solve it for $\mathcal{F}(G', \mathcal{M}^{(\mathrm{f})})$ where $G' = (V, E \cup F)$ with $F \subseteq V \times V$.

Family for more edges

- In fact, adding *any* set of edges F increases the family. We have that:

Theorem 2.4.4

Let $G = (V, E)$ be a graph with corresponding MRF family $\mathcal{F}(G, \mathcal{M}^{(f)})$. Let $F \subseteq V \times V$ be any *set of node pairs. Form a new graph $G_F = (V, E \cup F)$ by adding the pairs of nodes as edges to G to obtain G_F. Then $\mathcal{F}(G, \mathcal{M}^{(f)}) \subseteq \mathcal{F}(G_F, \mathcal{M}^{(f)})$.*

Proof.

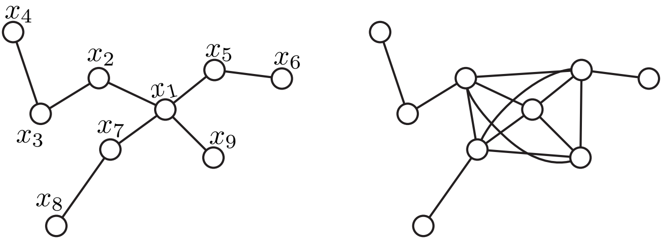
Take any $p \in \mathcal{F}(G, \mathcal{M}^{(\mathrm{f})})$. p factors w.r.t. the cliques in G. Take any clique C in G. Since G_F only has additional edges relative to G, C is a clique in G_F also but might be part of a larger clique. Therefore, any clique factor in G is either preserved in G_F or can be part of a larger factor in G_F, so p factors w.r.t. the cliques in G_F. □

Family for more edges

- Therefore, we are free to add these inevitable edges F to $G = (V, E)$, increase the family, and then solve the inference problem for this more general family.
- In chain case, there was an order of the nodes so that $F = \emptyset$, at each elimination step, the elimination node had only 1 neighbor.
- In chain case, the poor order eliminated a node that had two neighbors, leading to $O(r^3)$.
- Chains are such that there is an obvious "perfect" elimination order (always start at one of the ends).
- What about trees?

Trees and elimination

- Suppose we wish $p(x_3, x_4)$ for a $p \in \mathcal{F}(T, \mathcal{M}^{(f)})$ with $T = (V, E)$ being the following:



- Suppose we start with x_1

$$\ldots \sum_{x_1} \psi_{1,2}(x_1, x_2)\psi_{1,5}(x_1, x_5)\psi_{1,7}(x_1, x_7)\psi_{1,9}(x_1, x_9)$$
$$= \phi_{\cancel{1},2,5,7,9}(x_2, x_5, x_7, x_9) \tag{2.22}$$

any further computation results in $O(r^5)$ — x_1 a poor vertex to eliminate first.

Trees and elimination

- On the other hand, consider the elimination order $(6, 5, 9, 8, 7, 1, 2)$.
- Re summing: at each step, moving sum to right yields only factors that involve at most two variables $\rightarrow O(r^2)$ at each step.
- Re graph elimination: each node at point of elimination has only one neighbor, no fill-in, clique size is 2.
- A *leaf node* (or *pendant* node) in a tree is a node that has only one neighbor
- Eliminating leaf nodes is good, and trees always have them.

Trees and leaf nodes

Lemma 2.4.5

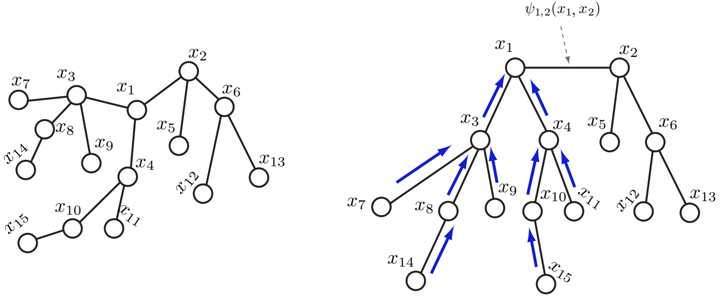
A tree with more than one node always has at least two leaf nodes.

Proof.

Obviously true for $|V| = n = 2$ nodes. Assume true for $n-1$ nodes and consider a tree with n nodes. The tree must have at least one leaf-node since if all nodes had two or more edges, we could find a cycle by traversing the nodes along the edges and marking the edges along the way — each node we encounter will either have an unmarked edge to allow the traversal to continue, or will have only marked edges implying the existence of a cycle, and eventually this latter condition will be reached since there are a finite number of nodes. The tree with $n-1$ nodes induced by removing this leaf-node must itself have two leaf-nodes by induction, and at least one of those leaf-nodes is retained when adding back in the node to form the n-node tree. □

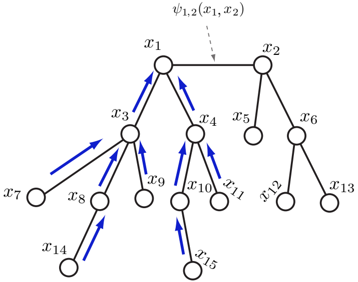
Morphing from elimination to belief propagation

- elimination can be seen as a message passing scheme on a graph
- Tree on left, goal is to produce computation for $p(x_1, x_2)$. We rooted at edge $(1, 2)$ on the right



- blue arrows show elimination steps starting at leaf nodes and continuing until we have reached the root.

Computations for this



$$\phi_{\cancel{1}4,8}(x_8) = \sum_{x_{14}} \psi_{8,14}(x_8, x_{14}) \tag{2.23a}$$

$$\phi_{\cancel{7},3}(x_3) = \sum_{x_7} \psi_{7,3}(x_7, x_3) \tag{2.23b}$$

$$\phi_{\cancel{8},\cancel{1}4,3}(x_3) = \sum_{x_8} \psi_{8,3}(x_8, x_3)\phi_{\cancel{1}4,8}(x_8) \tag{2.23c}$$

$$\phi_{\cancel{9},3}(x_3) = \sum_{x_9} \phi_{9,3}(x_9, x_3) \tag{2.23d}$$

$$\phi_{\cancel{7},\cancel{1}4,\cancel{8},\cancel{9},\cancel{3},1}(x_1) = \sum_{x_3} \psi_{1,3}(x_1, x_3)\phi_{\cancel{7},3}(x_3)\phi_{\cancel{8},\cancel{1}4,3}(x_3)\phi_{\cancel{9},3}(x_3) \tag{2.23e}$$

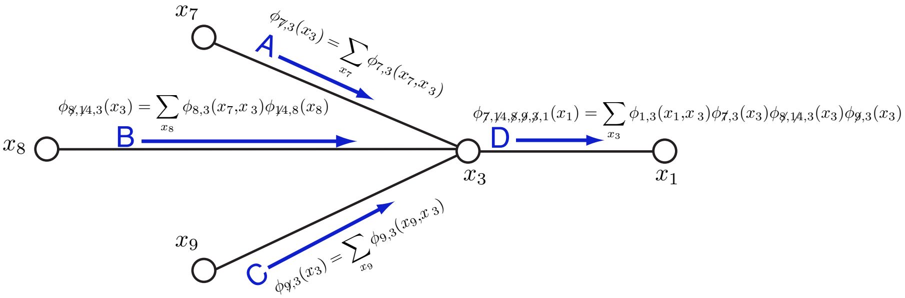
$$\phi_{\cancel{1}5,10}(x_{10}) = \sum_{x_{15}} \psi_{10,15}(x_{10}, x_{15}) \tag{2.23f}$$

$$\phi_{\cancel{1}5,\cancel{1}0,4}(x_4) = \sum_{x_{10}} \psi_{4,10}(x_4, x_{10})\phi_{\cancel{1}5,10}(x_{10}) \tag{2.23g}$$

$$\phi_{\cancel{1}1,4}(x_4) = \sum_{x_{11}} \psi_{4,11}(x_4, x_{11}) \tag{2.23h}$$

$$\phi_{\cancel{1}0,\cancel{1}1,\cancel{1}5,\cancel{4},1}(x_1) = \sum_{x_4} \psi_{1,4}(x_1, x_4)\phi_{\cancel{1}5,\cancel{1}0,4}(x_4)\phi_{\cancel{1}1,4}(x_4) \tag{2.23i}$$

Expanded messages on tree



- Consider Equation (2.23b), Equation (2.23c), Equation (2.23d), and Equation (2.23e). from previous slide.
- lets view these computations as a form of "message" being sent over a graph.
- Expanded graph showing the incoming messages into node x_3 from nodes x_7, x_8, and x_9 and then x_3's message sent out to its destination parent x_1.

Elimination $\rightarrow$ message passing

- Each node receives a "message" from children in rooted tree, once received enough "messages" can send a "message" to parent.
- General, node i may send message to parent j when i has received message from all of i's children
- at that point, i has become a leaf node in the tree (all children eliminated)
- The parent is chosen arbitrarily (it depends on root).
- There is a general pattern that is true regardless of root designation.

Message passing protocol

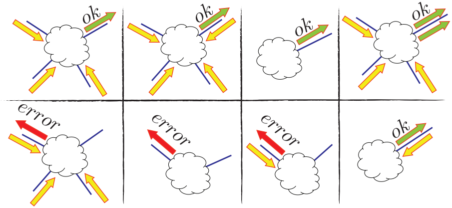
Definition 2.5.1

Message passing protocol (MPP): A message may be sent from node i to a neighbor node j only when node i has received a message from all its other neighbors besides j.

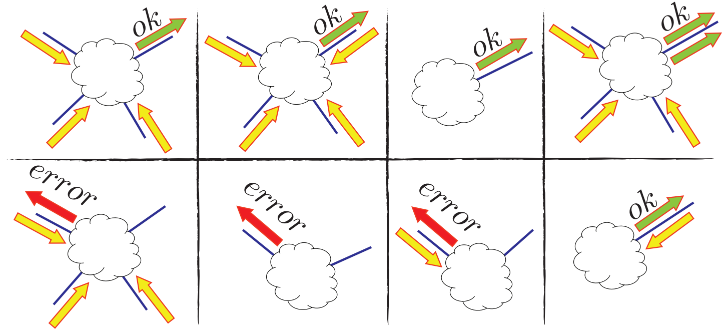
- Notationally, if $i \to j$ indicates a message from i to j, then the protocol may be written as $i \to j$ only when $\forall k \in \delta(i) \setminus \{j\}, k \to i$.
- If MPP is followed but otherwise the ordering of the messages is arbitrary, then we are guaranteed that the end result will be the correct marginal. That is, the protocol specifies only a *partial* (rather than a total) order on messages.

Message passing protocol

- Top two examples on right show that green outgoing message is ok, obeys MPP
- Bottom two examples on right violate MPP.



Message passing protocol examples



- Examples of valid and invalid messages. Yellow arrows correspond to incoming messages. Green outgoing arrows correspond to messages that obey MPP, and red outgoing arrows are messages that disobey MPP.
- Note that the 2nd from left example on top row corresponds to what happens at the root of a tree.

Better notation

- Notation is unwieldy. Rather than keep track of entire history, as in $\phi_{\cancel{1}5,\cancel{1}0,4}(x_4)$, use notation that only indicates neighbors in a message
- We use $\mu_{i\to j}(x_j)$ to indicate a message coming from node i going to node j along the edge (i,j) and which is a function only of x_j (since x_i has been eliminated).
- Before

$$\phi_{\cancel{1}4,8}(x_8) = \sum_{x_{14}} \psi_{8,14}(x_8, x_{14}) \tag{2.24}$$

After

$$\mu_{14\to 8}(x_8) = \sum_{x_{14}} \psi_{8,14}(x_8, x_{14}) \tag{2.25}$$

- Before

$$\phi_{\cancel{7},\cancel{1}4,\cancel{8},\cancel{9},\cancel{3},1}(x_1) = \sum_{x_3} \psi_{1,3}(x_1, x_3)\phi_{\cancel{7},3}(x_3)\phi_{\cancel{8},\cancel{1}4,3}(x_3)\phi_{\cancel{9},3}(x_3) \tag{2.26}$$

After

$$\mu_{3\to 1}(x_1) = \sum_{x_3} \psi_{1,3}(x_1, x_3)\mu_{7\to 3}(x_3)\mu_{8\to 3}(x_3)\mu_{9\to 3}(x_3) \tag{2.27}$$

Generic form of message

$$\mu_{i\to j}(x_j) = \sum_{x_i} \left(\psi_{i,j}(x_i, x_j) \prod_{k\in\delta(i)\setminus\{j\}} \mu_{k\to i}(x_i) \right) \tag{2.28}$$

Message is of form:

1. First, collect messages from all neighbors of i other than j,
2. next, incorporate these incoming messages by multiplying them in along with the factor $\psi_{i,j}(x_i, x_j)$,
3. the factor $\psi_{i,j}(x_i, x_j)$ relates x_i and x_j, and can be seen as a representation of a "communications channel" relating how the information x_i transforms into the information in x_j, thus motivating the terminology of a "message", and
4. then finally marginalizing away x_i thus yielding the desired message to be delivered at the destination node x_j.

Multiple Tree Queries

- Rather than one S we may have $\{S_1, S_2, \ldots, S_k\} = \mathcal{S}$ and wish to compute $p(x_{S_i})$ for all $i \in \{1, 2, \ldots, k\}$. Ex: all cliques/edges.

Multiple Tree Queries

- Rather than one S we may have $\{S_1, S_2, \ldots, S_k\} = \mathcal{S}$ and wish to compute $p(x_{S_i})$ for all $i \in \{1, 2, \ldots, k\}$. Ex: all cliques/edges.
- Naive way: Do the above k times leading to $O(kNr^2)$ computation.

Multiple Tree Queries

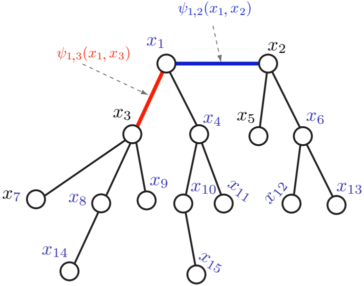
- Rather than one S we may have $\{S_1, S_2, \ldots, S_k\} = \mathcal{S}$ and wish to compute $p(x_{S_i})$ for all $i \in \{1, 2, \ldots, k\}$. Ex: all cliques/edges.
- Naive way: Do the above k times leading to $O(kNr^2)$ computation.
- We can reduce this to $O(Nr^2)$ when S_i are cliques by removing redundant computations.

Multiple Tree Queries

- Rather than one S we may have $\{S_1, S_2, \ldots, S_k\} = \mathcal{S}$ and wish to compute $p(x_{S_i})$ for all $i \in \{1, 2, \ldots, k\}$. Ex: all cliques/edges.
- Naive way: Do the above k times leading to $O(kNr^2)$ computation.
- We can reduce this to $O(Nr^2)$ when S_i are cliques by removing redundant computations.
- this is done using dynamic programming - re-use already computed partial solutions to one problem to help solve other problems, and vice verse.

Multiple Tree Queries

- Rather than one S we may have $\{S_1, S_2, \ldots, S_k\} = \mathcal{S}$ and wish to compute $p(x_{S_i})$ for all $i \in \{1, 2, \ldots, k\}$. Ex: all cliques/edges.
- Naive way: Do the above k times leading to $O(kNr^2)$ computation.
- We can reduce this to $O(Nr^2)$ when S_i are cliques by removing redundant computations.
- this is done using dynamic programming - re-use already computed partial solutions to one problem to help solve other problems, and vice verse.
- Example: compute both $p(x_1, x_2)$ and $p(x_1, x_3)$ as before.



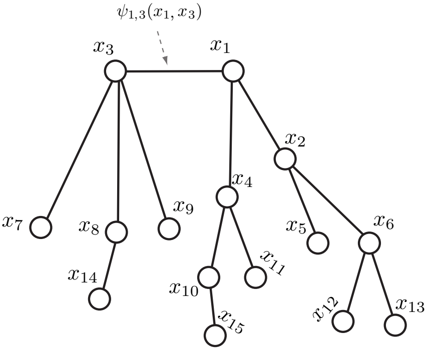
Multiple Tree Queries

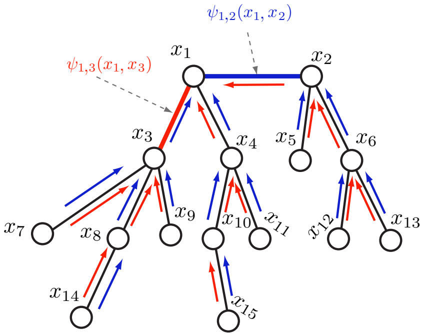
- Variable elimination
- For $p(x_1, x_2)$, the variable elimination ordering $(14, 7, 8, 9, 15, 10, 11, 4, 12, 13, 5, 6, 3)$ would suffice
- 13 messages: $\mu_{14\to8}(x_8)$, $\mu_{7\to3}(x_3)$, $\mu_{8\to3}(x_3)$, $\mu_{9\to3}(x_3)$, $\mu_{15\to10}(x_{10})$, $\mu_{10\to4}(x_4)$, $\mu_{11\to4}(x_4)$, $\mu_{4\to1}(x_1)$, $\mu_{12\to6}(x_6)$, $\mu_{13\to6}(x_6)$, $\mu_{5\to2}(x_2)$, $\mu_{6\to2}(x_2)$, and $\mu_{3\to1}(x_1)$.
- For $p(x_1, x_3)$, the variable ordering $(14, 7, 8, 9, 15, 10, 11, 4, 12, 13, 5, 6, 2)$ would suffice
- messages: $\mu_{14\to8}(x_8)$, $\mu_{7\to3}(x_3)$, $\mu_{8\to3}(x_3)$, $\mu_{9\to3}(x_3)$, $\mu_{15\to10}(x_{10})$, $\mu_{10\to4}(x_4)$, $\mu_{11\to4}(x_4)$, $\mu_{4\to1}(x_1)$, $\mu_{12\to6}(x_6)$, $\mu_{13\to6}(x_6)$, $\mu_{5\to2}(x_2)$, $\mu_{6\to2}(x_2)$, and $\mu_{2\to1}(x_1)$.

Multiple Tree Queries

- Variable elimination
- For $p(x_1, x_2)$, the variable elimination ordering $(14, 7, 8, 9, 15, 10, 11, 4, 12, 13, 5, 6, 3)$ would suffice
- 13 messages: $\mu_{14\to 8}(x_8)$, $\mu_{7\to 3}(x_3)$, $\mu_{8\to 3}(x_3)$, $\mu_{9\to 3}(x_3)$, $\mu_{15\to 10}(x_{10})$, $\mu_{10\to 4}(x_4)$, $\mu_{11\to 4}(x_4)$, $\mu_{4\to 1}(x_1)$, $\mu_{12\to 6}(x_6)$, $\mu_{13\to 6}(x_6)$, $\mu_{5\to 2}(x_2)$, $\mu_{6\to 2}(x_2)$, and $\mu_{3\to 1}(x_1)$.
- For $p(x_1, x_3)$, the variable ordering $(14, 7, 8, 9, 15, 10, 11, 4, 12, 13, 5, 6, 2)$ would suffice
- messages: $\mu_{14\to 8}(x_8)$, $\mu_{7\to 3}(x_3)$, $\mu_{8\to 3}(x_3)$, $\mu_{9\to 3}(x_3)$, $\mu_{15\to 10}(x_{10})$, $\mu_{10\to 4}(x_4)$, $\mu_{11\to 4}(x_4)$, $\mu_{4\to 1}(x_1)$, $\mu_{12\to 6}(x_6)$, $\mu_{13\to 6}(x_6)$, $\mu_{5\to 2}(x_2)$, $\mu_{6\to 2}(x_2)$, and $\mu_{2\to 1}(x_1)$.
- First 12 of variables in each order are identical! Results in marginal $p(x_1, x_2, x_3)$ from which both results are easy.

Multiple Tree Queries





.

- Another look: Left tree rooted at $(1, 3)$, right rooted at $(1, 2)$.
- Red arrows are messages are for $(1, 3)$, blue arrows are messages for $(1, 2)$.
- most messages are the same.

Multiple Tree Queries

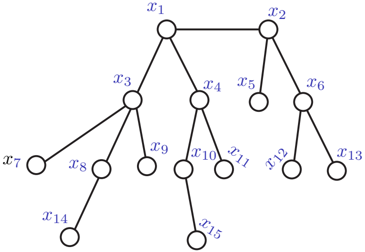
- Amount of available re-use depends on the desired queries

Multiple Tree Queries

- Amount of available re-use depends on the desired queries
- Ex: compute $p(x_8, x_{14})$ and $p(x_6, x_{13})$.

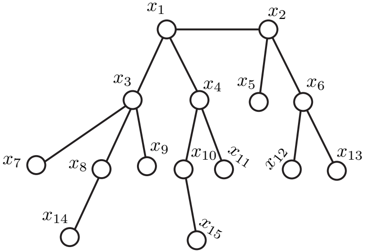
Multiple Tree Queries

- Amount of available re-use depends on the desired queries
- Ex: compute $p(x_8, x_{14})$ and $p(x_6, x_{13})$.
- both may start with order $(7, 9, 15, 10, 11, 4, 5, 12)$, messages: $\mu_{7\to 3}(x_3)$, $\mu_{9\to 3}(x_3)$, $\mu_{15\to 10}(x_{10})$, $\mu_{10\to 4}(x_4)$, $\mu_{11\to 4}(x_4)$, $\mu_{4\to 1}(x_1)$, $\mu_{5\to 2}(x_2)$, and $\mu_{12\to 6}(x_6)$ leaving chain $x_{14}, x_8, x_3, x_1, x_2, x_6, x_{13}$.



Multiple Tree Queries

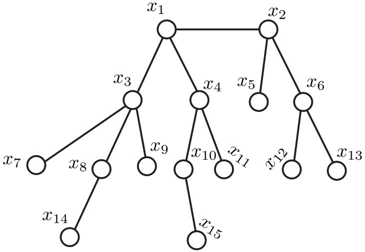
- Amount of available re-use depends on the desired queries
- Ex: compute $p(x_8, x_{14})$ and $p(x_6, x_{13})$.
- both may start with order $(7, 9, 15, 10, 11, 4, 5, 12)$, messages: $\mu_{7\to3}(x_3)$, $\mu_{9\to3}(x_3)$, $\mu_{15\to10}(x_{10})$, $\mu_{10\to4}(x_4)$, $\mu_{11\to4}(x_4)$, $\mu_{4\to1}(x_1)$, $\mu_{5\to2}(x_2)$, and $\mu_{12\to6}(x_6)$ leaving chain $x_{14}, x_8, x_3, x_1, x_2, x_6, x_{13}$.



- remaining messages, from x_{14} to x_{13} and from x_{13} back to x_{14}.

Multiple Tree Queries

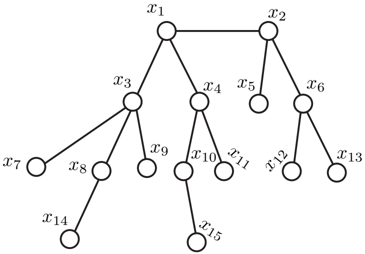
- Amount of available re-use depends on the desired queries
- Ex: compute $p(x_8, x_{14})$ and $p(x_6, x_{13})$.
- both may start with order $(7, 9, 15, 10, 11, 4, 5, 12)$, messages: $\mu_{7\to3}(x_3)$, $\mu_{9\to3}(x_3)$, $\mu_{15\to10}(x_{10})$, $\mu_{10\to4}(x_4)$, $\mu_{11\to4}(x_4)$, $\mu_{4\to1}(x_1)$, $\mu_{5\to2}(x_2)$, and $\mu_{12\to6}(x_6)$ leaving chain $x_{14}, x_8, x_3, x_1, x_2, x_6, x_{13}$.



- remaining messages, from x_{14} to x_{13} and from x_{13} back to x_{14}.
- Chain has least re-use for these queries (since they are on ends)

Multiple Tree Queries

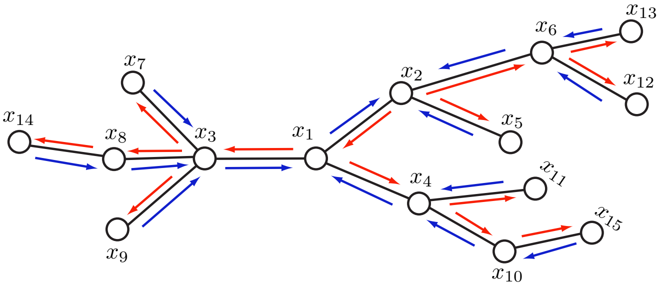
- Amount of available re-use depends on the desired queries
- Ex: compute $p(x_8, x_{14})$ and $p(x_6, x_{13})$.
- both may start with order $(7, 9, 15, 10, 11, 4, 5, 12)$, messages: $\mu_{7\to3}(x_3)$, $\mu_{9\to3}(x_3)$, $\mu_{15\to10}(x_{10})$, $\mu_{10\to4}(x_4)$, $\mu_{11\to4}(x_4)$, $\mu_{4\to1}(x_1)$, $\mu_{5\to2}(x_2)$, and $\mu_{12\to6}(x_6)$ leaving chain $x_{14}, x_8, x_3, x_1, x_2, x_6, x_{13}$.



- remaining messages, from x_{14} to x_{13} and from x_{13} back to x_{14}.
- Chain has least re-use for these queries (since they are on ends)
- Still, have saved quite a bit by "trimming" off branches tree relative to naive strategy.

All edge Queries

- As number of queries increases, so does efficiency (queries/message)
- Consider computing $p(x_i, x_j)$ for $(i, j) \in E(G)$.
- Naive case, $N - 1$ edges $O(N^2 r^2)$.
- Smart case, only $O(Nr^2)$ still.
- consider: root tree at all $(i, j) \in E(G)$ in turn
- mark edge with arrow only once (so don't redundantly send message)
- result is each edge has two arrows in each direction



All edge Queries

- When done, each edge $(i, j) \in E(G)$ is now in possession of $\psi_{i,j}(x_i, x_j)$ as well as $\mu_{k\to i}(x_i)$ for all $k \in \delta(i) \setminus \{j\}$ as well as $\mu_{k\to j}(x_j)$ for all $k \in \delta(j) \setminus \{i\}$.
- Thus, can compute the marginals

$$p(x_i, x_j) = \psi_{i,j}(x_i, x_j) \prod_{k\in\delta(i)\setminus\{j\}} \mu_{k\to i}(x_i) \prod_{k\in\delta(j)\setminus\{i\}} \mu_{k\to j}(x_j) \quad (2.29)$$

- Overall computation $O(Nr^2)$.

All edge Queries

Theorem 2.5.2

Given a tree $G = (V, E)$ and some $p \in \mathcal{F}(G, \mathcal{M}^{(f)})$, if messages are sent obeying the message passing protocol so that all edges have two messages across them in each direction, then the computation given above will correctly produce all marginals for all edges in $E(G)$.

Proof.

Consider any edge $(i, j) \in E(G)$ and consider rooting the graph at that edge, as described above. Since all messages obey the MPP, the messages correspond to eliminating the variables in an order from leaf to root, which precisely gives $p(x_i, x_j)$. □

All edge queries - algorithm

How do we ensure that all edges have messages in both directions applied, and all in the right order?

All edge queries - algorithm

How do we ensure that all edges have messages in both directions applied, and all in the right order?

- Choose arbitrary root node (node root rather than edge)

All edge queries - algorithm

How do we ensure that all edges have messages in both directions applied, and all in the right order?

- Choose arbitrary root node (node root rather than edge)
- Send messages from leaves up to root

All edge queries - algorithm

How do we ensure that all edges have messages in both directions applied, and all in the right order?

- Choose arbitrary root node (node root rather than edge)
- Send messages from leaves up to root
- Once root has received all messages from children, start sending messages back out to children.

All edge queries - algorithm

How do we ensure that all edges have messages in both directions applied, and all in the right order?

- Choose arbitrary root node (node root rather than edge)
- Send messages from leaves up to root
- Once root has received all messages from children, start sending messages back out to children.
- when done all nodes have all messages, MPP obeyed, and any marginal can be computed.

All edge queries - algorithm

How do we ensure that all edges have messages in both directions applied, and all in the right order?

- Choose arbitrary root node (node root rather than edge)
- Send messages from leaves up to root
- Once root has received all messages from children, start sending messages back out to children.
- when done all nodes have all messages, MPP obeyed, and any marginal can be computed.
- This procedure is formalized by algorithms *collect evidence* and *distribute evidence* as follows

Collect Evidence

Algorithm 1: CollectEvidence($c \to p$)

Input: A rooted tree $G = (V, E)$ with a child node $c \in V$ and its parent $p \in V$.

Result: A message propagated from c to p that obeys the message passing protocol.

foreach $u \in \text{child}(c)$ **do**
    call CollectEvidence($u \to c$)

Compute

$$\mu_{c \to p}(x_p) = \sum_{x_c} \psi_{c,p}(x_c, x_p) \prod_{u \in \text{child}(c)} \mu_{u \to c}(x_c)$$

Distribute Evidence

Algorithm 2: DistributeEvidence($p \to c$)

Input: A rooted tree $G = (V, E)$ with a parent node $p \in V$ and a child $c\,\mathrm{child}(p)$.

Result: A message propagated from p to c that obeys the message passing protocol.

1 Compute

$$\mu_{p \to c}(x_c) = \sum_{x_p} \psi_{p,c}(x_p, x_c) \prod_{u \in \delta(p) \setminus \{c\}} \mu_{u \to p}(x_p)$$

2 **foreach** $u \in \mathrm{child}(c)$ **do**

3 call DistributeEvidence($c \to u$)

Collect/Distribute Evidence

Algorithm 3: CollectDistributeEvidence

Input: A tree graph $G = (V, E)$

Result: All messages propagated between all pairs of nodes so that we may compute the marginals on all edges $(i, j) \in E(G)$ as shown in Equation 2.29.

Designate an arbitrary node $r \in V$ as the root.
foreach $c \in \mathrm{child}(r)$ **do**
    call CollectEvidence($c \to r$)
foreach $c \in \mathrm{child}(r)$ **do**
    call DistributeEvidence($r \to c$)

Collect/Distribute Evidence and MPP

- All messages obey the message passing protocol.

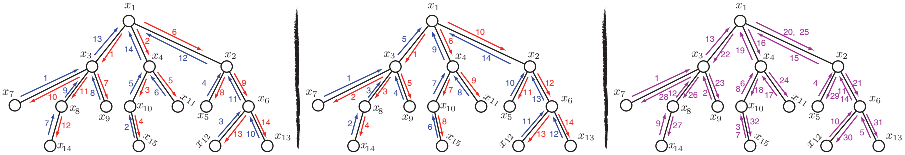
Collect/Distribute Evidence and MPP

- All messages obey the message passing protocol.
- At the collect evidence stage, a message is not sent to a node's (single) parent until it has received messages from all its children, so there is only one node it has not yet received a message from, namely the parent.

Collect/Distribute Evidence and MPP

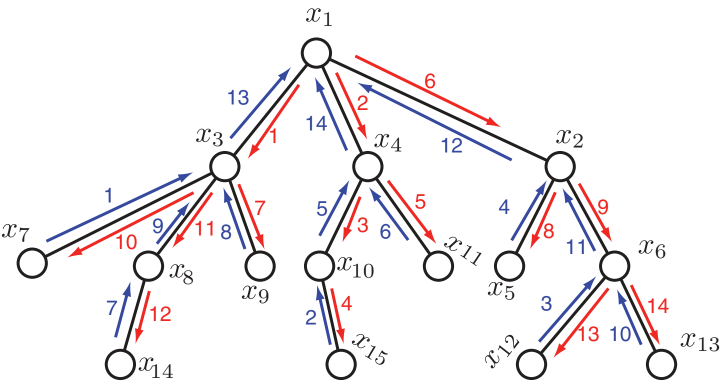
- All messages obey the message passing protocol.
- At the collect evidence stage, a message is not sent to a node's (single) parent until it has received messages from all its children, so there is only one node it has not yet received a message from, namely the parent.
- At the distribute evidence stage, once a node has received a message from its parent, it has received a message from all of its neighbors (since it received a message from all its children earlier, during the collect evidence phase) so it is free to send a message to any child that it likes.

Collect/Distribute Evidence



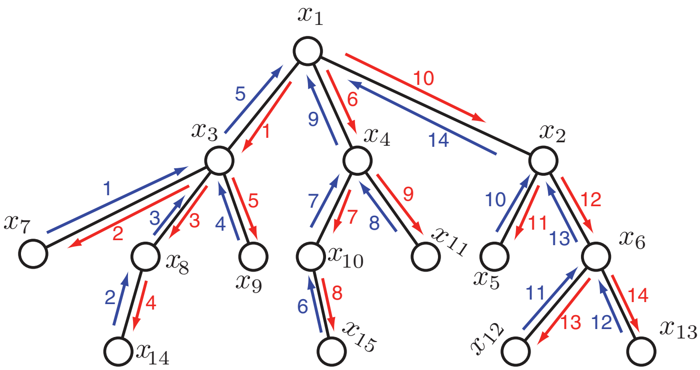
- Pictures show messages to compute all edge queries.
- Blue arrows indicate messages towards the root (node 1)
- Red arrow indicate messages away from the root.
- The numbers next to each arrow indicate the order of the message.
- Messages abide by MPP? Correspond to collect/distribute evidence?
- We'll next zoom into each one ...

Collect/Distribute Evidence



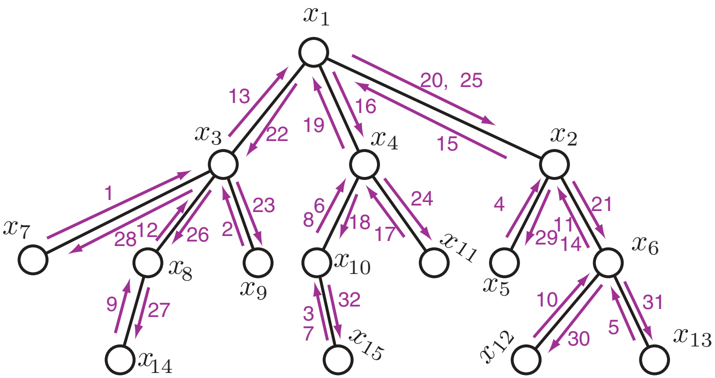
- Picture shows messages to compute all edge queries.
- Blue arrows indicate messages towards the root (node 1)
- Red arrow indicate messages away from the root.
- The numbers next to each arrow indicate the order of the message.
- Messages abide by MPP? Correspond to collect/distribute evidence?

Collect/Distribute Evidence



- Picture shows messages to compute all edge queries.
- Blue arrows indicate messages towards the root (node 1)
- Red arrow indicate messages away from the root.
- The numbers next to each arrow indicate the order of the message.
- Messages abide by MPP? Correspond to collect/distribute evidence?

Collect/Distribute Evidence



- Picture shows messages to compute all edge queries.
- Blue arrows indicate messages towards the root (node 1)
- Red arrow indicate messages away from the root.
- The numbers next to each arrow indicate the order of the message.
- Messages abide by MPP? Correspond to collect/distribute evidence?

Collect/Distribute Evidence

- Why called Collect/Distribute **Evidence**??

Collect/Distribute Evidence

- Why called Collect/Distribute **Evidence**?? Evidence is implicit via the delta (or generalized delta) functions.

Collect/Distribute Evidence

- Why called Collect/Distribute **Evidence**?? Evidence is implicit via the delta (or generalized delta) functions.
- The marginals we obtain really are $p(x_i, x_j, \bar{x}_E)$

Collect/Distribute Evidence

- Why called Collect/Distribute **Evidence**?? Evidence is implicit via the delta (or generalized delta) functions.
- The marginals we obtain really are $p(x_i, x_j, \bar{x}_E)$
- Note also that $p(x_i, \bar{x}_i) = \delta(x_i, \bar{x}_i)p(\bar{x}_i)$.

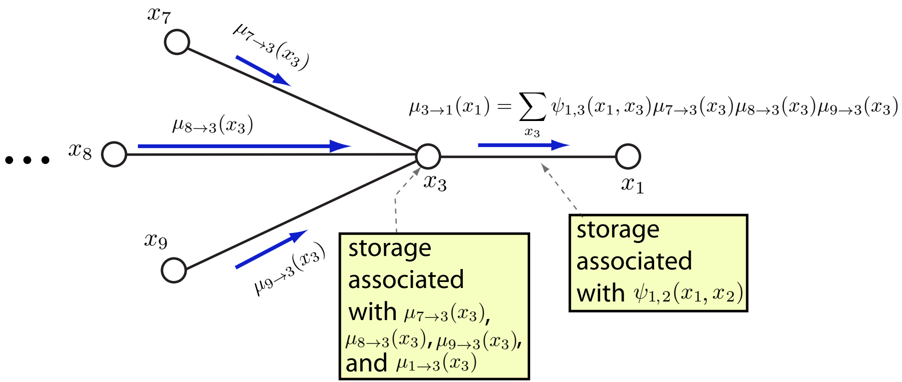
Collect/Distribute Evidence

- Why called Collect/Distribute **Evidence**?? Evidence is implicit via the delta (or generalized delta) functions.
- The marginals we obtain really are $p(x_i, x_j, \bar{x}_E)$
- Note also that $p(x_i, \bar{x}_i) = \delta(x_i, \bar{x}_i) p(\bar{x}_i)$.
- easy to obtain conditionals $p(x_i, x_j | \bar{x}_E)$

Collect/Distribute Evidence

- Why called Collect/Distribute **Evidence**?? Evidence is implicit via the delta (or generalized delta) functions.
- The marginals we obtain really are $p(x_i, x_j, \bar{x}_E)$
- Note also that $p(x_i, \bar{x}_i) = \delta(x_i, \bar{x}_i)p(\bar{x}_i)$.
- easy to obtain conditionals $p(x_i, x_j|\bar{x}_E)$
- Many orders possible: parallel implementations

Associated storage with message propagation



- for each edge (i, j), is storage associated with edge itself, $\psi_{i,j}(x_i, x_j)$, and all incoming messages, $\mu_{k\to i}(x_i)$ for all $k \in \delta(i) \setminus \{j\}$.
- $|E|(2r + r^2)$ total storage.
- Bad when $|\delta(i)|$ is large.

Alternative propagation styles

- Alternatively, incorporate in and then forget message as soon as it arrives
- Result of message would be new edge table:

$$\psi'_{i,j}(x_i, x_j) \leftarrow \psi_{i,j}(x_i, x_j)\mu_{k\to i}(x_i) \tag{2.30}$$

- Final factor, *after* incorporating all messages, has value $\psi'_{i,j}(x_i, x_j)$ where:

$$\psi'_{i,j}(x_i, x_j) = \psi_{i,j}(x_i, x_j) \prod_{k\in\delta(i)\setminus\{j\}} \mu_{k\to i}(x_i) \tag{2.31}$$

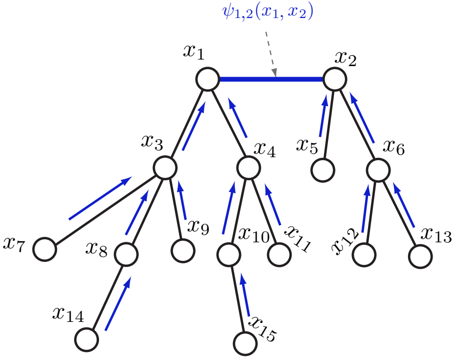
- Outgoing message to j depends only on the edge function, and becomes

$$\mu_{i\to j}(x_j) = \sum_{x_i} \psi'_{i,j}(x_i, x_j). \tag{2.32}$$

- Never require storage at only node i

Alternative propagation styles

- Never require storage at only nodes, only at edges.
- This can be good for certain queries. For example, for computing **just** $p(x_i)$, or $p(x_i, x_j)$ for $(i, j) \in E(G)$, this works out fine.



Alternative propagation styles

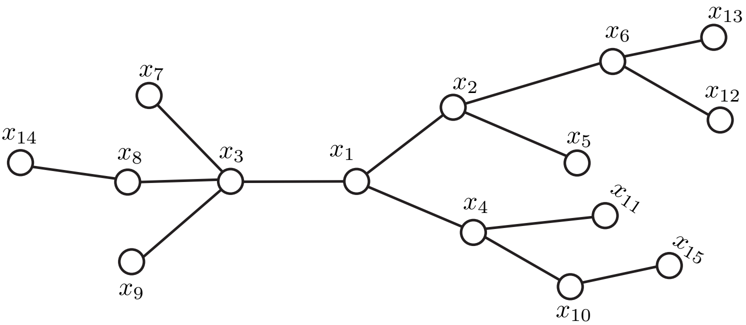
- Ultimately, messages will start arriving at x_j via nodes $k \in \delta(j) \setminus \{i\}$.

Alternative propagation styles

- Ultimately, messages will start arriving at x_j via nodes $k \in \delta(j) \setminus \{i\}$.
- Problem: Updated table no longer valid for sending message back to i and $\delta(i) \setminus \{j\}$.

Alternative propagation styles

- Ultimately, messages will start arriving at x_j via nodes $k \in \delta(j) \setminus \{i\}$.
- Problem: Updated table no longer valid for sending message back to i and $\delta(i) \setminus \{j\}$.



Alternative propagation styles

- Intuitively, we want to avoid double-counting the information sent from i to j, when a message is sent from j back to i — i (and the subtree rooted at i when the (i, j) edge is severed) already has that information, it doesn't need it again.

Alternative propagation styles

- Intuitively, we want to avoid double-counting the information sent from i to j, when a message is sent from j back to i — i (and the subtree rooted at i when the (i, j) edge is severed) already has that information, it doesn't need it again.
- Mathematically, from the elimination perspective, this would be equivalent to squaring the marginal functions after they have been constructed (i.e., ϕ^2 rather than ϕ).

Alternative propagation styles

- Intuitively, we want to avoid double-counting the information sent from i to j, when a message is sent from j back to i — i (and the subtree rooted at i when the (i, j) edge is severed) already has that information, it doesn't need it again.
- Mathematically, from the elimination perspective, this would be equivalent to squaring the marginal functions after they have been constructed (i.e., ϕ^2 rather than ϕ).
- $\therefore$ need somehow to divide out first set of messages before sending back, but can't do that if lost that info.

Alternative propagation styles

- Intuitively, we want to avoid double-counting the information sent from i to j, when a message is sent from j back to i — i (and the subtree rooted at i when the (i, j) edge is severed) already has that information, it doesn't need it again.
- Mathematically, from the elimination perspective, this would be equivalent to squaring the marginal functions after they have been constructed (i.e., ϕ^2 rather than ϕ).
- $\therefore$ need somehow to divide out first set of messages before sending back, but can't do that if lost that info.
- We still want to keep the node storage bounded regardless of node degree in tree.

Alternative propagation styles

- Solution 1: divide out the outgoing message from an edge as soon as it is ready, when it comes back it is multiplied back in and counted one time.
- During the first phase of message passing (e.g., collect evidence) we re-define our message definition as follows:

Algorithm 4: First phase message update $\mu_{i \to j}(x_j)$

$\mu_{i \to j}(x_j) = \sum_{x_i} \psi_{i,j}(x_i, x_j) \prod_{k \in \delta(i) \setminus \{j\}} \mu_{k \to i}(x_i)$; /* message as normal */

$\psi'_{i,j}(x_i, x_j) \leftarrow \psi_{i,j}(x_i, x_j) / \mu_{i \to j}(x_j)$; /* table update - divide outgoing message out */

if j *is not the root* **then**

    Let $k \in \delta(j)$ be the neighbor of j towards the root ;

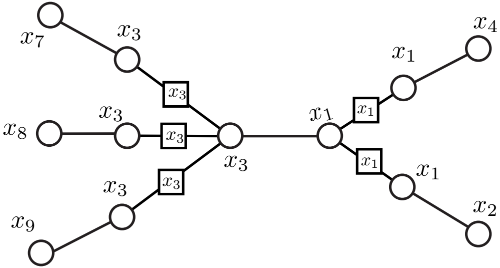
    $\psi'_{j,k}(x_j, x_k) \leftarrow \psi_{j,k}(x_j, x_k) \mu_{i \to j}(x_j)$; /* table update - multiply in incoming message */

Alternative propagation styles

- By dividing out $\mu_{i\to j}(x_j)$ from $\psi_{i,j}(x_i, x_j)$, we are sure that the $\mu_{i\to j}(x_j)$ will not be double counted once it is multiplied back in from the message coming back from k in $\mu_{k\to j}(x_j)$.
- when root has received all messages, start propagating messages towards leaves using standard message definition.
- No longer valid to send multiple messages along an edge in same direction
- new scheme is asymmetric, different message definitions during the collect vs. the distribute evidence phase of message passing.

Alternative propagation styles

- Solution 2: maintain distinct node separator functions



- every pair of edges that shares a common node has an extra node potential (shown as a square node) corresponding to that common node.
- common node separates tree into two separate sub-trees.
- edge $(7, 3)$ and $(3, 1)$ share the common node 3 and so there is a distinct square x_3 node corresponding to the edge pair $((7, 3), (3, 1))$ and separator potential function $\phi_{7,3,1}(x_3)$.

Alternative propagation styles

- Use only two extra tables per separator (square) node $i \in V$, which store incoming messages at i
- The two tables $\phi^n_{ijk}(x_j)$ (new) and $\phi^p_{ijk}(x_j)$ (previous) at each separator node, which keeps track of incoming messages.
- At start, initialize both tables to unity $\phi^n_{ijk}(x_j) = 1$, $\phi^p_{ijk}(x_j) = 1$ $\forall x_j \in \mathsf{D}_{X_j}$.
- Always update "new" table and divide out previous. Once "new" is used, it becomes "previous".
- we follow the collect/distribute evidence schedule for sending messages

Alternative propagation styles

Algorithm 5: collect evidence message update $\mu_{i \to j}(x_j)$

1 $\phi^n_{i,j,k}(x_j) = \sum_{x_i} \psi_{i,j}(x_i, x_j)$; /* message as normal stored in node */

2 $\psi_{j,k}(x_j, x_k) \leftarrow \psi_{j,k}(x_j, x_k) \frac{\phi^n_{i,j,k}(x_j)}{\phi^p_{i,j,k}(x_j)}$; /* update (j,k) edge potential. */

- At this point, step 2 same as $\psi_{j,k}(x_j, x_k) \leftarrow \psi_{j,k}(x_j, x_k) \phi^n_{i,j,k}(x_j)$
- we must ensure that there is no double counting of $\phi_{i,j,k}(x_j)$ when we do the distribute evidence phase, which is given in the next messages for the distribute evidence phase of the algorithm.

Alternative propagation styles

Algorithm 6: distribute evidence message update $\mu_{i \to j}(x_j)$

1 $\phi^n_{i,j,k}(x_j) = \sum_{x_i} \psi_{i,j}(x_i, x_j)$; /* message as normal stored in node */

2 $\psi_{j,k}(x_j, x_k) \leftarrow \psi_{j,k}(x_j, x_k) \frac{\phi^n_{i,j,k}(x_j)}{\phi^p_{i,j,k}(x_j)}$; /* update (j, k) edge potential. */

- Line 2 is where the double counting is avoided, divide out the previous separator potential table when we update the (j, k) edge function.
- General principle: at destination, multiply in new, and divide out old.
- uniform message style, and can once again send multiple messages along an edge if we want. ☺
- If divide same cost as multiply, less compute than previous style. ☺
- On the other hand, once again more storage, even more than originally!! ☹

Alternative propagation styles

Algorithm 7: collect evidence message update $\mu_{i \to j}(x_j)$

$\phi_{i,j,k}(x_j) = \sum_{x_i} \psi_{i,j}(x_i, x_j)$; /* message as normal stored in node */
$\psi_{j,k}(x_j, x_k) \leftarrow \psi_{j,k}(x_j, x_k)\phi_{i,j,k}(x_j)$; /* update (j,k) edge potential. */

Algorithm 8: asymmetric distribute evidence message update $\mu_{i \to j}(x_j)$

foreach $x_j \in \mathsf{D}_{X_j}$ **do**
  $\phi_{i,j,k}(x_j) \leftarrow \frac{1}{\phi_{i,j,k}(x_j)} \sum_{x_i} \psi_{i,j}(x_i, x_j)$; /* message as normal stored in node */
  $\psi_{j,k}(x_j, x_k) \leftarrow \psi_{j,k}(x_j, x_k)\phi_{i,j,k}(x_j)$; /* update (j,k) edge potential. */

- One table per separator.
- Recovered some storage ☺ but lost uniformity ☹ and multiple message sends ☹.

Three propagation styles

- The three different message styles we have described are called, respectively, the Shenoy-Shafer, the Lauritzen-Speigelhalter, and the Hugin message passing strategies.
- Normally given w.r.t. a junction tree (which we have not yet defined)
- Style of message can have practical consequences in an implementation.

Sources for Today's Lecture

- Most of this material comes from the reading handout `tree_inference.pdf`